BACnet® TESTING LABORATORIES ADDENDA

Addendum i to BTL Test Package 15.2

Revision 9
Revised 6/28/2019

Approved by the BTL Working Group on April 25, 2019.
Approved by the BTL Working Group Voting Members on June 28, 2019.
Published on July 1, 2019.

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

FOREWORD

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

In the following document, language to be added to existing clauses within the BTL Test Package 15.2 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout

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

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

BTL-15.2i-1: Lighting Output Object Tests [BTLWG-31]

Overview:
Add Lighting output object tests to the test package.
This addenda moves and updates sections BTL-TP15.0-5.1.0 and BTL-TP15.1-5.2.0 from the Interim Tests document into the Test Plan, Checklist and Specified Tests documents.

Changes:

[In Interim Tests, remove sections BTL-TP15.0-5.1.0 and BTL-TP15.1-5.2.0]

[In BTL Checklist, modify Lighting Output object type to Section 3, Objects]

Support	Listing	Option
Lighting Output Object		
	R[+]	Base Requirements
	R	*Supports command prioritization*
	R	*Supports all BACnetLightingOperations*
	S	*Supports writable Out_Of_Service properties*
	O	*Supports blink-warn*
	O	*Supports Transition property*
	O	*Supports Feedback_Value property*
	O	*Supports Min_Actual_Value and Max_Actual_Value properties*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
[+]~~Contact BTL for interim tests for this object.~~		

[In BTL Test Plan, modify the Base Requirements for 3.54 Lighting Output]

3.54 Lighting Output Object

3.54.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Lighting Output objects.*

[In BTL Test Plan, add the following tests, to 3.54.1 Base Requirements]

BTL - 7.3.2.X54.21 - Lighting Output Tracking Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X54.22 - Lighting Output Present Value between 0.0 and 1.0 Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

[In BTL Test Plan, Add the following new sections to 3.54 Lighting Output Object]

3.54.2 Supports Command Prioritization

135.1-2013 - 7.3.1.2 - Relinquish Default Test		
	Test Conditionality	If no object can be made to meet the configuration requirements, this test shall be skipped.

	Test Directives	
	Testing Hints	
135.1-2013 - 7.3.1.3 - Command Prioritization Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.54.3 Supports all BACnetLightingOperations

BTL -7.3.2.X54.31 - Lighting Command Operation NONE Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X54.32 - Lighting Command Operation FADE_TO Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and fade-time) and check that PTY1= Lighting_Command_Default_Priority and fade-time = Default_Fade_Time
	Testing Hints	
BTL - 7.3.2.X54.33 - Lighting Command Operation RAMP_TO Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and ramp-rate) and check that PTY1= Lighting_Command_Default_Priority and ramp-rate = Default_ Ramp_Rate
	Testing Hints	
BTL - 7.3.2.X54.34 - Lighting Command Operation STEP_UP Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and ramp-rate) and check that PTY1= Lighting_Command_Default_Priority and step-increment = Default_ Step_Increment
	Testing Hints	
BTL - 7.3.2.X54.35 - Lighting Command Operation STEP_DOWN Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and ramp-rate) and check that PTY1= Lighting_Command_Default_Priority and step-increment = Default_ Step_increment
	Testing Hints	
BTL - 7.3.2.X54.36 - Lighting Command Operation STEP_ON Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and ramp-rate) and check that PTY1= Lighting_Command_Default_Priority and step-increment = Default_ step-increment
	Testing Hints	
BTL - 7.3.2.X54.37 - Lighting Command Operation STEP_OFF Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test by using the BACnetLightingCommand without the optional fields (priority and ramp-rate) and check that PTY1= Lighting_Command_Default_Priority and step-increment = Default_ step-increment
	Testing Hints	

3.54.4 Supports Writable Out_Of_Service Property

The Out_Of_Service property in Lighting Output objects contained in the IUT are writable.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	Must be executed.
	Test Directives	The test shall be executed using an Lighting Output object
	Testing Hints	

3.54.5Supports Blink-Warn

The Blink_Warn_Enable property in Lighting Output is writable or can be changed to TRUE by other means.

BTL - 7.3.1.X41.Y1 - Blink-Warn WARN Command Test		
	Test Conditionality	Must be executed.
	Test Directives	Must be executed using both the Present_Value and Lighting_Command properties.
	Testing Hints	
BTL - 7.3.1.X41.Y2 - Blink-Warn WARN_OFF Command Test		
	Test Conditionality	Must be executed.
	Test Directives	Must be executed using both the Present_Value and Lighting_Command properties.
	Testing Hints	
BTL - 7.3.1.X41.Y3 - Blink-Warn WARN_RELINQUISH Command Test		
	Test Conditionality	Must be executed.
	Test Directives	Must be executed using both the Present_Value and Lighting_Command properties.
	Testing Hints	
BTL - 7.3.1.X41.Y4 - Blink-Warn STOP Command Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test with WARN_OFF and WARN_RELINQUISH commands
	Testing Hints	
BTL - 7.3.1.X41.Y5 - Blink-Warn WARN Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y6 - Blink-Warn WARN_OFF Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y7 - Blink-Warn WARN_RELINQUISH Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y8 - Blink-Warn WARN_OFF Command Halted Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y9 - Blink Warn WARN_RELINQUISH Command Halted Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.54.6Supports Transition Property

The IUT contains Lighting Output Objects in which the Transition property is supported.

