BACnet® TESTING LABORATORIES ADDENDA

Addendum Fix to BTL Test Package 23.0

Revision final
Revised 5/16/2023

Approved by the BTL Working Group on March 30, 2023.
Approved by the BTL Working Group Voting Members on May 12, 2023;
Published on May 18, 2023.

[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 23.0 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-23.0 Fix-1: Test 9.X36.2.2 should not include DM-DDA-A [BTLWG-1377]

Overview:

This test should not be in DM-DDA-A as it only applies to the device being configured, not to the device directing the configuration. Renaming to “Only Accepts Configuration When Received Parameters Match” to reduce the chance of misconstruing the purpose.

Changes:

Checklist Changes

None

Test Plan Changes

8.31 Device Management - Dynamic Device Assignment - A

8.31.1 Base Requirements

Base requirements must be met by any IUT that can claim this BIBB.

BTL - 9.X35.1 - Uses Who-Is to Configure Devices Supporting the Who-Am-I Service		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
~~**BTL - 9.X36.2.2 - Only Configures When Sent Parameters Match**~~		
	~~**Test Conditionality**~~	~~Must be executed.~~
	~~**Test Directives**~~	
	~~**Testing Hints**~~	
BTL - 8.X36.3 - Can Unconfigure Devices		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

8.32 Device Management - Dynamic Device Assignment - B

8.32.1 Base Requirements

Base requirements must be met by any IUT that can claim this BIBB.

Verify EPICS		
	Test Conditionality	Must be executed.
	Test Directives	Verify the Device Object has the Serial_Number property required to initiate Who-Am-I and execute You-Are.
	Testing Hints	
BTL - 9.X36.2.2 - *Only Accepts Configuration When Received Parameters Match*~~Only Configures When Sent Parameters Match~~		
	Test Conditionality	This test shall be skipped if the IUT is an MS/TP subordinate node.
	Test Directives	
	Testing Hints	
BTL - 8.X35.1 - Responds to Who-Is With Who-Am-I While in the Unconfigured State		
	Test Conditionality	If the IUT does not support having a MAC address but no configured Device object instance number, this test shall be skipped.
	Test Directives	
	Testing Hints	

BTL - 9.X36.1.6 - Retains Configuration Through Restarts		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 9.X36.1.7 - Unconfigurable by You-Are		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

Specified Test Changes

[Rename test 9.X36.2.2]

9.X36.2.2 *Only Accepts Configuration When Received Parameters Match*~~Only Configures When Sent 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)

BTL-23.0 Fix-2: Fix 7.3.2.15.X8 Life Safety Member_Of Test [BTLWG-1239, CR-0522]

Overview:

Jira item BTLWG-1239. CR-0522 pointed out a number issues with the test. The test should:

- be renamed to match the purpose test concept
- add in testing of accepted object type in references (only Life Safety Zone)
- test steps changed to reflect test concept

Changes:

Checklist Changes

None

Test Plan Changes

[Replace all references to 7.3.2.15.X8 Support Writable Member_Of Property with the new test of the same number but different name: 7.3.2.15.X8 Writable Member_Of Property Test]

Specified Test Changes

[Replace existing test 7.3.2.15.X8 Writable Member_Of Property with the following test (with the new name)]

7.3.2.15.X8 Writable Member_Of Property Test

Reason for Change: No test exists for this functionality.

BACnet Reference Clauses: 12.15.29, and 12.16.29

Purpose: To verify that a writable Member_Of property of a LifeSafety Point object only accepts references to life safety zone objects.

Test Concept: Write a local Life Safety Zone reference to Member_Of and verify that it is accepted. Write a remote Life Safety Zone reference to Member_Of and verify that it is accepted or rejected with the correct error code. Verify that the property does not accept writes of other object types.

Test Steps:

-- verify that the property accepts local references
1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (O1),
   'Property Identifier' = Member_Of
   'Property Value' = (X: any valid local life safety zone object reference)
2. RECEIVE Simple-ACK-PDU,
3. VERIFY Member_Of = X

-- verify that the property accepts remote references, or returns the correct error code if it does not
4. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (O1),
   'Property Identifier' = Member_Of
   'Property Value' = (X: any valid remote life safety zone object reference)
5. RECEIVE Simple-ACK-PDU |
   BACnet-Error-PDU,
   'Error Class' = PROPERTY,

'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED

-- verify that the property does not accept references to other object types

4. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (O1),
   'Property Identifier' = Member_Of
   'Property Value' = (X: any local non-life safety zone object reference)
5. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

6. IF the IUT accepted the previously written remote reference THEN {
   TRANSMIT WriteProperty-Request,
   'Object Identifier' = (O1),
   'Property Identifier' = Member_Of
   'Property Value' = (X: any local non-life safety zone object reference)
   RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

BTL-23.0 Fix-3: Add Missing Stop_Time Test in Test Plan [BTLWG-1380]

Overview:

The T-VMT-I-B, AE-EL-I-B and AE-EL-E-B sections in the Test Plan include 7.3.2.24.3 but T-VMMV-I-B is missing this test.

Changes:

Checklist Changes

None

Test Plan Changes

7.7.7 Supports Start_Time and Stop_Time Properties

The IUT can be made to start and stop logging using these properties.

If present these properties are required to be writable.

135.1-2019 - 7.3.2.24.2 - Start_Time Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.24.3 - Stop_Time Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.2.X6 - DateTime Non-Pattern Properties Test		
	Test Conditionality	Must be executed if the IUT claims Protocol_Revision 11 or greater.
	Test Directives	Apply to the Start_Time and again to the Stop_Time properties in a Trend Log object.
	Testing Hints	
BTL - 9.23.2.X11 - DateTime Non-Pattern Properties Test using WritePropertyMultiple Service		
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher and which supports execution of WritePropertyMultiple.
	Test Directives	Apply to the Start_Time and again to the Stop_Time properties in a Trend Log object.
	Testing Hints	

Specified Test Changes

None

BTL-23.0 Fix-4: Section Removed from Addenda

BTL-23.0 Fix-5: Update Tests in section 8.4 to reference correct Status_Flags Property [BTLWG-1180, CR-0509]

Overview:

Correct the various 8.4 tests to use the Status_Flags instead of the pStatusFlags property for Event_Enrollment Objects.

Changes:

Checklist Changes

None

Test Plan Changes

[Modify the below highlighted references]

5.2.8 Implements the CHANGE_OF_STATE Algorithm

The IUT contains, or can be made to contain, an object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications with an Event_Type of CHANGE_OF_STATE.

~~135.1-2019~~*BTL* - 8.4.2 - CHANGE_OF_STATE Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATE.
	Testing Hints	
135.1-2019 - 8.5.2 - CHANGE_OF_STATE Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATE.
	Testing Hints	

5.2.12 Implements the FLOATING_LIMIT Algorithm

The IUT contains, or can be made to contain, an object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications with an Event_Type of FLOATING_LIMIT.

