[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.

BTL-TP15.0a-1: New FAULT Algorithms (AF-007-09), pg 2. [wID0166]
BTL-TP15.0a-2: Add Program Object, pg 26. [wID0060]
BTL-TP15.0a-3: Add Pulse Converter Object, pg 28. [wID0269]
BTL-TP15.0a-4: Add Non-Pattern Tests, pg 36. [wID0440]
BTL-TP15.0a-5: Add Non-Pattern Tests to Date_List property pg 41. [wID0185]

In the following document, language to be added to existing clauses within the BTL Test Package 14.0 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 addition, changes to BTL Specified Tests might 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.

This addendum contains results of various clarification requests put forth to the BTL-WG that resulted in test package changes.

BTL 15.0a-1: New FAULT Algorithms

Overview:

Addendum 135-2010af-21 and af-32 at Protocol_Revision 13 added language and many new FAULT algorithms to all objects that provide fault reporting, and to the Event Enrollment object.

Changes:

[In BTL Specified Tests, add these new tests]

8.4.X1 CHANGE_OF_RELIABILITY Tests

8.4.X1.1 CHANGE_OF_RELIABILITY with the NONE fault Algorithm

Purpose: To verify the correct operation of the NONE fault algorithm.

Test Concept: Select an object O1 capable of generating fault using the NONE fault algorithm. Ensure that no other fault conditions exist for the object. Create a fault condition. Verify the transition to fault is generated with Reliability set to R1. Remove the fault condition and verify the object transitions out of fault.

Test Configuration: O1 is configured to detect and report faults using unconfirmed event notifications. O1 is configured to have no fault conditions present and the Event_State is NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(O1 enter a fault condition)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local datetime or time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(R1 any valid BACnetReliability, (?, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE(O1clear the fault condition)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local datetime or time or sequence number),

'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(?, F, ?, ?),
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)

9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY pCurrentState = NORMAL

Notes to Tester: The mechanism to enter the NONE fault algorithm is a local matter.

8.4.X1.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm

Purpose: To verify the correct operation of the FAULT_CHARACTERSTRING fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_CHARACTERSTRING algorithm, and no other fault conditions exist for the object. pMonitoredValue is changed to a fault string and back to a non-fault string. It is verified that O1 generates the correct transitions.

Test Configuration: O1 is configured to detect and report unconfirmed events and faults, to have no fault conditions present, and to be in the NORMAL state. FVSET is the set of character strings defined as fault values for O1. ONVSET is the set of character strings defined as offnormal values for O1. FV1 contain a substring that exists in FVSET. If the empty string is included in the FVSET, then FV1 should be the empty string. NFV1 is a string value that does not contain substrings from FVSET or ONVSET.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = FV1
ELSE
MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (MULTI_STATE_FAULT,
(T, T, ?, ?),

(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)

5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NFV1
   ELSE
   MAKE (pMonitoredValue = NFV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

Notes to Tester: Note that a string is considered a substring of itself. Values required and allowed for O1 are described in standard 135 as "Properties Reported in CHANGE_OF_RELIABILITY Notifications" (Table 13-5 in 135-2016) along with supporting paragraphs.

8.4.X1.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm

Purpose: To verify the correct operation of the FAULT_EXTENDED fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_EXTENDED algorithm, and either pMonitoredValue is configured. Ensure that no other fault conditions exist for the object. In object O1, a condition is created that is detected as a fault by the FAULT_EXTENDED algorithm configured. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Test Configuration: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present, and has an Event_State of NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (a fault condition exist)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),

'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid event state),
'To State' = FAULT,
'Event Values' = ((R1: any valid reliability value),
(T, T, ?, ?),
(a vendor specified set of values)
)

5. VERIFY pCurrentReliability = R1
6. MAKE (remove the fault condition)
7. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(?, F, ?, ?),
(a vendor specified set of values)
)
8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm

Purpose: To verify the correct operation of the FAULT_LIFE_SAFETY fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_LIFE_SAFETY algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_LIFE_SAFETY fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and has an Event_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = FV1
ELSE

MAKE (pMonitoredValue = FV1)

4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MULTI_STATE_FAULT,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm

Purpose: To verify the correct operation of the FAULT_STATE fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_STATE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_STATE fault condition. Verify the correct transition is generated. The fault condition is removed by setting

pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and an Event_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(MULTI_STATE_FAULT, (T, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(NO_FAULT_DETECTED, (F, F, ?, ?), (A list of valid values for properties required to be reported

for O1, and 0 or more other properties of O1)
)

8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm

Purpose: To verify the correct operation of the FAULT_STATUS_FLAGS fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_STATUS_FLAGS algorithm. Ensure that no other fault conditions exist for the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_STATUS_FLAGS fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MEMBER_FAULT,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = MEMBER_FAULT
6. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,

'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(NO_FAULT_DETECTED, (F, F, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

8. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.7 CHANGE_OF_RELIABILITY for Event Enrollment Fault Condition Precedence

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the Event Enrollment object, then faults in the monitored object, and finally faults detected by the configured Fault algorithm.

