BACnet® TESTING LABORATORIES ADDENDA

Addendum fix3 to BTL Test Package 18.1

Revision 6
Revised 10/26/2021

Approved by the BTL Working Group on September 23, 2021.
Approved by the BTL Working Group Voting Members on October 12, 2021.
Published on October 29, 2021.

BACnet is a registered trademark of ASHRAE. ASHRAE does not endorse, approve or test products for compliance with ASHRAE standards. Compliance of listed products to the requirements of ASHRAE Standard 135 is the responsibility of BACnet International. BTL is a registered trademark of BACnet International.

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

FOREWORD

The purpose of this addendum is to present current changes being made to the BTL Test Package. These modifications are the result of change proposals made pursuant to the continuous maintenance procedures and of deliberations within the BTL-WG Committee. The changes are summarized below.

In the following document, language to be added to existing clauses within the BTL Test Package 18.1 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

In contrast, changes to BTL Specified Tests also contain a yellow highlight to indicate the changes made by this addendum. When this addendum is applied, all highlighting will be removed. Change markings on tests will remain to indicate the difference between the new test and an existing 135.1 test. If a test being modified has never existed in 135.1, the applied result should not contain any change markings. When this is the case, square brackets will be used to describe the changes required for this test.

Each addendum can stand independently unless specifically noted via dependency within the addendum. If multiple addenda change the same test or section, each future released addendum that changes the same test or section will note in square brackets whether or not those changes are reflected.

BTL-18.1-fix3-1: Add DNET to Reject Messages in BTL 10.6.3 [BTLWG-531]

Overview:
BTLWG-531 states that the optional DNET Parameter has to be expected in the Reject-Message-To-Network Messages emitted from the IUT

Changes:

Test Changes

[Modify BTL-10.6.3 Ignore Router Commands in BTL Specified Tests]

10.6.3 Ignore Router Commands
Reason for Change: Changed test to support a Reject or a discard per Addendum 12.0d-4. Added DNET field to Reject Messages.

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

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

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

Test Steps:

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

(RECEIVE
DESTINATION = TD,
SOURCE = IUT,
Reject-Message-To-Network
DNET = (any valid value)
Rejection-Reason = 0 (other) | 3 (unknown))

BTL-18.1-fix3-2: Acked_Transitions Test for Latching LifeSafety Objects [BTLWG-1132]

Overview:
As noted in CR-0500, latching Life Safety objects are not always able to transition through all states without acknowledgements of the states as they occur.

This work item resolves this by providing an alternate test to use for such objects.

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

[Modify the 7.3.1.11 entry in AE-ACK-B Base Requirements, section 5.5.1]

~~**BTL - 7.3.1.11 - Acked_Transitions Tests**~~ ***BTL - 7.3.1.11.1 - Acked_Transitions Test***		
	Test Conditionality	~~Must be executed.~~ *If the IUT only supports event generating objects which latch their event state and require an acknowledgement before unlatching, this test shall be skipped.* *Only life safety objects are allowed to latch in this manner.*
	Test Directives	
	Testing Hints	

[Add an entry for 7.3.1.11.2 immediately after the entry for 7.3.1.11.1 in AE-ACK-B Base Requirements, section 5.5.1]

BTL - 7.3.1.11.2 - Acked_Transitions Test for Latching Objects		
	Test Conditionality	If the IUT does not support event generating objects which latch their event state and require an acknowledgement before unlatching, this test shall be skipped. Only life safety objects are allowed to latch in this manner.
	Test Directives	Apply the test for each supported transition.
	Testing Hints	

Test Changes

[Renumber and rename test 7.3.1.11 Acked_Transitions Tests to 7.3.1.11.1 Acked_Transitions Test]

7.3.1.11.*1* Acked_Transitions ~~Tests~~ *Test*

[Add new clause 7.3.1.11]

7.3.1.11 Acked_Transitions Tests

[Add a new test into BTL Specified Tests]

7.3.1.11.2 Acked_Transitions Test for Latching Objects

Reason for Change: No test exists for this functionality.

Purpose: To verify that the Acked_Transitions property tracks the acknowledgment state for a transition type.

Test Concept: This test is a single transition test for latching life safety objects which are not able to perform the regular Acked_Transitions test for all transitions. An object, O1, in the IUT is made to generate a transition which requires an acknowledgement. The Acked_Transitions property is verified that the corresponding flag is cleared (set to FALSE). The transition is acknowledged, and the flag is verified to have been set back to TRUE.