~~135.1-2019~~*BTL* - 8.4.5 - FLOATING_LIMIT Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of FLOATING_LIMIT.
	Testing Hints	
135.1-2019 - 8.5.5 - FLOATING_LIMIT Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of FLOATING_LIMIT.
	Testing Hints	

5.2.13 Implements the OUT_OF_RANGE Algorithm

The IUT contains, or can be made to contain, an object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications with an Event_Type of OUT_OF_RANGE.

~~135.1-2019~~*BTL* - 8.4.6 - OUT_OF_RANGE Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.

	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of OUT_OF_RANGE.
	Testing Hints	
135.1-2019 - 8.5.6 - OUT_OF_RANGE Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of OUT_OF_RANGE.
	Testing Hints	

5.2.24 Implements the CHANGE_OF_CHARACTERSTRING Algorithm

The IUT contains, or can be made to contain, an object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications with an Event_Type of CHANGE_OF_CHARACTERSTRING.

~~135.1-2019~~*BTL* - 8.4.13 - CHANGE_OF_CHARACTERSTRING Test (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_CHARACTERSTRING.
	Testing Hints	
135.1-2019 - 8.5.13 - CHANGE_OF_CHARACTERSTRING Test (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_CHARACTERSTRING.
	Testing Hints	

5.2.25 Implements the CHANGE_OF_STATUS_FLAGS Algorithm

The IUT contains, or can be made to contain, an object that can generate ConfirmedEventNotifications and UnconfirmedEventNotifications with an Event_Type of CHANGE_OF_STATUS_FLAGS.

~~135.1-2019~~*BTL* - 8.4.15 - CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATUS_FLAGS.
	Testing Hints	
135.1-2019 - 8.5.15 - CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATUS_FLAGS.
	Testing Hints	

5.2.26 Implements the UNSIGNED_RANGE Algorithm

The IUT contains, or can be made to contain, an object such as an Accumulator object, that can generate EventNotifications with an Event_Type of UNSIGNED_RANGE.

~~135.1-2019~~*BTL* - 8.4.14 - UNSIGNED_RANGE Test (ConfirmedEventNotification Test)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of UNSIGNED_RANGE.
	Testing Hints	
135.1-2019 - 8.5.14 - UNSIGNED_RANGE Test (UnconfirmedEventNotification)		

	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of UNSIGNED_RANGE.
	Testing Hints	

5.3.6 Implements the CHANGE_OF_STATE Algorithm

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

~~135.1-2019~~BTL - 8.4.2 - CHANGE_OF_STATE Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	
135.1-2019 - 8.5.2 - CHANGE_OF_STATE Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	

5.3.10 Implements the FLOATING_LIMIT Algorithm

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

~~135.1-2019~~BTL - 8.4.5 - FLOATING_LIMIT Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	
135.1-2019 - 8.5.5 - FLOATING_LIMIT Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	

5.3.11 Implements the OUT_OF_RANGE Algorithm

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

~~135.1-2019~~BTL - 8.4.6 - OUT_OF_RANGE Tests (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	
135.1-2019 - 8.5.6 - OUT_OF_RANGE Tests (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	

5.3.15 Implements the CHANGE_OF_CHARACTERSTRING Algorithm

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

~~135.1-2019~~*BTL* - 8.4.13 - CHANGE_OF_CHARACTERSTRING Test (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.

	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT
	Testing Hints	
135.1-2019 - 8.5.13 - CHANGE_OF_CHARACTERSTRING Test (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT
	Testing Hints	

5.3.16 Implements the CHANGE_OF_STATUS_FLAGS Algorithm

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

~~135.1-2019~~*BTL* - 8.4.15 - CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATUS_FLAGS.
	Testing Hints	
135.1-2019 - 8.5.15 - CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_STATUS_FLAGS.
	Testing Hints	

5.3.17 Implements the UNSIGNED_RANGE Algorithm

The IUT contains, or can be made to contain an Event Enrollment object that can generate EventNotifications with an Event_Type of UNSIGNED_RANGE.

~~135.1-2019~~*BTL* - 8.4.14 - UNSIGNED_RANGE Test (ConfirmedEventNotification Test)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	
135.1-2019 - 8.5.14 - UNSIGNED_RANGE Test (UnconfirmedEventNotification)		
	Test Conditionality	Must be executed.
	Test Directives	This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT.
	Testing Hints	

Specified Test Changes

[Modify test 8.4.2 in 135.1-2019]

8.4.2 CHANGE_OF_STATE Tests (ConfirmedEventNotification)

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

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

notification message. If the IUT claims conformance to Protocol_Revision 12 or lower, and a Multi-state Input or Multi-state Value object is being tested, the transition to and from the FAULT state is also tested.

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

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

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

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

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

1. VERIFY pCurrentState = NORMAL
2. IF ((Protocol_Revision is present AND Protocol_Revision ≥ 13)
   OR ((Protocol_Revision is present AND Protocol_Revision < 13)
   AND (pAlarmValues contains at least one value))) THEN {
   IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value from pAlarmValues)
   ELSE
   MAKE (pMonitoredValue have a value pAlarmValues)
   WAIT (pTimeDelay)
   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' =	(T1, any valid time stamp),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO_OFFNORMAL transition),
'Event Type' =	CHANGE_OF_STATE,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	pMonitoredValue, pStatusFlags

   TRANSMIT BACnet-SimpleACK-PDU
   IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
   VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
   VERIFY pCurrentState = OFFNORMAL
   IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = (T1, Ta, Tb)
   IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
   ELSE
   MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)

WAIT (pTimeDelay)
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' = (T2, any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pStatusFlags
TRANSMIT BACnet-SimpleACK-PDU
IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
VERIFY pCurrentState = NORMAL
IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (T1, Ta, T2)
}

3. IF ((Protocol_Revision is present AND Protocol_Revision < 13)
AND (intrinsic reporting is being tested)
AND (the intrinsic reporting object is configured with pFaultValues containing at least one values)) THEN {
IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value from pFaultValues)
ELSE
MAKE (pMonitoredValue have a value from pFaultValues)
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested),
'Time Stamp' = (Tfault: any valid timestamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_FAULT transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = pMonitoredValue, pStatusFlags
TRANSMIT BACnet-SimpleACK-PDU
VERIFY pCurrentState = FAULT
IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)
VERIFY pCurrentReliability = MULTI_STATE_FAULT
IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
ELSE
MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested),

'Time Stamp' = (Tnormal: any valid timestamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_NORMAL transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pStatusFlags
TRANSMIT BACnet-SimpleACK-PDU
VERIFY pCurrentState = NORMAL
IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)
}

Notes to Tester: The time stamps indicated by "Ta" and "Tb" can have a value that indicates an unspecified time or a time that precedes the timestamp T1.

