

Clarification Request

References: ASHRAE 135.1-2025, ASHRAE 135-2024, Test Package 26.1.

Date of BTL-WG Response: June 25, 2026

Background:

135-2024 Table 12-4 Properties of the Analog Value Object Type

LOW_LIMIT	Real	3,6
Deadband	Real	3,6
Present_Value property.		
³ These properties are required if the object supports intrinsic reporting.		
⁴ If Present_Value is commandable, then it is required to be writable. This property is required to be writable when Out_Of_Service is TRUE.		
⁵ Footnote removed.		
⁶ These properties shall be present only if the object supports intrinsic reporting.		
⁷ ...		

135-2024 Clause 12.4.18, Deadband property definition

This property is the pDeadband parameter for the object's event algorithm. See Clause 13.3 for event algorithm parameter descriptions.

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 Status_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY pCurrentState = HIGH_LIMIT
12. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
    VERIFY Event_Time_Stamp = (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 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 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 $\geq$ 13) THEN

VERIFY Status_Flags = (FALSE, FALSE, ?, ?)

44. VERIFY pCurrentState = NORMAL

45. IF (Protocol_Revision is present AND Protocol_Revision $\geq$ 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.

Problem:

The Deadband property is not required to be writable or configurable. Steps 2 and 13 require the tester to select a value x such that $(pHighLimit - pDeadband) < x < pHighLimit$. Steps 24 and 35 require the tester to select a value x such that $pLowDiffLimit < x < (pLowDiffLimit + pDeadband)$.

If the Deadband property is hardcoded to '0', it is not possible to select a value 'x' that satisfies these conditions.

Question:

Should Steps 2, 13, 24, and 35 be skipped when the Deadband property is hardcoded to '0'?

Response:

Yes.