Configuration Requirements: O1 is configured to generate events and to require acknowledgements for the transition being tested. O1 should have no event transitions which have outstanding acknowledgements.

Test Steps:

1. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
2. MAKE (O1 transition)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (PI1: any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (T1: any valid time stamp),
   'Notification Class' = (NC1: the class corresponding to the object being tested),
   'Priority' = (PRIO1: the value configured to correspond to the transition type),
   'Event Type' = (E1: any valid event type),
   'Message Text' = (optional, any valid message text),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = S1,
   'To State' = S2,
   'Event Values' = (values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentState = S2
7. IF S2 is NORMAL THEN
   VERIFY Acked_Transitions = (TRUE, TRUE, FALSE)
   VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
   ELSE IF S2 is FAULT THEN
   VERIFY Acked_Transitions = (TRUE, FALSE, TRUE)
   VERIFY pStatusFlags = (TRUE, TRUE, ?, ?)
   ELSE
   VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
   VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = PI1,
   'Event Object Identifier' = O1,
   'Event State Acknowledged' = S2,
   'Time Stamp' = T1,
   'Acknowledgement Source' = (a character string),
   'Time of Acknowledgment' = (the TD’s current time)
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = PI1,
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = T1,
    'Notification Class' = NC1,
    'Priority' = PRIO1,
    'Event Type' = E1,
    'Message Text' = (optional, any valid message text),

'Notify Type' = ACK_NOTIFICATION,
'To State' = S2

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = PI1,
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = T1,
'Notification Class' = NC1,
'Priority' = PRIO1,
'Event Type' = E1,
'Message Text' = (optional, any valid message text),
'Notify Type' = ACK_NOTIFICATION

11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

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.

BTL-18.1-fix3-3: Fix Access Zone Object Supports Writable Out_Of_Service Property [BTLWG-681_1149]

Overview:
The “BTL Test Plan-18.1.2.final” requires to execute the Test “BTL - 7.3.1.1.X1” if the Access_Zone Object Supports Writable Out_Of_Service Property. This Test will fail in Step 6 because Property Present_Value is not present in the BACnet Object Access_Zone.

This work item replaces
BTL - 7.3.1.1.X1 - Out_Of_Service, Status_Flags, and Reliability Tests
with
BTL - 7.3.1.1.X3 - Out_Of_Service, Status_Flags, and Reliability test for Objects without Present_Value

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

3.45 Access Zone Object

3.45.2 Supports Writable Out_Of_Service Property

The Out_Of_Service property in Access Zone objects contained in the IUT are writable.

~~**BTL - 7.3.1.1.X1 - Out_Of_Service, Status_Flags, and Reliability Tests**~~ ***BTL - 7.3.1.1.X3 - Out_Of_Service, Status_Flags, and Reliability test for Objects without Present_Value***		
	Test Conditionality	Must be executed.
	Test Directives	The test shall be executed using an Access Zone object
	Testing Hints	

Test Changes

None

BTL-18.1-fix3-4: Fix APDU_Length Test [BTLWG-1057]

Overview:

Test 7.3.2.X62.5 APDU_Length Test is calculating the wrong size and needed to be fixed. There was also some clarification within the SSCP which is added as well.

Changes:

[In BTL Test Plan 18.1 add test directives in chapter 3.56.1 for 7.3.2.X62.5]

BTL - 7.3.2.X62.5 - APDU_Length Test		
	Test Conditionality	Must be executed.
	Test Directives	If the IUT supports data links with different allowable APDU lengths, run this test at least twice where the calculated maximum APDU length would be different.
	Testing Hints	

[In BTL Specified Tests, Change test 7.3.2.X62.5]

7.3.2.X62.5 APDU_Length Test

Reason for Change: New test per Addendum 135-2012ai.

Purpose: To verify that the Device object~~s~~ does not report a Max_APDU_Length_Accepted that is ~~not~~ 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.