[Modify test 8.4.5 in 135.1-2019]

8.4.5 FLOATING_LIMIT Tests (ConfirmedEventNotification)
Purpose: To verify the correct operation of the Floating Limit event algorithm.

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

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

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < (pSetpoint + pHighDiffLimit))
   ELSE
   MAKE (pMonitoredValue have a value x:
   (pSetpoint + pHighDiffLimit – pDeadband) < x < (pSetpoint + pHighDiffLimit))
3. WAIT (pTimeDelay + **Notification Fail Time**)
4. CHECK (verify that no notification message has been transmitted)
5. VERIFY pCurrentState = NORMAL
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value x: x > (pSetpoint + pHighDiffLimit))
   ELSE
   MAKE (pMonitoredValue have a value x: x > (pSetpoint + pHighDiffLimit))
7. WAIT (pTimeDelay)
8. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (any valid time stamp),

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

9. TRANSMIT BACnet-SimpleACK-PDU
10. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
11. VERIFY pCurrentState = pHighDiffLimit
12. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, *)
13. IF (pMonitoredValue is writable) THEN
    WRITE (pMonitoredValue) = (a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < pSetpoint + pHighDiffLimit))
    ELSE
    MAKE (pMonitoredValue have a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < pSetpoint + pHighDiffLimit))
14. WAIT (pTimeDelayNormal + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY pCurrentState = pHighDiffLimit
17. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))
    ELSE
    MAKE (pMonitoredValue have a value x:
    (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))
18. WAIT (pTimeDelayNormal)
19. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = FLOATING_LIMIT,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pHighDiffLimit,
20. TRANSMIT BACnet-SimpleACK-PDU
21. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
22. VERIFY pCurrentState = NORMAL
23. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
24. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit < x < (pSetpoint - pLowDiffLimit + pDeadband))
    ELSE
    MAKE (pMonitoredValue have a value x:(pSetpoint - pLowDiffLimit < x < (pSetpoint - pLowDiffLimit + pDeadband))
25. WAIT (pTimeDelay + **Notification Fail Time**)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY pCurrentState = NORMAL

28. IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = (a value x: x < (pSetpoint - pLowDiffLimit))
    ELSE
        MAKE (pMonitoredValue have a value x: x < (pSetpoint - pLowDiffLimit))
29. WAIT (pTimeDelay)
30. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object being tested),
            'Time Stamp' = (any valid time stamp),
            'Notification Class' = (the configured notification class),
            'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' = FLOATING_LIMIT,
            'Message Text' = (optional, any valid message text),
            'Notify Type' = EVENT | ALARM,
            'AckRequired' = TRUE | FALSE,
            'From State' = NORMAL,
            'To State' = LOW_LIMIT,
            'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pLowDiffLimit
31. TRANSMIT BACnet-SimpleACK-PDU
32. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
        VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
33. VERIFY pCurrentState = LOW_LIMIT
34. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35. IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit) < x < (pSetpoint - pLowDiffLimit + pDeadband))
    ELSE
        MAKE (pMonitoredValue have a value x: (pSetpoint - pLowDiffLimit) < x < (pSetpoint - pLowDiffLimit + pDeadband))
36. WAIT (pTimeDelayNormal + **Notification Fail Time**)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY pCurrentState = pLowDiffLimit
39. IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))
    ELSE
        MAKE pMonitoredValue have a value x: (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))
40. WAIT (pTimeDelayNormal)
41. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object being tested),
            'Time Stamp' = (any valid time stamp),
            'Notification Class' = (the configured notification class),
            'Priority' = (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' = FLOATING_LIMIT,
            'Message Text' = (optional, any valid message text),
            'Notify Type' = EVENT | ALARM,
            'AckRequired' = TRUE | FALSE,
            'From State' = LOW_LIMIT,
            'To State' = NORMAL,
            'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pLowDiffLimit
42. TRANSMIT BACnet-SimpleACK-PDU
43. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
        VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
44. VERIFY pCurrentState = NORMAL

45. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)

Notes to Tester: The time stamps indicated by "*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

[Modify test 8.4.6 in 135.1-2019]

8.4.6 OUT_OF_RANGE Tests (ConfirmedEventNotification)
Purpose: To verify the correct operation of the OUT_OF_RANGE event algorithm.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is raised to a value that is below but within pDeadband of the pHighLimit. At this point the object should still be in a NORMAL state. pMonitoredValue is raised to a value that is above the pHighLimit. After pTimeDelay expires the object should enter the HIGH_LIMIT state and transmit an event notification message. pMonitoredValue is lowered to a value that is below the pHighLimit but still within pDeadband of pHighLimit. The object should remain in the HIGH_LIMIT state. pMonitoredValue is lowered further to a normal value that is not within pDeadband of pHighLimit. After pTimeDelayNormal expires the object should enter the NORMAL state and issue an event notification. The same process is repeated to test pLowLimit.

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

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN
       WRITE pMonitoredValue = (a value x: (pHighLimit – pDeadband) < x < pHighLimit)
   ELSE
       MAKE (pMonitoredValue have a value x: (pHighLimit – pDeadband) < x < pHighLimit)
3. WAIT (pTimeDelay + **Notification Fail Time**)
4. CHECK (verify that no notification message has been transmitted)
5. VERIFY pCurrentState = NORMAL
6. IF (pMonitoredValue is writable) THEN
       WRITE pMonitoredValue = (a value x such x > pHighLimit)
   ELSE
       MAKE (pMonitoredValue have a value x: x > pHighLimit)
7. WAIT (pTimeDelay)
8. BEFORE **Notification Fail Time**
       RECEIVE ConfirmedEventNotification-Request,
           'Process Identifier' = (any valid process ID),
           'Initiating Device Identifier' = IUT,
           'Event Object Identifier' = (the object being tested),
           'Time Stamp' = (any valid time stamp),
           'Notification Class' = (the configured notification class),
           'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
           'Event Type' = OUT_OF_RANGE,
           'Message Text' = (optional, any valid message text),
           'Notify Type' = EVENT | ALARM,
           'AckRequired' = TRUE | FALSE,
           'From State' = NORMAL,
           'To State' = HIGH_LIMIT,
           'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pHighLimit
9. TRANSMIT BACnet-SimpleACK-PDU
10. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
        VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
11. VERIFY pCurrentState = HIGH_LIMIT
12. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 8, *, *)

13. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (pHighLimit – pDeadband)< x < pHighLimit)
    ELSE
    MAKE (pMonitoredValue have a value x: (pHighLimit – pDeadband)< x < pHighLimit)
14. WAIT (pTimeDelayNormal + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY pCurrentState = HIGH_LIMIT
17. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))
    ELSE
    MAKE (pMonitoredValue have a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))
18. WAIT (pTimeDelayNormal)
19. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pHighLimit
20. TRANSMIT BACnet-SimpleACK-PDU
21. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
22. VERIFY pCurrentState = NORMAL
23. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
24. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
    ELSE
    MAKE (pMonitoredValue have a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
25. WAIT (pTimeDelay + **Notification Fail Time**)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY pCurrentState = NORMAL
28. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x such x < pLowDiffLimit)
    ELSE
    MAKE (pMonitoredValue have a value x: x < pLowDiffLimit)
29. WAIT (pTimeDelay)
30. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Message Text' = (optional, any valid message text),
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = LOW_LIMIT,
    'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pLowDiffLimit

31. TRANSMIT BACnet-SimpleACK-PDU
32. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
        VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
33. VERIFY pCurrentState = LOW_LIMIT
34. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35. IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = (a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
    ELSE
        MAKE (pMonitoredValue have a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
36. WAIT (pTimeDelayNormal + **Notification Fail Time**)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY pCurrentState = LOW_LIMIT
39. IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = (a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))
    ELSE
        MAKE (pMonitoredValue have a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))
40. WAIT (pTimeDelayNormal)
41. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object being tested),
            'Time Stamp' = (any valid time stamp),
            'Notification Class' = (the configured notification class),
            'Priority' = (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' = OUT_OF_RANGE,
            'Message Text' = (optional, any valid message text),
            'Notify Type' = EVENT | ALARM,
            'AckRequired' = TRUE | FALSE,
            'From State' = LOW_LIMIT,
            'To State' = NORMAL,
            'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pLowDiffLimit
42. TRANSMIT BACnet-SimpleACK-PDU
43. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
        VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
44. VERIFY pCurrentState = NORMAL
45. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)

Notes to Tester: The time stamps indicated by "*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

[Modify test 8.4.13 in 135.1-2019]

8.4.13 CHANGE_OF_CHARACTERSTRING Test (ConfirmedEventNotification)

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

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

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

value of TRUE. The property and Event_Parameters element, respectively, represented by pAlarmValues shall be non-empty. The event-generating object shall be in a NORMAL state at the start of the test.

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

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

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

Test Steps:

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

'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_CHARACTERSTRING,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = OFFNORMAL,
'Event Values' = pMonitoredValue, pStatusFlags, pAlarmValues(matching list element)

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

'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = pMonitoredValue, pStatusFlags, (any characterstring)

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

}

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

[Modify test 8.4.14 in 135.1-2019]

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

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

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN

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

'Event Object Identifier' = (the object being tested),
'Time Stamp' = (Tlowlimit: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_OFFNORMAL transition),
'Event Type' = UNSIGNED_RANGE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = LOW_LIMIT,
'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit

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

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

[Modify test 8.4.15 in 135.1-2019]

8.4.15 CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification)
Purpose: To verify the correct operation of the CHANGE_OF_STATUS_FLAGS event algorithm.

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

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

Test Steps:

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

[Modify test 7.3.1.11.1 in BTL Specified Tests]

7.3.1.11.1 Acked_Transitions Test

…

2. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
3. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
   VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
4. IF (pMonitoredValue is writable) THEN

…
9. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
10. IF (Protocol_revision is present AND Protocol_Revision >= 13 THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
11. IF (pMonitoredValue is writable) THEN
…
16. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
17. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?,?)
18. IF (the event-triggering object can be placed into a fault condition) THEN {
…

[Modify test 8.4.3.1 in BTL Specified Tests]

8.4.3.1 Numerical Algorithm (ConfirmedEventNotification)
…
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
10. VERIFY pCurrentState = NORMAL
…

[Modify test 8.4.4 in BTL Specified Tests]

8.4.4 COMMAND_FAILURE Tests (ConfirmedEventNotification)
…
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
8. VERIFY pCurrentState = OFFNORMAL
…
13. TRANSMIT BACnet-SimpleACK-PDU
14. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
    VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
15. VERIFY pCurrentState = NORMAL
…

[Modify test 8.4.8.7 in BTL Specified Tests]

8.4.8.7 Mode Transition Tests when Event State is Maintained
…
Test Steps:
1. VERIFY Event_Detection_Enable = TRUE
2. CHECK (pCurrentState = NORMAL)
3. MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
4. IF (IUT supports another pMode value which maintains the NORMAL state) THEN {
~~4~~ MAKE (pMode = different value that maintains pCurrentState as NORMAL)
~~5~~ WAIT (pTimeDelayNormal)
~~6~~ BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' = (T1: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,

'Message Text' = (S1: optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected

~~7~~ TRANSMIT BACnet-SimpleACK-PDU
~~8~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
~~9~~ VERIFY pCurrentState = NORMAL
~~10~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (*, *, T1)
~~11~~ IF (Event_Message_Texts property exists) THEN
VERIFY Event_Message_Texts = (*, *, S1)

}

5. ~~12~~MAKE (pMonitoredValue have a value that corresponds to an OFFNORMAL state)
6. *VERIFY pCurrentState = OFFNORMAL*
7. *IF (IUT supports another pMode value which maintains the OFFNORMAL state) THEN {*

~~13~~ MAKE (pMode = different value that maintains pCurrentState as OFFNORMAL)
~~14~~ WAIT (pTimeDelay)
~~15~~ BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' = (T2: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,
'Message Text' = (S2: optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = OFFNORMAL,
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
~~16~~ TRANSMIT BACnet-SimpleACK-PDU
~~17~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
~~18.~~ VERIFY pCurrentState = OFFNORMAL
~~19.~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (T2, *, *)
~~20.~~ IF (Event_Message_Texts property exists) THEN
VERIFY Event_Message_Texts = (S2, *, *)

}

8. ~~21.~~MAKE (pMonitoredValue have a value that corresponds to a LIFE_SAFETY_ALARM state)
9. *IF (IUT supports another pMode value which maintains the LIFE_SAFETY_ALARM state) THEN {*

~~22.~~ MAKE (pMode = different value that maintains pCurrentState = LIFE_SAFETY_ALARM)
~~23.~~ WAIT (pTimeDelay)
~~24.~~ BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
'Time Stamp' = (T3: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,

'Message Text' = (S3: optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = OFFNORMAL,
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
~~25.~~ TRANSMIT BACnet-SimpleACK-PDU
~~26.~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE, ?, ?)
~~27.~~ VERIFY pCurrentState = OFFNORMAL
~~28.~~ IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
VERIFY Event_Time_Stamps = (T3, *, *)
~~29.~~ IF (Event_Message_Texts property exists) THEN
VERIFY Event_Message_Texts = (S3, *, *)
}

[Modify test 8.4.9 in BTL Specified Tests]

8.4.9 EXTENDED Test (ConfirmedEventNotification)
…
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, FALSE,?,?)
8. VERIFY pCurrentState = CS2
…
14. TRANSMIT BACnet-SimpleACK-PDU
15. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE, ?, ?)
16. VERIFY pCurrentState = NORMAL
…

[Modify test 8.4.17.X9.15 in BTL Specified Tests]

8.4.17.X9.15 CHANGE_OF_RELIABILITY with the FAULT_OUT_OF_RANGE Algorithm (ConfirmedEventNotification)
…
7. VERIFY pCurrentState = FAULT
8. VERIFY ~~pStatusFlags~~*Status_Flags* = (TRUE, TRUE,?,?)
9. IF (pMonitoredValue is writable) THEN
…
13. VERIFY pCurrentState = NORMAL
14. VERIFY ~~pStatusFlags~~*Status_Flags* = (FALSE, FALSE,?, ?)