BTL - 7.3.2.X54.41 - Transition None Test		
	Test Conditionality	Must be executed.
	Test Directives	

	Testing Hints	
BTL - 7.3.2.X54.42 - Transition Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.54.7 Supports Feedback_Value Property

The IUT contains Lighting Output Objects in which the the Feedback_Value property is supported.

BTL - 7.3.2.X54.51 - Feedback_Value Clamping Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.54.8 Supports Min_Actual_Value and Max_Actual_Value Properties

The IUT contains Lighting Output Objects in which the the Min_Actual_Value and Max_Actual_Value properties are supported.

BTL - 7.3.2.X54.61 - Min_Actual_Value and Max_Actual_Value Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X54.62 - Min_Actual_Value and Max_Actual_Value ScalingTest		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.54.9 Contains an Object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.
	Test Directives	
	Testing Hints	

[In BTL Test Plan, modify existing 'Supports COV for Lighting Output Objects' in 'Change Of Value - B' section 4.10.32]

4.10.32 Supports COV for Lighting Output Objects

The IUT supports change of value notifications for at least one object of type Lighting Output.

~~Contact BTL for interim tests for this object.~~

BTL - 8.2.1 - Change of Value Notification from an Analog Input, Analog Output, Lighting Output, Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object's Present_Value Property		
	Test Conditionality	*Must be executed.*
	Test Directives	

	Testing Hints	*This may be skipped if 8.3.1 is executed against a Lighting Output object.*
BTL - 8.2.2 - Change of Value Notification from an Analog Input, Analog Output, Lighting Output, Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object's Status_Flags Property		
	Test Conditionality	*Must be executed.*
	Test Directives	*The selected object must be a Lighting Output.*
	Testing Hints	*This may be skipped if 8.3.2 is executed against a Lighting Output object.*
BTL - 8.3.1 - Change of Value Notification from a Numeric Object's Present_Value Property		
	Test Conditionality	*Must be executed.*
	Test Directives	*The selected object must be a Lighting Output.*
	Testing Hints	*This may be skipped if 8.2.1 is executed against a Lighting Output object.*
BTL - 8.3.2 - Change of Value Notification from a Numeric Object's Status_Flags Property		
	Test Conditionality	*Must be executed.*
	Test Directives	*The selected object must be a Lighting Output.*
	Testing Hints	*This may be skipped if 8.2.2 is executed against a Lighting Output object.*

[In BTL Specified Tests, add Lighting Output object specific tests in section 7.3.x]

7.3.2.X54.21 - Lighting Output Tracking Test

Purpose: To verify that the Tracking_Value property follows the Present_Value property.

Test Concept: Write to the Present_Value of a Lighting Output object, O1, and verify that the Tracking_Value property follows Present_Value once In-Progress returns to IDLE.

Configuration Requirements: The IUT shall be configured with a lighting output, O1, that can be observed during the test. O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1 and Out_Of_Service = FALSE.

Test Steps:
1. WRITE Present_Value = 100, PRIORITY = PTY1
2. VERIFY Present_Value = 100
3. WHILE (In_Progress <> IDLE) DO {
   }
4. VERIFY Tracking_Value = 100
5. WRITE Present_Value = 1, PRIORITY = PTY1
6. VERIFY Present_Value = 1
7. WHILE (In_Progress <> IDLE) DO {
   }
8. VERIFY Tracking_Value = 1
9. WRITE Present_Value = 0, PRIORITY = PTY1
10. VERIFY Present_Value = 0
11. WHILE (In_Progress <> IDLE) DO {
    }
12. VERIFY Tracking_Value = 0

7.3.2.X54.22 - Lighting Output Present Value between 0.0 and 1.0 Test

Purpose: To verify that writing a value numerically greater than 0.0 but less than 1.0 to Present_Value shall result in Present_Value taking on the value 1.0.

Test Concept: Select a value, V1, which is numerically greater than 0.0 and less than 1.0. Write V1 to Present_Value and verify that Present_Value takes on the value 1.0.

Configuration Requirements: The Lighting Output object, O1, shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Present_Value shall be different from 1.0.