1. MAX_APDU = ~~655360~~
2. REPEAT NP = (all configured and enabled Network Port objects with a
   Protocol_Level of BACNET_APPLICATION) {
   IF NP.APDU_Length < 50 THEN
   ERROR "APDU_Length must not be less than 50."
   IF NP.APDU_Length > (the maximum allowable for the Network_Type) THEN
   ERROR "APDU_Length is too large for the connected Network_Type"
   IF MAX_APDU ~~><~~ NP.APDU_Length THEN
   MAX_APDU = NP.APDU_Length
   }
3. VERIFY (Device, 4194303), Max_APDU_Length_Supported <= MAX_APDU

Note for 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.

BTL-18.1-fix3-5: Fix 14.YY.1.2.2 Malformed BVLC Test [BTLWG-1159, CR-0504, BTLWG-1164, CR-0507]

Overview:

CR-0504 asked if Test 14.YY.1.2.2 Malformed BVLC Test was too restrictive and if a valid response in step 11 could also be PAYLOAD_EXPECTED. The BTL agreed and the test is changed.

Per CR-507, steps 5 and 8, the Error Code expected is "absent or OTHER". However, there are more specific Error Codes that would be more applicable to these test cases, such as HEADER_ENCODING_ERROR.

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

None

Test Changes

[In the Test section, change tracking marks the changes **to 135.1** that need to be made. Where the test has never been included in 135.1, there should be not ~~strikethrough~~ or *italics*. When these changes are applied to the BTL Specified Tests document, the strikethrough and italics will be kept. They yellow highlight will be removed.
This allows testers to see how the BTL Specified Test version of a test differs from the 135.1 version of the test.]

[Modify test 14.YY.1.2.2 Malformed BVLC Test]

14.YY.1.2.2 Malformed BVLC Test

Reference YY.3.1.5

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

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

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

Test Steps:

-- Invalid BVLC function
1. TRANSMIT
   'BVLC Function' = (IV: an invalid 1-octet value),
   'Message ID' = (M1: any valid value),
   -- 'Originating Virtual Address' absent
   -- 'Destination Virtual Address' absent
   -- 'Destination Options' absent
   -- 'Data Options' absent
2. RECEIVE BVLC-Result,
   'Message ID' = M1,
   -- 'Originating Virtual Address' absent

-- 'Destination Virtual Address' absent
'Destination Options' = (absent or a valid list of options),
-- 'Data Options' absent
'Result for BVLC Function' = IV, -- the supplied invalid BVLC function from the request
'Result Code' = X'01', -- NAK
'Error Header Marker' = X'00', -- not a header option problem
'Error Class' = COMMUNICATION,
'Error Code' = BVLC_FUNCTION_UNKNOWN

3. CHECK(that the IUT did not process nor forward the request)

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

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

5. RECEIVE BVLC-Result,
'Message ID' = M2,
-- 'Originating Virtual Address' absent
'Destination Virtual Address' = D3
'Destination Options' = (absent or a valid list of options),
-- 'Data Options' absent
'Result for BVLC Function' = X'08', -- Disconnect-Request
'Result Code' = X'01', -- NAK
'Error Header Marker' = X'00', -- not a header option problem
'Error Class' = (COMMUNICATION or SERVICES),
'Error Code' = (HEADER_ENCODING_ERROR, INCONSISTENT_PARAMETER, PARAMETER_OUT_OF_RANGE or OTHER)
-- allowable combinations for the above 'Error Class' and 'Error Code' parameters are:
-- COMMUNICATION/HEADER_ENCODING_ERROR,
-- COMMUNICATION/OTHER,
-- SERVICES/INCONSISTENT_PARAMETERS
-- SERVICES/PARAMETER_OUT_OF_RANGE
-- SERVICES/OTHER

6. CHECK(that the IUT did not process the request)

-- Inclusion of a 'Destination Virtual Address when it is required to be absent

7. TRANSMIT Disconnect-Request,
'Message ID' = (M3: any valid value),
-- 'Originating Virtual Address' absent,
'Destination Virtual Address' = (IUT's VMAC),
'Destination Options' = (absent or a valid list of options),
-- 'Data Options' absent