BTL-23.0 Fix-6: Correct Test 7.3.2.X56.7 Lockout State [BTLWG-1212, CR-0516]

Overview:

An Access Point object may be set to a lockout state due to too many failed access attempts, as defined in the Max_Failed_Attempts property, or by writing TRUE to this property.

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

None

Test Changes

[Modify test BTL – 7.3.2.X56.7]

7.3.2.X56.7 Lockout State Test

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

Purpose: To verify that access is denied for any credential when the access point is in the lockout state. To verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state. To verify that the lockout will automatically relinquish after the specified time.

Test Concept: A credential which will result in denied access is repeatedly presented at the access point until the access point becomes locked out. When the access point becomes locked valid credentials will also be denied access until the lockout relinquish time has expired.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) The Max_Failed_Attempts property, if present, has a value greater than 0.
b) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
c) An active credential with no valid access rights for the access point shall be represented by Access Credential object C2.
d) The Failed_Attempts_Events list, if present, shall have at least one entry corresponding to the reason why C2 is denied access.
e) The Lockout_Relinquish_Time has a value greater than 0.

Test Steps:

-- verify that valid credentials are denied when the Lockout property is TRUE
1. WRITE Lockout = TRUE
2. WAIT **Internal Processing Fail Time**
3. VERIFY Access_Event = LOCKOUT_OTHER
4. VERIFY Access_Event_Time = (the time that TRUE was written to the Lockout property)
5. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
6. MAKE (present credential C1 at credential reader for this access point)

7. VERIFY Access_Event = DENIED_LOCKOUT
8. VERIFY Access_Event_Time = (the time that credential C1 was presented)
9. VERIFY Access_Event_Credential = C1

-- verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state

10. WRITE Lockout = FALSE
11. WAIT **Internal Processing Fail Time**
12. VERIFY Access_Event = LOCKOUT_RELINQUISH
13. VERIFY Access_Event_Time = (the time that FALSE was written to the Lockout property)
14. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
15. IF (Failed_Attempts and Max_Failed_Attempts are supported) THEN
    REPEAT X= (1 to Max_Failed_Attempts ~~+ 1~~) DO {
    READ FailedAttempts = Failed_Attempts
    MAKE (present credential C2 at credential reader for this access point)
    VERIFY (Failed_Attempts = FailedAttempts + 1)
    }
16. VERIFY (Lockout = TRUE)
17. VERIFY (Access_Event = LOCKOUT_MAX_ATTEMPTS)
18. VERIFY (Access_Event_Time = the time that Lockout was set to TRUE)
19. VERIFY (Access_Event_Credential = C2)
20. MAKE (present credential C1 at credential reader for this access point)
21. VERIFY (Access_Event = DENIED_LOCKOUT)
22. VERIFY (Access_Event_Time = the time that credential C1 was presented)
23. VERIFY (Access_Event_Credential = C1)

-- verify that the lockout will automatically relinquish after the specified time

24. WAIT Lockout_Relinquish_Time
25. VERIFY (Lockout = FALSE)
26. VERIFY (Access_Event = LOCKOUT_RELINQUISHED)
27. VERIFY (Access_Event_Time = the time that Lockout was set to FALSE)
28. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
29. MAKE (present credential C1 at credential reader for this access point)
30. VERIFY (Access_Event = GRANTED)
31. VERIFY (Access_Event_Time = the time that credential C1 was presented)
32. VERIFY (Access_Event_Credential = C1)

BTL-23.0 Fix-7: Add Missing Conditionality for Test 9.24.1.12 [BTLWG-1393, CR-0543]

Overview:

Not all devices supporting the DCC and RD services can initiate requests. Add conditionality.

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

8.14.6 Supports DM-RD-B

The IUT also supports the DM-RD-B BIBB.

135.1-2019 - 9.24.1.2 - Indefinite Time Duration Restored by ReinitializeDevice		
	Test Conditionality	If the IUT claims Protocol_Revision >= 20, this test shall be skipped. If the IUT does not support indefinite Time Duration, this test may be skipped.
	Test Directives	
	Testing Hints	
BTL - 9.24.1.5 - Finite Time Duration Restored by ReinitializeDevice		
	Test Conditionality	If the IUT claims Protocol_Revision >= 20, this test shall be skipped. If the IUT does not support an internal clock, this test may be skipped.
	Test Directives	
	Testing Hints	
135.1-2019 - 9.24.1.7 - Indefinite Time Duration, Disable-Initiation, Restored by ReinitializeDevice		
	Test Conditionality	If the IUT does not support indefinite Time Duration, this test shall be skipped.
	Test Directives	
	Testing Hints	
BTL - 9.24.1.12 - Disable of Service Initiation Restored by ReinitializeDevice		
	Test Conditionality	If the IUT does not support an internal clock, this test shall be skipped. *If the IUT does not initiate any services other than an I-Am in response to a Who-Is, then this test case shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 9.24.2.3 - Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'		
	Test Conditionality	If the IUT claims Protocol_Revision >= 20, this test shall be skipped.
	Test Directives	If the IUT does not support an internal clock this test shall be tested with indefinite time duration.
	Testing Hints	

Test Changes

None

BTL-23.0 Fix-8: Cleanup checklist footnotes for Data Link Layers IPv4 and IPv6 [BTLWG-1311]

Overview:

Data Link Layer IPv4 and IPv6 footnotes are confusing and need to be rewritten

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

[In BTL Checklist, modify checklist Data Link Layer – IPv4 and IPv6]