Test Concept: Select an Event Enrollment object EE1 which monitors an object O2 that can transition into FAULT. EE1 should be able to be put into a state where it has an internal fault (internal to the Event Enrollment object and unrelated to the Reliability of the monitored object). Starting with both objects in a NORMAL state, cause a condition which results in a fault in O2. Verify that EE1 reports the fault. Make a condition exist that results in EE1 entering an internal fault. Verify that EE1 reports the new fault condition. Verify that a fault detectable by the fault algorithm does not generate an event. Clear EE1's the internal fault condition and verify that EE1 reports O2's fault. Clear the condition causing O2's fault and verify that EE1 reports fault algorithm event. Clear the condition causing the fault algorithm and verify the return to NORMAL event occurs.

Test Configuration: EE1 is configured to detect faults and contains a fault algorithm and is able to report those using unconfirmed event notifications. EE1 and O2 are each initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY pCurrentReliability = NO_FAULT_DETECTED
3. MAKE(a condition exist which will cause O2 to detect a fault)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	EE1,
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	FALSE,

'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (MONITORED_OBJECT_FAULT,
(T, T, ?, ?),
O2,
(optional, the value of the monitored property),
(optional, Reliability of O2),
(optional, Status_Flags of O2),
(0 or more other properties of O2)
)

5. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault)
7. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
‘Message Text’ = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = FAULT,
'Event Values' = ((R1: any value other than
MONITORED_OBJECT_FAULT
and NO_FAULT_DETECTED),
(T, T, ?, ?),
O2,
(optional, the value of the monitored property),
(optional, Reliability of O2),
(optional, Status_Flags of O2),
(0 or more other properties of O2)
)
8. VERIFY pCurrentReliability = R1
9. MAKE(a condition that results in a fault detectable by the configured fault algorithm with a reliability value, R2, different from R1)
10. CHECK(that the IUT does not send any notifications)
11. VERIFY pCurrentReliability = R1
12. MAKE(clear the condition that caused EE1 to enter into an internal fault)
11. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
‘Message Text’ = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = FAULT,

'Event Values' = (MONITORED_OBJECT_FAULT,
(T, T, ?, ?),
O2,
(optional, the value of the monitored property),
(optional, Reliability of O2),
(optional, Status_Flags of O2)
(0 or more other properties of O2)
)

12. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
13. MAKE(clear the condition that caused O2 to enter into fault)
14. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
‘Message Text’ = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = FAULT,
'Event Values' = (R2,
(T, T, ?, ?),
O2,
(optional, the value of the monitored property),
NO_FAULT_DETECTED,
(optional, Status_Flags of O2),
(0 or more other properties of O2)
)
15. VERIFY pCurrentReliability = R2
16. MAKE(clear the condition for the fault algorithm)
17. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
‘Message Text’ = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
O2,
(the value of the monitored property),
NO_FAULT_DETECTED,
(optional, Status_Flags of O2),
(0 or more other properties of O2)
)

19. VERIFY pCurrentReliability = NO_FAULT_DETECTED
20. VERIFY pCurrentState = NORMAL

Notes to Tester: If O2 is located in the IUT, then the IUT shall know and report the property values of O2 in the CHANGE_OF_RELIABILITY notifications. If O2 is not located in the IUT, then more time must be allowed between making or clearing a fault condition in O2 and the IUT detecting the change in O2's Reliability (the Notification Fail Time allowance does not start until after the IUT has acquired the information from O2).

8.4.X1.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault

Purpose: To verify the proper operation of the Event Enrollment object's fault detection when the monitored object enters the fault state.

Test Concept: Select an Event Enrollment object EE1 that monitors an object M1 that can transition into FAULT. Starting with both objects in a NORMAL state, cause a condition which results in a fault in M1. Verify EE1 reports the fault. Clear the condition and verify EE1 reports the return to NORMAL.

Test Configuration: EE1 is configured to process faults in M1 and to report those using unconfirmed event notifications. EE1 and M1 are each initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (M1 enter any fault state)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	EE1,
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the notification class configured for EE1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(MONITORED_OBJECT_FAULT, (T, T, ?, ?), M1, (optional, property value of M1), (optional, M1 Status_Flags, (?, T, ?, ?)), (0 or more other properties of M1))

5. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
6. VERIFY pCurrentState = FAULT
7. MAKE (M1 clear fault state)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' =	(any valid process identifier),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	EE1,
'Time Stamp' =	(the current local time or sequence number),

'Notification Class' = (the notification class configured for EE1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
M1,
(optional, property value of M1),
(optional, M1 Status_Flags, (?, F, ?, ?)),
(0 or more other properties of M1)
)

9. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault

Purpose: To verify the Event Enrollment object generates a fault event when the object enters into fault due to an internal unreliable operation.

Test Concept: Select an Event Enrollment object EE1 that can be made to enter into fault due to an internal unreliable operation. Starting EE1 in a NORMAL state, cause a condition which results in a fault. Verify that EE1 reports the fault. Clear the condition and verify that EE1 reports the return to NORMAL.