8. RECEIVE BVLC-Result,
'Message ID' = M3,
-- 'Originating Virtual Address' absent
-- 'Destination Virtual Address' absent
'Destination Options' = (absent or a valid list of options),
-- 'Data Options' absent
'Result for BVLC Function' = X'08', -- Disconnect-Request
'Result Code' = X'01', -- NAK
'Error Header Marker' = X'00', -- not a header option problem
'Error Class' = (COMMUNICATION or SERVICES),
'Error Code' = (HEADER_ENCODING_ERROR, INCONSISTENT_PARAMETER, PARAMETER_OUT_OF_RANGE or OTHER)
-- allowable combinations for the above 'Error Class' and 'Error Code' parameters are:
-- COMMUNICATION/HEADER_ENCODING_ERROR,
-- COMMUNICATION/OTHER,
-- SERVICES/INCONSISTENT_PARAMETERS

```
        --          SERVICES/PARAMETER_OUT_OF_RANGE
        --          SERVICES/OTHER
9.      CHECK(that the IUT did not process the request)

-- A truncated message
10.     TRANSMIT Encapsulated-NPDU,
            'Message ID' =                  (M4: any valid value),
            'Originating Virtual Address' = (OVA: absent, or D3 if IUT is configured as a hub),
            'Destination Virtual Address' = (IUT's VMAC),
            -- 'Destination Options' absent
            -- 'Data Options' absent
            -- no NPDU included in the message
11.     RECEIVE BVLC-Result,
            'Message ID' =                  M4,
            -- 'Originating Virtual Address' absent
            'Destination Virtual Address' = OVA,
            'Destination Options' =         (absent or a valid list of options),
            -- 'Data Options' absent
            'Result for BVLC Function' =    X'01',    -- Encapsulated-NPDU
            'Result Code' =                 X'01',    -- NAK
            'Error Header Marker' =         X'00',    -- not a header option problem
            'Error Class' =                 COMMUNICATION,
            'Error Code' =                  MESSAGE_INCOMPLETE | PAYLOAD_EXPECTED
12.     CHECK(that the IUT did not process the request)

-- A message with extra octets added on
13.     TRANSMIT Disconnect-Request,
            'Message ID' =                  (M5: any valid value),
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            -- 'Destination Options' absent
            -- 'Data Options' absent
            (extra octets) =                ({ X'E1',           --a bunch of octets that look like valid data options.
                                            X'BF0003000012',
                                            X'3F0003000034'})
14.     RECEIVE BVLC-Result,
            'Message ID' =                  M5,
            -- 'Originating Virtual Address' absent
            -- 'Destination Virtual Address' absent
            'Destination Options' =         (absent or a valid list of options),
            -- 'Data Options' absent
            'Result for BVLC Function' =    X'08',    -- Disconnect-Request
            'Result Code' =                 X'01',    -- NAK
            'Error Header Marker' =         X'00',    -- not a header option problem
            'Error Class' =                 COMMUNICATION,
            'Error Code' =                  UNEXPECTED_DATA
15.     CHECK(that the IUT did not process the request)

-- A truncated broadcast message
16.     TRANSMIT Encapsulated-NPDU,
            DESTINATION =                   GLOBAL_BROADCAST,
            'Message ID' =                  (M6: any valid value),
            -- 'Originating Virtual Address' absent,
            'Destination Virtual Address' = (local broadcast VMAC),
            -- 'Destination Options'  absent
            'Data Options' =                ({ X'41'  -- 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)
```

BTL-18.1-fix3-6: Fix Inconsistencies in Verify Checklist [BTLWG-812]

Overview:
During the generation of the Test Package history spreadsheet when we moved to TP 18, a number of inconsistencies in the application of Verify Checklist, Verify EPICS and No Specific Test were noted.

In preparing this work item, all existing Verify Checklist, Verify EPICS and No Specific Test entries were reviewed for correctness of the type (Verify Checklist, Verify EPICS vs No Specific Test) and the language used in the Test Directives.

A new verification type, Verify Test Selection, was added as the existing types of checks did not apply in those cases.

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

[Modify section 3.41.1 of the BTL Test Plan]

3.41.1 Base Requirements

Base requirements must be met by any IUT that can contain Pulse Converter objects.

…

Verify EPICS		
	Test Conditionality	Must be executed.
	Test Directives	Verify in the EPICS that Update_Time and Count_Change_Time ~~in the object~~ are read-only *in all Pulse Converter objects*.
	Testing Hints	

[Modify section 3.51.5 of the BTL Test Plan]

3.51.5 Supports Port_Filter Property

The IUT contains the Port_Filter property. This is a required capability for Notification Forwarder objects if the IUT is a router.