Data Link Layer - IPv4		
	R	Base Requirements
	C[1]	Is able to operate in Normal mode
	C[1]	Is able to operate in Foreign mode
	C[~~2~~1]	Is able to operate in BBMD mode
	O	Supports configuration through Network Port object
	O	Is able to initiate broadcast messages
	O	Supports Network Port objects and DHCP
	O	Supports Network Address Translation in BBMD mode
	BTL-C[~~3~~2]	*Supports NM-BBMDC-B*
	[1] Either BBMD or both Normal and Foreign modes are required. *[~~1~~] ~~Required if the device does not support BBMD mode.~~* *[~~2~~] ~~Required if the device does not support Foreign mode.~~* *[~~3~~2] Required if the device is able to operate in BBMD mode*	

Data Link Layer - IPv6		
	R	Base Requirements
	C[1]	Is able to operate in Normal mode
	C[1]	Is able to operate in Foreign mode
	C[~~2~~1]	Is able to operate in BBMD mode
	R	Supports configuration through Network Port object
	O	Supports DHCP
	BTL-C[~~3~~2]	*Supports NM-BBMDC-B*
	[1] Either BBMD or both Normal and Foreign modes are required. *[~~1~~] ~~Required if the device does not support BBMD mode.~~* *[~~2~~] ~~Required if the device does not support Foreign mode.~~* *[~~3~~2] Required if the device is able to operate in BBMD mode*	

Test Changes

None

BTL-23.0 Fix-9: Update Example for Test 9.21.1.3 [BTLWG-1355]

Overview:

The example provided in 9.21.1.3 could be interpreted to always include reading position #1 in the list.

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

[None]

Test Changes

[Modify test 9.21.1.3 in BTL Specified Tests]

9.21.1.3 Reading Items by Position with Negative Count

Reason for Change: Make the test applicable to object types other than trends. *Updated example to not read position #1.*

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a position and the number of items before that position to return.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in *the list property P* ~~the Log_Buffer~~. This range is specified using the 'By Position' option and a negative value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

Configuration Requirements: A list property, P, is configured with N items.

Test Steps:

1. TRANSMIT ReadRange-Request,
   'Object Identifier' = (the ~~log~~ object configured for this test),
   'Property Identifier' = ~~Log_Buffer~~ *P*,
   'Reference Index' = (any value x: $1 \le x \le N$ ~~Record_Count~~),
   'Count' = (any value y: $y < 0$ AND $|y| \le x$)
2. RECEIVE ReadRange-ACK,
   'Object Identifier' = (the ~~log~~ object configured for this test),
   'Property Identifier' = ~~Log_Buffer~~ *P*,
   'Result Flags' = {?, ?, FALSE},
   'Item Count' = |y|,
   'Item Data' = (all of the *items* specified ~~trend records~~ in order of increasing position. The items specified include the item at the index specified by x, plus |y|-1 items preceding.)

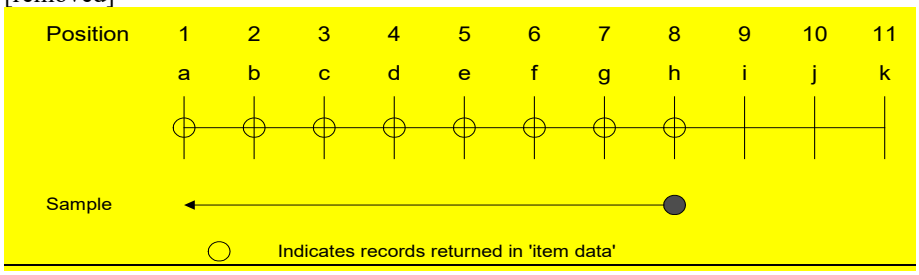
Test Example (using the sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,
   'Object Identifier' = (Trend Log, Instance 1),
   'Property Identifier' = Log_Buffer,
   'Reference Index' = 8,
   'Count' = ~~8~~ *-6*

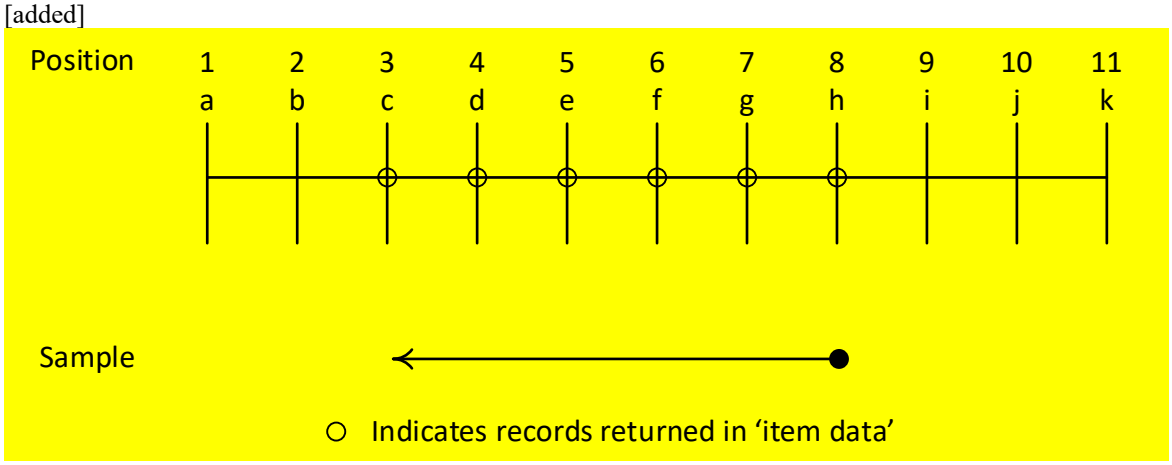
2. RECEIVE ReadRange-ACK,

'Object Identifier'	= (Trend Log, Instance 1),
'Property Identifier'	= Log_Buffer,
'Result Flags'	= {~~TRUE~~*FALSE*, FALSE, FALSE},
'Item Count'	= ~~8~~ *6*
'Item Data'	= Records < ~~a, b,~~ c, d, e, f, g, h > in that order.

[removed]



[added]



BTL-23.0 Fix-10: Update Test 12.X.2.1.5 Execute Forwarded-Address-Resolution [BTLWG-1230, CR-0520]

Overview:

BTL-CR-0520 formulates a change to test step 1. and an addition to the test configuration requirements of test "BTL-12.X.2.1.5 Execute Forwarded-Address-Resolution".

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

[None]

Test Changes

[Modify test 12.X.2.1.5 in BTL Specified Tests Revision 20.0.1]

[Modify test "BTL - 12.X.2.1.5 - Execute Forwarded-Address-Resolution".]

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

Reason for Change: ~~New to standard.~~ *BBMD will broadcast a Forwarded-Address-Resolution to a device which is not registered in it.*

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

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

Configuration Requirements: IUT and BBMD should be in the same domain.

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

BTL-23.0 Fix-11: Update Test 9.20.1.X2 ReadPropertyMultiple Array Properties [BTLWG-1329]

Overview:

As per BACnet clause 15.7 ReadPropertyMultiple Service, IUT could respond ACK with an error for single property results. refer: 9.20.2.1 Reading a Single, Unsupported Property from a Single Object. However, it is not permitted by the 9.20.1.X2 ReadPropertyMultiple Array Properties test case.