Test Configuration: EE1 is configured to be able to enter a fault state and to report those using unconfirmed event notifications. EE1 is initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (EE1 enter any fault state)
4. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = EE1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for EE1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = ((R1: any value other than
MONITORED_OBJECT_FAULT
and NO_FAULT_DETECTED),
(T, T, ?, ?),
(M1, any valid monitored object),
(optional, property value of M1),
(optional, M1 Status_Flags, (?, F, ?, ?)),
(0 or more other properties of M1)

)

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (EE1 clear fault state)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   M1,
   (optional, property value of M1),
   (optional, M1 Status_Flags, (?, F, ?, ?)),
   (0 or more other properties of M1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED

8.4.X1.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL

Purpose: To verify that objects go to the NORMAL state after leaving the FAULT state, then transition to OFFNORMAL if the condition still exists.

Test Concept: Select a fault detecting object O1 which is able to detect OFFNORMAL conditions. Make O1 transition to an OFFNORMAL state and then transition to FAULT. Remove the condition causing the FAULT and verify O1 transitions from FAULT to NORMAL, then verify that the object transitions from NORMAL to the original OFFNORMAL state.

Test Configuration: O1 is configured to detect and report unconfirmed events and faults. O1 is configured to have no fault conditions present, and Event_State is OFFNORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(O1transition to an off normal state)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),

'Event Type' = (ET1, any valid off normal event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (property-values appropriate for O1)

5. VERIFY pCurrentState = OFFNORMAL
5. MAKE(O1 enter a fault state)
6. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = FAULT,
'Event Values' = ((R1 any valid BACnetReliability),
(?, T, ?, ?),
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
7. MAKE(O1 clear the fault condition)
8. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),

'Priority' =	(the value configured for the transition),
'Event Type' =	ET1,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	(property-values appropriate for O1)

8.4.X1.11 CHANGE_OF_RELIABILITY with Internal Object Fault
Purpose: To verify thatfault conditions, unrelated to fault algorithms, are detected and reported.

Test Concept: An object in the IUT, O1, which can detect at least one internal fault is selected. One of O1's detectable internal faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO_FAULT_DETECTED. The internal fault condition, R1, is made to exist and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO_FAULT_DETECTED and the appropriate event notification message is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local datetime or time or sequence number),
'Notification Class' =	(the notification class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	(R1, (?, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (pCurrentReliability = NO_FAULT_DETECTED)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	O1,
'Time Stamp' =	(the current local datetime or time or sequence number),
'Notification Class' =	(the notification class configured for O1),

'Priority' =	(the value configured for the transition),
'Event Type' =	CHANGE_OF_RELIABILITY,
'Message Text' =	(optional, any valid message text),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	(NO_FAULT_DETECTED, (?, F, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY pCurrentState = NORMAL

[In BTL Test Plan, add the new test to section “Supports Event Reporting”]

3.36.20 Supports Event Reporting

Verify Checklist		
	Test Method	Manual
	Configuration	
	Test Conditionality	Must be executed.
	Test Directives	Verify that the IUT claims support for AE-N-I-B or AE-N-E-B in the Checklist with option " Implements the CHANGE_OF_RELIABILITY – FAULT_STATUS_FLAGS Algorithm".
	Testing Hints	
	Notes & Results	

BTL - 8.4.X1.11 - CHANGE_OF_RELIABILITY with Internal Object fault		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	This test shall be executed if the object's Reliability property can be made to equal COMMUNICATION_FAILURE otherwise this test shall be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, Append section 5.2.1 Base Requirements]

BTL - 8.4.X1.10 - After FAULT-to-NORMAL, Re-Notification of OFFNORMAL		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY is implemented in an object that can be configured into an offnormal state, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, add tests to Alarm and Event - Notification - Internal - B Base Requirements, with Test Directives to indicate selecting objects to which to apply the tests]

5.2.30 Implements the CHANGE_OF_RELIABILITY – NONE Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.1 - CHANGE_OF_RELIABILITY with the NONE Fault Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the NONE fault Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester shall include all object types that support this algorithm.
	Testing Hints	
	Notes & Results	

5.2.31 Implements the CHANGE_OF_RELIABILITY – FAULT_CHARACTERSTRING Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.2 - CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm is implemented, this

		test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.2.32 Implements the CHANGE_OF_RELIABILITY – FAULT_EXTENDED Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.3 - CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.2.33 Implements the CHANGE_OF_RELIABILITY – FAULT_LIFE_SAFETY Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.4 - CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.2.34 Implements the CHANGE_OF_RELIABILITY – FAULT_STATE Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.5 - CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.2.35 Implements the CHANGE_OF_RELIABILITY – FAULT_STATUS_FLAGS Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.6 - CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.2.36 Supports CHANGE_OF_RELIABILITY in the Event Enrollment Object