135.1-2019 - 7.3.2.30.10 - Port_Filter Test		
	Test Conditionality	Must be executed.
	Test Directives	Test each port value, if the IUT is a router.
	Testing Hints	
Verify EPICS		
	Test Conditionality	Must be executed.
	Test Directives	*Verify that* ~~Every~~ *every* Notification Forwarder object *contains* ~~shall contain~~ a Port_Filter property if the IUT is a router. *Verify that every Port_Filter property is marked as writable* ~~This property shall be writable if present~~, though neither the size of the array nor the Port_ID portion of the BACnetPortPermission entries shall be modifiable via writes to this property. *Verify that the* ~~The~~ number of entries in the array shall match the number of BACnet ports currently defined in the device.
	Testing Hints	

[Modify section 3.57.3 of the BTL Test Plan]

3.57.3 Supports Writable Present_Value while Out_Of_Service is FALSE

The Present_Value property in a Timer object is writable while Out_Of_Service is FALSE.

Verify EPICS		
	Test Conditionality	*Must be executed.*
	Test Directives	*Verify that Present_Value property is marked as conditionally writable in at least one Timer object.*
	Testing Hints	
Verify EPICS		
	Test Conditionality	Must be executed.
	Test Directives	Verify that Min_Pres_Value and Max_Pres_Value properties are present in ~~the object~~ *every Timer object which has a conditionally writable Present_Value*.
	Testing Hints	

[Modify sections 5.1.2, 5.1.3, 5.21.3, 5.21.4, 5.31.3, 5.31.4, 6.5.4, of the BTL Test Plan]

…

~~**No Specific Test**~~ ***Verify EPICS***

…

[Modify section 5.2.5 of the BTL Test Plan]

5.2.5 Implements Intrinsic Alarming

The IUT contains, or can be made to contain, an object other than an Event Enrollment object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications.

Verify ~~Checklist~~ *Test Selection*		
	Test Conditionality	
	Test Directives	~~This~~ *Ensure this* functionality ~~will be~~ *is* tested *on non-Event Erollment obejcts* by the clause 8.4 or 8.5 algorithm tests listed later in this section.
	Testing Hints	

[Modify section 5.2.6 of the BTL Test Plan]

5.2.6 Supports the Event Enrollment Object

The IUT contains, or can be made to contain an Event Enrollment object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications.

Verify ~~Checklist~~ *Test Selection*		
	Test Conditionality	
	Test Directives	Ensure this functionality is tested on Event Enrollment objects by the clause 8.4 or 8.5 algorithm tests listed later in this section.
	Testing Hints	

[Modify section 5.3.4 of the BTL Test Plan]

5.3.4 Supports the Event Enrollment Object

The IUT contains, or can be made to contain an Event Enrollment object, that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims support for the Event Enrollment object in the Checklist. ~~The functionality will be tested by the clause 8.4 or 8.5 algorithm tests listed later in this section.~~
	Testing Hints	

Verify Test Selection		
	Test Conditionality	*Must be executed.*
	Test Directives	*Ensure this functionality is tested on Event Enrollment objects by the clause 8.4 or 8.5 algorithm tests listed later in this section.*
	Testing Hints	

[Modify section 5.22.3 of the BTL Test Plan]

5.22.3 Implements Intrinsic Alarming

The IUT contains, or can be made to contain, an object other than an Event Enrollment object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications.

Verify ~~Checklist~~ *Test Selection*		
	Test Conditionality	Must be executed.
	Test Directives	*Ensure this* ~~This~~ functionality *is tested on non-Event Enrollment objects* ~~will be tested~~ by the clause 8.4 or 8.5 algorithm tests listed later in this section.
	Testing Hints	

[Modify section 5.24.1 of the BTL Test Plan]

5.24.1 Base Requirements

Base requirements must be met by any IUT claiming conformance to this BIBB.

…

Verify ~~Checklist~~ *EPICS*		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims ~~initiate~~ support for *initiation of* both Unconfirmed*EventNotification* and Confirmed-Event-Notification services.
	Testing Hints	

…

[Modify section 5.25.2 of the BTL Test Plan]

5.25.2 Supports DS-WP-B

The IUT supports the Write Property service for its Recipient_List in Notification Class or Notification Forwarder objects.