For the 9.20.1.X2 ReadPropertyMultiple Array Properties test case, we shall permit ACK with error for single property results by means of this proposal.

Changes:

BTL Checklist Changes

[None]

BTL Test Plan Changes

Test Changes

[Modify existing BTL Test]

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.

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

Test Steps:

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN
3. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Array Index' = 1
4. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Array Index' = 1,
   'Property Value' = (V, any valid value of the correct data type for property P1)
5. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Array Index' = X,
6. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Array Index' = X,

'Property Value' = (V, any valid value of the correct data type for property P1)

7. CHECK (V is any valid value of the correct data type for property P1)
8. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Array Index' = (X+1)
9. RECEIVE ReadPropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = INVALID_ARRAY_INDEX |
   (ReadPropertyMultiple-ACK,
   'Object Identifier' = (O1),
   'Property Identifier' = P1,
   'Error Class' = PROPERTY,
   'Error Code' = INVALID_ARRAY_INDEX)

BTL-23.0 Fix-12: Test 7.3.2.20.5 Multi-State Objects Writable State_Text but not Number_Of_States [BTLWG-1402, CR-0547]

Overview:

Replace one test with two tests that separately validate the writability of the value of the Number_Of_States and the size of the State_Text.

Changes:

BTL Checklist Changes

Multi-state Input Object		
	R	Base Requirements
	S	Supports configurable Out_Of_Service property
	S	Supports State_Text
	O	Supports resizable State_Text property
	O	*Supports writable Number_Of_States*
Multi-state Output Object		
	R	Base Requirements
	R	Supports command prioritization
	S	Supports configurable Out_Of_Service property
	S	Supports State_Text
	O	Supports resizable State_Text property
	O	*Supports writable Number_Of_States*
	O	Supports the value source mechanism.
Multi-state Value Object		
	R	Base Requirements
	S	Supports configurable Out_Of_Service property
	S	Supports State_Text
	O	Supports command prioritization
	O	Supports resizable State_Text property
	O	*Supports writable Number_Of_States*
	O	Supports the value source mechanism.

BTL Test Plan Changes

3.14.4 Supports Resizable State_Text Property

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a *Multi-state Input object with a* State_Text property that is resizable by writing to the array.

Verify Checklist		
	Test Conditionality	*Must be executed.*
	Test Directives	*Verify that the IUT claims support for DS-WP-B option 'Contains resizable array properties'*
	Testing Hints	
~~**135.1-2019 - 7.3.2.18.5 - Number_Of_States and State_Text Size Change Test**~~ ***BTL - 7.3.1.X110.1 - Resizable State_Text Test***		
	Test Conditionality	Must be executed if the IUT claims Protocol_Revision 4 or greater ~~and supports a resizable State_Text property~~.

	Test Directives	*This test shall be performed using a Multi-state Input object.*
	Testing Hints	

3.14.5Supports writable Number_Of_States

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a Multi-state Input object with a writable Number_Of_States property.

BTL - 7.3.1.X73.1 – Writable Number_Of_States Test		
	Test Conditionality	*Must be executed if the IUT claims Protocol_Revision 4 or greater.*
	Test Directives	*This test shall be performed using a Multi-state Input object.*
	Testing Hints	

3.14Multi-state Output Object

…

3.15.5Supports Resizable State_Text Property

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a *Multi-state Output object with a* State_Text property that is resizable by writing to the array.

Verify Checklist		
	Test Conditionality	*Must be executed.*
	Test Directives	*Verify that the IUT claims support for DS-WP-B option 'Contains resizable array properties'*
	Testing Hints	
~~**135.1-2019 - 7.3.2.19.6 - Number_Of_States and State_Text Size Change Test**~~ ***BTL – 7.3.1.X110.1 - Resizable State_Text Test***		
	Test Conditionality	Must be executed if the IUT claims Protocol_Revision 4 or greater ~~and supports a resizable State_Text property~~.
	Test Directives	*This test shall be performed using a Multi-state Output object.*
	Testing Hints	

3.15.6Supports writable Number_Of_States

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a Multi-state Output object with a writable Number_Of_States property.

BTL - 7.3.1.X73.1 - Writable Number_Of_States Test		
	Test Conditionality	*Must be executed if the IUT claims Protocol_Revision 4 or greater.*
	Test Directives	*This test shall be performed using a Multi-state Output object.*
	Testing Hints	

~~3.15.6~~*3.15.7* Supports the Value Source Mechanism

…

3.16Multi-state Value Object

…

3.16.5Supports Resizable State_Text Property

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a *Multi-state Value object with a* State_Text property that is resizable by writing to the array.

Verify Checklist		
	Test Conditionality	*Must be executed.*
	Test Directives	*Verify that the IUT claims support for DS-WP-B option 'Contains resizable array properties'*
	Testing Hints	

~~135.1-2019 - 7.3.2.20.5 - Number_Of_States and State_Text Size Change Test~~ *BTL - 7.3.1.X110.1 - Resizable State_Text Test*		
	Test Conditionality	Must be executed if the IUT claims Protocol_Revision 4 or greater ~~and supports a resizable State_Text property~~.
	Test Directives	*This test shall be performed using a Multi-state Value object.*
	Testing Hints	

3.16.6Supports writable Number_Of_States

The IUT is protocol revision 4 or higher and the IUT contains, or can be made to contain, a Multi-state Value object with a writable Number_Of_States property.

BTL - 7.3.1.X73.1 - Writable Number_Of_States Test		
	Test Conditionality	*Must be executed if the IUT claims Protocol_Revision 4 or greater.*
	Test Directives	*This test shall be performed using a Multi-state Value object.*
	Testing Hints	

~~3.16.6~~*3.16.7* Supports the Value Source Mechanism

…

Test Changes

[Add new Test]
7.3.1.X110.1 Resizable State_Text 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

[Add new Test]
7.3.1.X73.1 Writable Number_Of_States 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

[Remove Test 7.3.2.18.5]
~~**7.3.2.18.5 Number_Of_States and State_Text Size Change Test**~~
~~Dependencies: WriteProperty Service Execution Tests, 9.22~~

~~BACnet Reference Clauses: 12.18.11 and 12.18.12~~

~~Purpose: This test case verifies that when the value of the Number_Of_States property is changed, the size of the State_Text array is changed*, the value of the Number_Of_States property is changed* accordingly to the same size *and vice versa*. If the Number_Of_States and the size of the State_Text arrays cannot be changed, then this test shall not be performed. If Protocol_Revision is not present, or has a value less than 4, then this test shall not be performed.~~