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

BTL - 8.4.X1.7 - CHANGE_OF_RELIABILITY for Event Enrollment Fault Condition Precedence		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY, can be made to transition to fault and supports a fault algorithm and the Monitored_Object can transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.4.X1.8 - CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault		

	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed if and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY and the Monitored_Object that can transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.4.X1.9 - CHANGE_OF_RELIABILITY of Event Enrollment Object Fault		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed if and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY and can be made to transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, Append section 5.3.1 Base Requirements]

BTL - 8.4.X1.10 - After FAULT-to-NORMAL, Re-Notification of OFFNORMAL		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY is implemented in an object that can be configured into an offnormal state, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, add tests to Alarm and Event - Notification - External - B Base Requirements, with Test Directives to indicate selecting objects to which to apply the tests]

5.3.22 Implements the CHANGE_OF_RELIABILITY – NONE Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.1 - CHANGE_OF_RELIABILITY with the NONE Fault Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with

		the NONE fault Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester shall include all object types that support this algorithm.
	Testing Hints	
	Notes & Results	

5.3.23 Implements the CHANGE_OF_RELIABILITY – FAULT_CHARACTERSTRING Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.2 - CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.3.24 Implements the CHANGE_OF_RELIABILITY – FAULT_EXTENDED Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.3 - CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.3.25 Implements the CHANGE_OF_RELIABILITY – FAULT_LIFE_SAFETY Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.4 - CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.3.26 Implements the CHANGE_OF_RELIABILITY – FAULT_STATE Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.5 - CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.3.27 Implements the CHANGE_OF_RELIABILITY – FAULT_STATUS_FLAGS Algorithm

The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_RELIABILITY and supports the specified algorithm.

BTL - 8.4.X1.6 - CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	If the IUT has no object in which CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm is implemented, this test shall be skipped.
	Test Directives	The objects selected by the tester should include all variants that differ in the set of supported alarming properties, or the writability of any of those properties. At least one instance of each variant shall be selected.
	Testing Hints	
	Notes & Results	

5.3.28 Supports CHANGE_OF_RELIABILITY in the Event Enrollment Object

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

BTL - 8.4.X1.7 - CHANGE_OF_RELIABILITY for Event Enrollment Objects Precedence		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY, can be made to transition to fault and supports a fault algorithm and the Monitored_Object can transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.4.X1.8 - CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed if and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY and the Monitored_Object that can transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.4.X1.9 - CHANGE_OF_RELIABILITY of Event Enrollment Object Fault		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	This test shall be executed if and only if the IUT contains an Event Enrollment object that supports CHANGE_OF_RELIABILITY and can be made to transition to fault.
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Checklist, add new sections as shown here]

Alarm and Event - Notification - Internal - B

Support	**Listing**	**Option**
. . .		
	O	Implements intrinsic alarming in an Integer object
	C[3]	*Implements the CHANGE_OF_RELIABILITY – NONE Algorithm*

Support	Listing	Option
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_CHARACTERSTRING Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_EXTENDED Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_LIFE_SAFETY Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_STATE Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_STATUS_FLAGS Algorithm*
	C[3]	*Supports CHANGE_OF_RELIABILITY in the Event Enrollment Object*
. . .		

Alarm and Event - Notification - External - B

Support	Listing	Option
. . .		
	C[1]	Implements the UNSIGNED_RANGE algorithm
	C[3]	*Implements the CHANGE_OF_RELIABILITY – NONE Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_CHARACTERSTRING Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_EXTENDED Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_LIFE_SAFETY Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_STATE Algorithm*
	C[3]	*Implements the CHANGE_OF_RELIABILITY – FAULT_STATUS_FLAGS Algorithm*
	C[3]	*Supports CHANGE_OF_RELIABILITY in the Event Enrollment Object*
. . .		

Global Group Object		
	…	
	O	Supports Event Reporting

BTL 15.0a-2: Add Program Object

Overview:

This document shows added testing for Program objects:

- Verify writability test of Program Change.

[In BTL Specified Tests, add new sections and tests as shown here]

7.3.2.22 Program Object Tests

~~The Program object makes parameters of a custom program network visible. Since BACnet does not define the functionality of the program there are no standard tests to verify this functionality.~~ *The Program object utilizes parameter control through its writable Program_Change property.*

7.3.2.22.1 Program_Change property test

Reason for Change: This test is not specified in any SSPC proposal.

Purpose: To verify writability of Program_Change property.

Test Concept: The Program_Change property is set to a value other than READY and then it and the Program_State property are verified to update correctly.

Configuration Requirements: The Program_Change property of the program object being tested shows a value of READY.

Test Steps:

1. VERIFY Program_Change = READY
2. WRITE Program_Change = (a value other than READY)
3. WAIT (for the processing to consume that value written to Program_Change)
4. VERIFY Program_Change = READY
5. VERIFY Program_State = the new state reflected, based upon value written to Program_Change in step 2.

Notes to Tester: In step 2, depending on the current Program_State, and the implementation, certain requested values for Program_Change may be invalid and would return a Result(-) if an attempt were made to write them.