Verify Checklist ~~or BTL - 9.22.1.X2 - Writing to Properties Based on Data Type~~		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims support for DS-WP-B option ‘contains writable list properties’.
	Testing Hints	
Verify Test Selection		
	Test Conditionality	*Must be executed.*
	Test Directives	*Verify that for Test Plan section 4.6.5, test BTL - 9.22.1.X2 - Writing to Properties Based on Data Type, is applied to Recipient_List properties in each supported notification object type (Notification Forwarder and Notification Class object types).*
	Testing Hints	

[Modify section 5.25.5 of the BTL Test Plan]

5.25.5 Supports Writable Recipient_List Property in Notification Forwarder Objects

The IUT must support a writable Recipient_List property in its Notification Forwarder objects to claim this BIBB.

Verify ~~Checklist~~ *EPICS*		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the Recipient_List property of each Notification Forwarder is marked as writable in the EPICS. ~~Verify that the 'Supports Writable Recipient_List Property' is checked in the Notification Forwarder Object section.~~
	Testing Hints	
Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify device contains 1 or more Notification Forwarder objects.
	Testing Hints	

[Modify section 5.26.2 of the BTL Test Plan]

5.26.2 Supports DM-LM-A for Subscribed_Recipients Property

The IUT must support the DM-LM-A services for the Subscribed_Recipients property of the Notification Forwarder object.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify *the IUT claims support for the* DM-LM-A *option "Supports adding and removing Notification Forwarder / Subscribed_Recipients entries"*~~for the Subscribed_Recipients property is supported~~.
	Testing Hints	

[Modify section 6.1.3 of the BTL Test Plan]

6.1.3 Supports DM-OCD-A

The IUT supports DM-OCD-A for the Timer, Calendar and Schedule objects.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims support for the ~~BIBB~~ DM-OCD-A and that ~~the~~ *creation and deletion of* Calendar and Schedule objects is claimed.~~can be created and deleted.~~ If the IUT claims Protocol_Revision 17 or higher, verify that the creation and deletion of Timer objects is also claimed.
	Testing Hints	

[Modify section 6.7.1 of the BTL Test Plan]

6.7.1 Base Requirements

Base requirements must be met by any IUT claiming conformance to this BIBB.

…

Verify ~~Checklist~~ *EPICS*		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT does not claim Weekly_Schedule or Exception_Schedule *properties are* ~~to be~~ writable in the EPICS.
	Testing Hints	

…

[Modify sections 9.10.4, 9.10.5, 9.10.6, 10.9.1, 10.10.1, 10.11.1 of the BTL Test Plan]

…

~~**VERIFY**~~ ***Verify*** **Checklist**

…

[Modify section 10.5.2 of the BTL Test Plan]

[Note that the heading for this section is incorrectly formatted in BTL Test Plan -- it is not auto-numbered]

10.5.2 Is able to Configure Network Port Objects

The IUT is able to create, delete, read and modify Network Port objects.

135.1-2019 - 8.18.3 - Reading and Presenting Properties		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test for <u>all</u> standard properties of the Network Port object, excluding the Object_Identifier and Object_Type properties.
	Testing Hints	
135.1-2019 - 8.22.4 - Accepting Input and Modifying Properties		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test for <u>all</u> standard properties of the Network Port object, excluding the Object_Identifier and Object_Type properties. Also exclude any properties that are required to be read-only by the BACnet standard. Repeat the test for a variety of values that cover the range of values required by the "Minimum Writable Value Ranges" table in the DS-M-A BIBB definition.
	Testing Hints	
Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims support for DM-OCD-A *and that creation and deletion of* ~~for~~ Network Port objects *is claimed*.
	Testing Hints	

Test Changes

None. The test changes that were here are in section 11 below.

BTL-18.1-fix3-7: Configurable Device Name [BTLWG-916]

Overview:

Add testing to ensure that the device name and instance are configurable.

Changes:
[Change clause **BTL Checklist-18.1_final– 3**, Page No. 5]

Device Object		
X	R	Base Requirements
	R	Supports configurable device name and instance
	C[1]	Supports Database_Revision property
	BTL-C[2]	Supports Time_Synchronization_Recipients
	BTL-C[2]	Supports UTC_Time_Synchronization_Recipients
	O	Contains a writable Local_Date property
	O	Contains a writable Local_Time property
	O	Supports UTC_Offset property
[1] Required if the device implements protocol revision 2 or higher. [2] The BTL requires that if the device implements protocol revision 7 or higher, then either both, or none of Time_Synchronization_Recipients or UTC_Time_Synchronization_Recipients is supported.		