~~Configuration Requirements: The IUT shall be configured with a Multi-state Input object *O1,* with writable Number_Of_States and a resizable State_Text arrays *array*.~~

~~Test Concept: Number_Of_States and the State_Text array are set to a certain size. They are then increased by writing the Number_Of_States, decreased by writing the State_Text array, increased by writing the State_Text array and decreased by writing Number_Of_States.~~

~~Test Steps:~~

~~1. TRANSMIT WriteProperty-Request,~~
~~'Object Identifier' = (the Multi-state Input object being tested),~~
~~'Property Identifier' = Number_Of_States,~~
~~'Property Value' = 2~~
~~2. RECEIVE Simple-ACK-PDU~~
~~3. VERIFY Number_Of_States = 2~~
~~4. VERIFY State_Text = 2, ARRAY INDEX = 0~~
~~5. TRANSMIT WriteProperty-Request,~~
~~'Object Identifier' = (the Multi-state Input object being tested),~~
~~'Property Identifier' = Number_Of_States,~~
~~'Property Value' = (some value greater than 2)~~

~~6. RECEIVE Simple-ACK-PDU~~
~~7. VERIFY Number_Of_States = (the value written in step 5)~~
~~8. VERIFY State_Text = (the value written in step 5), ARRAY INDEX = 0~~
~~9. TRANSMIT WriteProperty-Request,~~
~~'Object Identifier' = (the Multi-state Input object being tested),~~
~~'Property Identifier' = State_Text,~~
~~'Property Value' = (State_Text array of length 2)~~
~~10. RECEIVE Simple-ACK-PDU~~
~~11. VERIFY Number_Of_States = 2~~
~~12. VERIFY State_Text = 2, ARRAY INDEX = 0~~
~~13. TRANSMIT WriteProperty-Request,~~
~~'Object Identifier' = (the Multi-state Input object being tested),~~
~~'Property Identifier' = State_Text,~~
~~'Property Value' = (State_Text array of length greater than 2)~~
~~14. RECEIVE Simple-ACK-PDU~~
~~15. VERIFY Number_Of_States = (the length of the array written in step 13)~~
~~16. VERIFY State_Text = (the length of the array written in step 13), ARRAY INDEX = 0~~
~~17. TRANSMIT WriteProperty-Request,~~
~~'Object Identifier' = (the Multi-state Input object being tested),~~
~~'Property Identifier' = Number_Of_States,~~
~~'Property Value' = 2~~
~~18. RECEIVE Simple-ACK-PDU~~
~~19. VERIFY State_Text = (an array consisting of elements 1 & 2 from the array written in step 13)~~
~~20. VERIFY Number_Of_States = 2~~

[Remove Test 7.3.2.19.6]

~~**7.3.2.19.6 Number_Of_States and State_Text Size Change Test**~~

~~Dependencies: WriteProperty Service Execution Tests, 9.22~~

~~BACnet Reference Clauses: 12.19.11 and 12.19.12~~

~~The test to verify the Number_Of_States value and State_Text array size of Multi-state Output objects are defined in 7.3.2.18.5. Run the test using a Multi-state Output object.~~

[Remove Test 7.3.2.20.5]

~~**7.3.2.20.5 Number_Of_States and State_Text Size Change Test**~~

~~Dependencies: WriteProperty Service Execution Tests, 9.22~~

~~BACnet Reference Clauses: 12.20.10 and 12.20.11~~

~~Test Steps:~~

~~Tests to verify the Number_Of_States value and State_Text array size of Multi-state Value objects are defined in 7.3.2.18.5. Run the tests using a Multi-state Value object.~~

BTL-23.0 Fix-13: Add Missing Checklist Entries for 135-2020bv [BTLWG-1422, BTLWG-1228]

Overview:

Addendum 135-2020bv (PR24) introduced the Write_Every_Scheduled_Action property to the Schedule object. The introduction of this property also came with some required behavior which also applies even if the Write_Every_Scheduled_Action property is not present. As a result, the criteria for which testing must occur is more dependent on Protocol_Revision than the presence or absence of the Write_Every_Scheduled_Action property. Ultimately, the changes introduced in 135-2020bv will likely be incorporated into standard schedule testing, but for the short term, we need only to be concerned with the implementations at PR24 or higher containing a schedule object. (In other words, the solution proposed here is not intended to be the permanent one).

Changes:

BTL Checklist Changes

[In BTL Checklist, make changes to section 3 (Objects)]

Schedule Object		
	R	Base Requirements
	BTL-C[1]	Supports SCHED-I-B
	BTL-C[1]	Supports SCHED-WS-I-B
	BTL-C[1]	Supports SCHED-R-B
	O	Supports resizable Exception_Schedule property
	O[2]	*Protocol_Revision 24 or higher is claimed*
[1] You must support one of the listed scheduling BIBBs if your device contains a schedule object [2] *Contact BTL for interim tests for this object*		

BTL Test Plan Changes

[In BTL Test Plan, add entry to sections 3.19]

3.19.6 Protocol_Revision 24 or higher is claimed

The IUT supports a Schedule object and claims support for Protocol_Revision 24 or higher.

Contact BTL for interim tests for this functionality.

Test Changes

None

BTL-23.0 Fix-14: Test Plan Changes for WPM Testing Requirements [BTLWG-1394, CR-0545]

Overview:

Testplan Clause 4.7.26 tests that a IUT correctly writes constructed values that the IUT supports. This clause indicates all forms of constructed values should be tested including all forms of ABSTRACT-SYNTAX.&Type.

CR-0545 directed a change to the Test Plan Test Directives and Testing Hints to reduce the number of instances of the test that must be run.

Proposed Changes:

BTL Checklist Changes

None

BTL Test Plan Changes

4.7.26 Can Write Constructed Property Values

The IUT is able to write constructed property values.

135.1-2019 - 8.23.1 - Writing a Single Property of a Single Object, 135.1-2019 - 8.23.2 - Writing Multiple Properties of a Single Object, 135.1-2019 - 8.23.3 - Writing Multiple Objects, One Property Each, or 135.1-2019 - 8.23.4 - Writing Multiple Objects, Multiple Properties for Each		
	Test Conditionality	At least one of the tests (8.23.1..8.23.4) shall be executed against a property with a constructed value. ~~This test shall be repeated for each standard constructed value that the IUT is able to write.~~
	Test Directives	~~At least one of the properties written by the selected test shall contain a constructed value.~~ *This test shall be repeated for each standard constructed value that the IUT is able to write.*
	Testing Hints	Where a constructed value can take on different forms, such as a constructed value that contains optional elements, or is a CHOICE, the tester should test all *supported* forms of the datatype. *Where the constructed value contains an ANY type, the tester should limit testing to supported primitive datatypes and, if supported, BACnetDateTime.*

Test Changes

None