[In BTL Test Plan, add new sections and test references, as shown here]

3.39 Program Object

3.39.1 Base Requirements

All BACnet devices must meet these base requirements.

BTL - 7.3.2.22.1 - Program_Change property test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests.***
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	Test only Program_Change values RUN and HALT.
	Notes & Results	

[In BTL Checklist, add new sections as shown here]

Support	Listing	Option
. . .		
Program Object		
	R	Base Requirements

BTL 15.0a-3: Add Pulse Converter Object

Overview:

Pulse Converter object type, specified in the standard in Protocol_Revision 4, deserves testing coverage in our Test Plan.

Changes:

[In BTL Checklist, add new Pulse Converter section in existing 3. Object testing]

Pulse Converter		
	R	Base Requirements
	O	Supports writable Out_Of_Service properties

[In BTL Checklist, add new object type section in existing 4.10 DS-COV-B testing]

Data Sharing - COV - B		
	R	. . .
	C[1]	Supports COV for OctetString Value objects
	C[1]	Supports COV for Pulse Converter objects
	O	Supports COV for proprietary objects
	S	Will accept infinite COV subscriptions
[1] At least one of these options must be supported to claim support for this BIBB.		

[In BTL Test Plan, add new Pulse Converter section at end of existing 3. Object testing]

. . .

3.38 Pulse Converter Object

3.38.1 Base Requirements

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

BTL - 7.3.2.X38.1.1 - Adjust_Value Write Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.2.X38.1.2 - Scale_Factor Pulse Converter Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed .
	Test Directives	
	Testing Hints	
	Notes & Results	

BTL – 7.3.2.X38.1.5 - Update_Time Reflects Change to the Count and is Updated Atomically Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.2.X38.2.1 - Adjust_Value Out-of-Range WriteProperty Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	Verify in the EPICS that Value_Before_Change in the object is read-only.
	Testing Hints	
	Notes & Results	
Verify EPICS		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	Verify in the EPICS that Update_Time and Count_Change_Time in the object are read-only.
	Testing Hints	
	Notes & Results	

3.38.2 Supports Writable Out_Of_Service Properties

The Out_Of_Service property in Accumulator objects is writable.

135.1-2013 - 7.3.2.X38.1.3 - Out_Of_Service in Pulse Converter Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, add new COV for Pulse Converter objects section near end of existing COV - B objects in section 4.10 before proprietary objects, and renumber subsequent sections as indicated.]

4.10.27 Supports COV for Pulse Converter objects

The IUT supports change of value notifications for at least one object of type Pulse Converter.

BTL - 8.2.X9 - ConfirmedCOVNotification Pulse Converter changing Present_Value		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.

	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.2.X10 - ConfirmedCOVNotification Pulse Converter changing Status_Flags		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.3.X12 - UnconfirmedCOVNotification Pulse Converter changing Present_Value		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 8.3.X13 - UnconfirmedCOVNotification Pulse Converter changing Status_Flags		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	

4.10.287 Supports COV for Proprietary Objects

The IUT supports change of value notifications for at least one proprietary object.

There is no test defined for this functionality at this time.

4.10.298 Will accept infinite COV subscriptions

The IUT will accept subscriptions that do not include a lifetime parameter.

The BTL does not consider the use of infinite subscriptions a wise implementation choice due to the inability of the server to timeout the subscription if the client disappears. For the sake of interoperability, the BTL considers it wise for all COV server implementations to accept infinite subscriptions.

. . .

[In BTL Specified Tests, add new tests 7.3.2.X38.1.1 through 7.3.2.X38.1.5]

7.3.2.X38.1.1 Adjust_Value Write Test

Purpose: To verify the correct write operation of a Pulse Converter's several properties, when writing the Adjust_Value. Count_Before_Change reflects the prior Count before a write to the Adjust_Value property.

Test Steps:

1. READ OldV = Present_Value
2. READ OldC = Count
3. READ OldU = Update_Time
4. READ OldT = Count_Change_Time
5. READ OldA = Adjust_Value
6. READ OldS = Scale_Factor
7. READ OldB = Count_Before_Change
8. WRITE Adjust_Value = (NewA, any valid value, different from OldA so that it can be distinguished)
9. CHECK (Count is decremented by the value calculated by performing the integer division (NewA/OldS) and discarding the remainder)
10. VERIFY Present_Value is decremented by the value NewA
11. VERIFY Count_Change_Time = (approximately the current local time, and different from OldT)
12. VERIFY Count_Before_Change = OldC and != OldB

7.3.2.X38.1.2 Scale_Factor Test

Purpose: To verify the correct effect of Scale_Factor on the Present_Value in Pulse Converter.

Test Concept: The IUT shall be configured with a Scale_Factor whose influence on the behavior of Present_Value is observable. After Present_Value is read, then the value derived from Count and Scale_Factor is compared to the expected Present_Value.