[Insert new clause **BTL Test Plan-18.1.Final - 3.10.2**, re-number remaining clauses : 3.10.2 to 3.10.8]

3.10.2 Support Configurable Device Name and Instance

Ensures that the Object_Name and Object_Identifier properties are configurable.

BTL-7.3.2.10.X3 - Ensure Device Object_Name is Configurable		
	Test Conditionality	Must be executed
	Test Directives	
	Testing Hints	
BTL-7.3.2.10.X4 - Ensure Device Object_Identifier is Configurable		
	Test Conditionality	Must be executed
	Test Directives	
	Testing Hints	

[Added clause **BTL Specified Tests-18.1_final.pdf - 7.3.2**, Page No. 86]

7.3.2.10.X3 Ensure Device Object_Name is Configurable

Reason for Change: New test to standard.

Purpose: This test verifies Device Object_Name value is configurable as per BACnet Clause 12.1.1.4.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object_Name
2. MAKE (Configure the device Object_Name to a value other than S)
3. VERIFY (Device, IUT), Object_Name <> S

7.3.2.10.X3 Ensure Device Object_Identifier is Configurable

Reason for Change: New test to standard.

Purpose: This test verifies Device Object_Identifier property value is configurable as per BACnet Clause 12.1.1.3.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object_Identifier
2. MAKE (Configure the device Object_Identifier to a value other than S)
3. VERIFY (Device, IUT), Object_Identifier <> S

BTL-18.1-fix3-8: Allow Test 9.23.1.7 to Use Any Object [BTLWG-955, CR-0013]

Overview:

In response to CR-0013, the BTL-WG agreed that test 9.23.1.7 should be allowed to be performed against any object, not just the device object.

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

None

Test Changes

[Modify the Test Steps of test 9.23.1.7 from 135.1-2019, and move into BTL Specified Tests]

9.23.1.7 Writing Maximum Multiple Properties

[Reason for Change: the test should not mandate that the writable property be in the device object]

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be written to it using a single WritePropertyMultiple request, *to object O1.*

Test Concept: A writable property is written to an object in the IUT as many times as can be conveyed in the largest request accepted by the IUT. The calculation of the maximum request size shall be based on the IUT's Max_APDU_Length_Accepted and maximum segments per request.

The procedure to determine the number of values to use is:

MaxAPDU = IUT's Max_APDU_Length_Accepted
MaxRxSegs = IUT's maximum segments accepted per request
MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4
SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6
ObjIdSize = size of (an Object-Identifier) = 5
TagsSize = size of (an open and a close tag) = 2

PropIdSize = size of (chosen property Id) = depends on property ID and includes ARRAY INDEX size if required
ValueSize = size of (chosen property value) = depends on property and value chosen

If the IUT does not support receiving segmented requests:
NumPropertiesToWrite =
(MaxAPDU – NonSegRqstHdrSize – ObjIdSize – TagsSize) / (PropIdSize + TagsSize + ValueSize) =
(MaxAPDU – 11) / (PropIdSize + 2 + ValueSize)

If the IUT does support receiving segmented requests:
NumPropertiesToWrite =

((MaxAPDU – SegRqstHdrSize) * MaxRxSegs – ObjIdSize – TagsSize) / PropIdSize =
((MaxAPDU – 6) * MaxRxSegs – 7) / 2

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = (~~Device, X~~*O1*),
   'Property Identifier' = P1,
   'Priority Array Index' = A1, -- only if required
   'Property Value' = V1,
   …
   'Property Identifier' = P1,
   'Priority Array Index' = A1, -- only if required
   'Property Value' = V1
2. RECEIVE Simple-ACK
3. VERIFY ~~(P1 = V1)~~*O1, P1 = V1*

BTL-18.1-fix3-9: Allowable Time Granularity [BTLWG-961, CR-0329]

Overview:

12:01.55 needs to be equal to 12:01 when the device's internal resolution does not support the level of granularity that was written.

CR-0329 determined that it was acceptable for a device to round time values in a Schedule object to the nearest minute if the device's time granularity was 1 minute per the example given. The test referenced in the CR is BTL 9.22.1.X2:

Changes:

BTL Checklist Changes

[There are no checklist changes]

BTL Test Plan Changes

[No changes to test plan]

Test Changes

[Add new section 6.X6 into BTL Specified Tests to specify a new clause for 135.1-2019]

6.X6 Test Execution Considerations
There are some implicit test considerations which apply to all tests.

6.X6.1 Value Comparisons
Value comparisons, such as those in conditionals and statements (6.1.1 and 6.2) should always take into account the resolution constraints of 4.4.2, such that if any value is written with a higher or finer granularity than the IUT supports, any and all subsequent comparisons to that written value shall not result in the test step failing once the constraints have been applied to the value from the testing device.
Any value from the IUT is compared to a value from the testing device, the comparison shall not fail as long as the values are identical once the resolution constraints are applied to the external value. Devices may either truncate or round values written/received at a finer resolution than they claim in 4.4.2. This should be consistent across all values of that datatype.

Floating point values may be stored in the IUT using a different precision than what the value is encoded with from the testing device. Comparisons to these types of values should always consider that the IUT can round or truncate them, and that the value may also lose precision when stored, although the IUT must always present the stored value using full precision on a subsequent read.

Example: An IUT has a property with type REAL which has a resolution of 0.5. If a TD writes 0.75, it is acceptable for the IUT to present, on a subsequent read, either 1.0 or 0.5 for the property value so long as it is consistent.

6.X6.2 Functional Expectations
Because devices may store information in truncated or rounded format due to internal limitations, they may not behave exactly as all test cases prescribe. In these cases, it is acceptable for the IUT to present or act on information based on the granularity supported by the IUT. For example, an IUT may be configured with a schedule which begins at 12:01:01, but since the device only supports granularity of one minute, the scheduled behavior may begin at 12:01:00 or 12:02:00.

6.X6.3 Complex Datatypes
Since all complex datatypes can be broken down into one or more primitive datatypes, value comparisons and functional expectations from the above sections shall apply to each of the components that comprise a complex datatype. When there

are relationships present within the fields of a complex datatype, the IUT shall continue to meet the functional expectations between all fields of complex datatypes when applying any rounding or truncation.

BTL-18.1-fix3-10: Removed

Removed. Changes combined with BTL-18.1-fix3-5 above.

BTL-18.1-fix3-11: Allow Test 8.20.5.2 To Write to Multiple Objects [BTLWG-926, CR-0331]

Overview:

Changes according to CR-0331 and additional changes due to issues observed in the process.

Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

None

Test Changes

[Modify 135.1-2019 8.20.5.2 in BTL Specified Tests]

8.20.5.2 Fallback to ReadProperty on Reject - UNRECOGNIZED_SERVICE Response

Reason for Change: Modified to allow use of RPM and use of multiple objects.

BACnet Reference Clauses: 15.5 and 15.7

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the TD returns a Reject-PDU and a Reject Reason of UNRECOGNIZED_SERVICE.

Test Concept: The IUT is configured to send the TD a ReadPropertyMultiple request to access one or more properties of one or more objects. The TD responds with a Reject-PDU and a Reject Reason of UNRECOGNIZED_SERVICE. With no additional configuration, the IUT sends one or more ReadProperty requests to the TD, where each ReadProperty request specifies an individual property from the original ReadPropertyMultiple request covering all the properties from the original ReadPropertyMultiple request.

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it attempts to acquire values from the TD using the ReadPropertyMultiple service without first interrogating the TD's Protocol_Services_Supported property. If the IUT cannot be configured in this way then this test shall be omitted.

Test Steps:

1. *MAKE (the IUT send a ReadPropertyMultiple-Request for a list of properties L)*
2. RECEIVE ReadPropertyMultiple-Request,
   -- the following is repeated for each object O referenced in L
   'Object Identifier' = (*O~~object indentifier of the specified object~~*),
   'List of Property References' = (one or more properties of *O as specified in L ~~the specified object~~*)
3. TRANSMIT BACnet-Reject-PDU,
   'Reject Reason' = UNRECOGNIZED_SERVICE
4. REPEAT X = (the properties from *L in the order the IUT chooses ~~Step 1~~*) DO {
   RECEIVE ReadProperty-Request,
   'Object Identifier' = (object identifier referenced by X),
   'Property Identifier' = (property identifier referenced by X)
   TRANSMIT ReadProperty-Ack,
   'Object Identifier' = (object identifier referenced by X),
   'Property Identifier' = (property identifier referenced by X),
   'Property Value' = (any valid value)
   }