Test Steps:

1. IF (Scale_Factor is writable) THEN
   WRITE Scale_Factor = (any valid value V_1)
   ELSE
   MAKE (Scale_Factor equal any valid value V_1)
2. VERIFY (Present_Value = conversion specified by Scale_Factor V_1 coefficient times the Count property)

7.3.2.X38.1.3 Out_Of_Service Pulse Converter Test

Purpose: This test case verifies that Present_Value the Pulse_Rate, and the Reliability property are writable when Out_Of_Service is TRUE. It also verifies the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties. If the PICS indicates that the Out_Of_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.

Test Concept: The IUT will select one instance of each appropriate object type and test it as described. If the Reliability property is not supported then step 5 shall be omitted.

Test Steps:

1. IF (Out_Of_Service is writable) THEN
   WRITE Out_Of_Service = TRUE
   ELSE
   MAKE (Out_Of_Service TRUE)
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4. REPEAT X = (any values meeting the functional range requirements of 7.2.1) DO {
   WRITE Present_Value = X
   VERIFY Present_Value = X
   }
5. IF (Reliability is present and writable) THEN
   REPEAT X = (any values of the Reliability enumeration appropriate to the object type except
   NO_FAULT_DETECTED) DO {
   WRITE Reliability = X

VERIFY Reliability = X
VERIFY Status_Flags = (?, TRUE, ?, TRUE)
WRITE Reliability = NO_FAULT_DETECTED
VERIFY Reliability = NO_FAULT_DETECTED
VERIFY Status_Flags = (?, FALSE, ?, TRUE)
}

6. REPEAT X = (any values meeting the functional range requirements of 7.2.1) DO {
   WRITE Pulse_Rate = X
   VERIFY Pulse_Rate = X
   }
7. IF (Out_Of_Service is writable) THEN
   WRITE Out_Of_Service = FALSE
   ELSE
   MAKE (Out_Of_Service FALSE)
8. VERIFY Out_Of_Service = FALSE
9. VERIFY Status_Flags = (?, ?, ?, FALSE)

7.3.2.X38.1.5 Update_Time Reflects Change to the Count and is Updated Atomically Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter's several properties, for an inherent change in Count.

Test Steps:

1. READ OldV = Present_Value
2. READ OldC = Count
3. READ OldU = Update_Time
4. READ OldT = Count_Change_Time
5. READ OldA = Adjust_Value
6. READ OldS = Scale_Factor
7. READ OldB = Count_Before_Change
8. WAIT (for a change in Count to any valid value, different from OldC so that it can be distinguished)
9. CHECK Present_Value is recalculated, increasing in proportion to the change in Count multiplied by OldS (or such that Present_Value minus OldA is still the same fixed difference)
10. VERIFY Update_Time = (approximately the current local time, and different from OldU)
11. VERIFY Count_Change_Time = OldT

7.3.2.X38.2.1 Adjust_Value Out-of-Range WriteProperty Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter Count property, when an attempt is made to write Adjust_Value with a value that would cause an overflow or underflow condition in Count. The test is performed once using WriteProperty and once using WritePropertyMultiple, if IUT supports both services.

Test Steps:

1. READ OldV = Present_Value
2. READ OldC = Count
3. READ OldU = Update_Time
4. READ OldT = Count_Change_Time
5. READ OldA = Adjust_Value
6. READ OldS = Scale_Factor
7. READ OldB = Count_Before_Change
8. TRANSMIT WriteProperty-Request
   'Property Identifier' = Adjust_Value
   'Property Value' = (NewA, a valid value that would cause an overflow or underflow condition in Count)
9. RECEIVE BACnet-Error-PDU

'Error Class' = PROPERTY
'Error Code' = VALUE_OUT_OF_RANGE
10. VERIFY Update_Time = OldU
11. VERIFY Adjust_Value = OldA
12. VERIFY Count_Before_Change = OldB

[In BTL Specified Tests, add new tests 8.2.X9, 8.2.X10, 8.3.X12, and 8.3.X13]

8.2.X9 ConfirmedCOVNotification Pulse Converter changing Present_Value

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present_Value property.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present_Value property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will enable the Present_Value property to be changed by the WriteProperty service. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control by more than COV_Increment or which has a writable Out_Of_Service.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = O1,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value, initial Status_Flags, and Update_Time)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment
6. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment,
   'Property Value' = (a value "increment" that will be used below)
7. IF (Out_Of_Service is writable) THEN
   WRITE O1, Out_Of_Service = TRUE
   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,

'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (ReportedPV = the current Present_Value, new Status_Flags, and current Update_Time)
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Present_Value is now writable) THEN
WRITE O1, Present_Value = (any value that differs from ReportedPV by less than "increment")
ELSE
MAKE (Present_Value = any value that differs from ReportedPV by less than "increment")
10. WAIT **Notification Fail Time**
11. CHECK (verify that no COV notification was transmitted)
12. IF (Present_Value is now writable) THEN
WRITE O1, Present_Value = (any value that differs from ReportedPV by an amount greater than "increment")
ELSE
MAKE (Present_Value = any value that differs from ReportedPV by an amount greater than "increment")

13. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the new Present_Value, new Status_Flags, and current Update_Time)
14. TRANSMIT BACnet-SimpleACK-PDU
15. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = O1
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Out_Of_Service was changed in step 7) THEN
WRITE O1, Out_Of_Service = FALSE

8.2.X10 ConfirmedCOVNotification Pulse Converter changing Status_Flags

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status_Flags property.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status_-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control by more than COV_Increment or which has a writable Out_Of_Service.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
'Monitored Object Identifier' = O1,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L

2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value, initial Status_Flags, and Update_Time)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE O1, Out_Of_Service = TRUE | WRITE O1, Status_Flags = (a value that differs from initial Status_Flags) |
   MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the current Present_Value, new Status_Flags, and Update_Time)
7. TRANSMIT BACnet-SimpleACK-PDU
8. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = O1
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Out_Of_Service was changed in step 5) THEN
    WRITE O1, Out_Of_Service = FALSE

8.3.X12 UnconfirmedCOVNotification Pulse Converter changing Present_Value

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present_Value property.

Test Concept: This test is the same as 8.2.X9 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

8.3.X13 UnconfirmedCOVNotification Pulse Converter changing Status_Flags

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status_Flags property.

Test Concept: This test is the same as 8.2.X10 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

BTL 15.0a-4: Add Non-Pattern Tests

Overview:

Tests for the Time in time-value pairs in both Exception_Schedule and Weekly_Schedule properties are to be applied to devices claiming protocol revision 11 or higher. Also adds testing for Effective_Period, and for the BACnetCalendarEntry in Exception_Schedule property which are BACnetDateRange, to the restrictions of BACnetDateRange which were added in Addendum 135-2008*ac*-1.

Changes:

[In BTL Specified Test, add two new tests]

7.2.X7 BACnetDateRange Non-Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

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

Test Concept: A BACnetDateRange property P_1 is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange DR_1 is selected so it is within the range that the IUT will accept for the property. The value, V_1, written to the property is the daterange DR_1 with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalenderEntry, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified,
   odd months, even months, last day of month, even days, odd days) DO {
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O_1,
   'Property Identifier' = P_1,
   'Property Value' = (DR_1 with one-half updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P_1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

Purpose: To verify that the property being tested accepts a fully unspecified date in either or both halves of the value.

Test Concept: A BACnetDateRange property P_1 is written with a fully unspecified date in either or both halves to ensure that the property accepts them. A date DR_1 is selected which is within the date range that the IUT will accept for the property. The value, written to the property is the date DR_1 with one of its fields replaced with a fully unspecified date in either or both startDate and endDate. If the property is a complex datatype the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
Test Steps:

1. WRITE P_1 = (DR_1 updated with a fully unspecified date in startDate)
2. VERIFY P_1 = (the value written)
3. WRITE P_1 = (DR_1 updated with a fully unspecified date in endDate)
4. VERIFY P_1 = (the value written)
5. WRITE P_1 = (DR_1 updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P_1 = (the value written)

Notes to Tester: if P_1 is an array, then an array index shall be provided in the WRITEs and VERIFYs.

[In BTL Test Plan in the Schedule object Base Requirements section, relocate test 7.2.X1, and revise the preamble.]

3.19 Schedule

3.19.1 Base Requirements

Base requirements must be met by any IUT that can contain Schedule objects. There are no base requirements tests for this section.

~~**BTL - 7.2.X1 - Date Pattern Properties Test**~~		
	Test Method	
	Configuration	~~As per ***BTL Specified Tests.***.~~
	Test Conditionality	~~Must be executed.~~
	Test Directives	~~Apply to the Exception_Schedule property.~~
	Testing Hints	
	Notes & Results	

[In BTL Test Plan in the Schedule - Internal - B, Base Requirements section, move the test 7.2.X1, and add three tests]

6.4 Scheduling - Internal - B

6.4.1 Base Requirements

Base requirements must be met by any IUT claiming conformance to this BIBB. (The BIBB requires, among other things, support for either TimeSynchronization-Request or UTCTimeSynchronization-Request execution; these are tested by the Device Management tests.)

. . .		
BTL - 7.2.X1 - Date Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to BACnetCalendarEntry in the Exception_Schedule property in the BACnet Date form.
	Testing Hints	
	Notes & Results	

BTL - 7.2.X5 - Time Non-Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to the time portion of BACnetTimeValues in the Exception_Schedule property, then apply to the time portion of BACnetTimeValues in the Weekly_Schedule property.
	Testing Hints	
	Notes & Results	
BTL - 7.2.X7 - BACnetDateRange Non-Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to BACnetCalendarEntry in the Exception_Schedule property in the BACnetDateRange form.
	Testing Hints	
	Notes & Results	
BTL - 7.2.X8 - BACnetDateRange Open-Ended Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to BACnetCalendarEntry in the Exception_Schedule property in the BACnetDateRange form.
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, within Scheduling - Internal - B existing section named: Supports Configurable Effective_Period, append two additional test references.]

6.4 Scheduling - Internal - B

6.4.4 Supports Configurable Effective_Period

The IUT supports the Effective_Period property and it is configurable.

BTL - 7.3.2.23.1 - Effective_Period Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	This test shall be executed if and only if the IUT is prior to protocol revision 4. If the IUT is of the correct Protocol_Revision, the IUT is required to be configurable such that this test can be run. This test may not be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	Old Reference: 135.1-2003 - 7.3.2.22.1

BTL - 7.3.2.23.X.1 - Revision 4 Effective_Period Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	This test shall be executed if and only if the IUT is protocol revision 4 or higher. If the IUT is of the correct Protocol_Revision, the IUT is required to be configurable such that this test can be run. This test may not be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	

BTL - 7.2.X7 - BACnetDateRange Non-Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to the Effective_Period property.
	Testing Hints	
	Notes & Results	

BTL - 7.2.X8 - BACnetDateRange Open-Ended Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to the Effective_Period property.
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, within Scheduling - Weekly Schedule - Internal - B existing section named: Supports Configurable Effective_Period, append two additional test references.]

6.6 Scheduling - Weekly Schedule - Internal - B

6.6.7 Supports Configurable Effective_Period

The IUT supports the Effective_Period property and it is configurable.

BTL - 7.3.2.23.1 - Effective_Period Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	This test shall be executed if and only if the IUT is prior to protocol revision 4. If the IUT is of the correct Protocol_Revision, the IUT is required to be configurable such that this test can be run. This test may not be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	Old Reference: 135.1-2003 - 7.3.2.22.1

BTL - 7.3.2.23.X.1 - Revision 4 Effective_Period Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	This test shall be executed if and only if the IUT is protocol revision 4 or higher. If the IUT is of the correct Protocol_Revision, the IUT is required to be configurable such that this test can be run. This test may not be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.2.X7 - BACnetDateRange Non-Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to the Effective_Period property.
	Testing Hints	
	Notes & Results	
BTL - 7.2.X8 - BACnetDateRange Open-Ended Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to the Effective_Period property.
	Testing Hints	
	Notes & Results	

BTL 15.0a-5: Add Non-Pattern Tests to Date_List property

Overview

Tests for the restrictions of BACnetDateRange, which were added in Addendum 135-2008*ac*-1, should be applied to the Date_List property.

Changes

[In BTL Specified Test, add these two tests, with modifications, as shown relative to the versions in wID0440]

7.2.X7 BACnetDateRange Non-Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

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

Test Concept: A BACnetDateRange property, *or property that is a complex datatype containing a BACnetDateRange* P_1 is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange DR_1 is selected so it is within the range that the IUT will accept for the property. The value, V_1, written to the property is the date*R*~~r~~ange DR_1 with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalenderEntry, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified,
   odd months, even months, last day of month, even days, odd days) DO {
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Value' = (DR1 with *startDate*~~one half~~ updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O_1,
   'Property Identifier' = P_1,
   'Property Value' = (DR_1 with endDate updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P_1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

Purpose: To verify that the property being tested accepts a fully unspecified date in either or both halves of the value.

Test Concept: A BACnetDateRange property, *or property that is a complex datatype containing a BACnetDateRange* P_1 is written with a fully unspecified date in either or both halves to ensure that the property accepts them. ~~A date~~ DR_1 is selected which is within the date range that the IUT will accept for the property. The value, written to the property is the date*Range* DR_1 ~~with one of its fields~~ replaced with a fully unspecified date in either or both *startDate and endDate*~~halves~~. If the property is a complex datatype the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
Test Steps:

1. WRITE P_1 = (DR_1 updated with a fully unspecified date in startDate)
2. VERIFY P_1 = (the value written)
3. WRITE P_1 = (DR_1 updated with a fully unspecified date in endDate)
4. VERIFY P_1 = (the value written)
5. WRITE P_1 = (DR_1 updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P_1 = (the value written)

Notes to Tester: if P_1 is an array, then an array index shall be provided in the WRITEs and VERIFYs.

[In BTL Test Plan in the Calendar object Base Requirements section, reference tests 7.2.X7 and 7.2.X8.]

3.8 Calendar

3.8.1 Base Requirements

Base requirements must be met by any IUT that can contain Calendar Objects.

BTL - 7.3.2.8.1 - Single Date Rollover Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.2.8.2 - Date Range Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.2.8.3 - WeekNDay Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

	Notes & Results	
BTL - 7.2.X1 - Date Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	Must be executed.
	Test Directives	Apply to Date_List property.
	Testing Hints	
	Notes & Results	
BTL - 7.2.X7 - BACnetDateRange Non-Pattern Properties Test		
	Test Method	
	Configuration	As per ***BTL Specified Tests.***.
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to Date_List property.
	Testing Hints	
	Notes & Results	
BTL - 7.2.X8 - BACnetDateRange Open-Ended Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	This test shall only be applied to devices claiming Protocol_Revision 11 or higher.
	Test Directives	Apply to Date_List property.
	Testing Hints	
	Notes & Results	