Test Steps:
1. VERIFY Presnet_Value <> 1.0
2. WRITE Present_Value = a value numerically greater than 0.0 but less than 1.0
3. VERIFY Present_Value = 1.0

7.3.2.X54.31 Lighting Command Operation NONE Test

Purpose: To verify that the IUT can execute WriteProperty service requests when an attempt is made to write a value that is outside of the supported range.

Test Concept: The TD writes the Lighting Command Operation NONE to the IUT, and expects Error Class of PROPERTY and an Error Code of VALUE_OUT_OF_RANGE

Test Steps:
1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = O1
   'Property Identifier' = Lighting_Command
   'Property Value' = NONE
3. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE
4. VERIFY (Object1), Lighting_Command = (the value defined for this property in the EPICS)

7.3.2.X54.32 Lighting Command Operation FADE_TO Test

Purpose: To verify the correct operation of FADE_TO lighting command by observing the value of Present_Value, In_Progress and Tracking_Value.

Test Concept: The TD writes to the Present_Value at each end of the range (i.e. 0% or 100%), and then writes to the Lighting Command Operation with FADE_TO with a long enough fade-time to allow In_Progress and Tracking_Value to be observed while set to FADE_ACTIVE. The Tracking_Value will be checked at the end of the fade to verify that it tracked the target level. The IUT shall be tested for fade up (0% to 100%) and fade down (100% to 0%).

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. V1 > 1 and V2 < 100%

Test Steps:
-- Start with 0% Present_Value to test fade up
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a FADE_TO command (operation, target-level, priority, fade-time)
5. WRITE Lighting_Command = (FADE_TO, V1, PTY1, FT)
6. WAIT **Internal Processing Fail Time**
7. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
8. VERIFY Present_Value = V1

-- In a half way of fading up, check In_Progress and Tracking_Value
9. WAIT FT/2
10. VERIFY In_Progress = FADE_ACTIVE,
11. VERIFY Tracking_Value ~=V1 / 2
12. WAIT FT/2

-- When fading up is completed, check In_Progress and Tracking_Value
13. VERIFY In_Progress = IDLE
14. VERIFY Tracking_Value = V1

-- Now repeat the test with 100% Present_Value to test fade down
15. WRITE Present_Value = 100, ARRAY INDEX = PTY1
16. VERIFY Present_Value = 100
17. WAIT **Internal Processing Fail Time**
18. VERIFY Tracking_Value = 100

-- Write a FADE_TO command (operation, target-level, priority, fade-time)
19. WRITE Lighting_Command = (FADE_TO, V2, PTY1, FT)
20. WAIT **Internal Processing Fail Time**
21. VERIFY Priority_ Array = V2, ARRAY INDEX = PTY1
22. VERIFY Present_Value = V2

-- In a half way of fading down, check In_Progress and Tracking_Value
23. WAIT FT/2
24. VERIFY In_Progress = FADE_ACTIVE,
25. VERIFY Tracking_Value ~=V1 / 2
26. WAIT FT/2

-- When fading down is completed, check In_Progress and Tracking_Value
27. VERIFY In_Progress = IDLE
28. VERIFY Tracking_Value = V2

7.3.2.X54.33 Lighting Command Operation RAMP_TO Test

Purpose: To verify the correct operation of RAMP_TO lighting command by observing the value of Present_Value, In_Progress and Tracking_Value.

Test Concept: The TD writes to Present_Value at each end of the range (i.e. 0% or 100%), and then writes to the Lighting Command Operation with RAMP_TO with a slow enough ramp rate to allow In_Progress and Tracking_Value to be observed while set to RAMP_ACTIVE. The Tracking_Value will be checked at the end of the ramp to verify that it tracked the target level. The IUT shall be tested for ramp up (0% to 100%) and ramp down (100% to 0%).

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. V1 > 1 and V2 < 100%

Test Steps:
-- Start with 0% Present_Value to test ramp up
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a RAMP_TO command (operation, target-value, priority, ramp-rate)
5. WRITE Lighting_Command = (RAMP_TO, V1, PTY1, any valid rate)
6. WAIT **Internal Processing Fail Time**
7. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
8. VERIFY Present_Value =V1

-- Check In_Progress while ramping up
9. VERIFY In_Progress = RAMP_ACTIVE

-- Make sure that Tracking_Value increases with the ramp-rate
10. WHILE (In_Progress <> IDLE) DO {
11. VERIFY Tracking_Value > 0 < V1

12. CHECK (Tracking_Value is increasing with the ramp-rate)}

-- When ramping up is completed, check In_Progress and Tracking_Value
13. VERIFY In_Progress = IDLE
14. VERIFY Tracking_Value = V1

-- Now repeat the test with 100% Present_Value to test ramp down
15. WRITE Present_Value = 100, ARRAY INDEX = PTY1
16. VERIFY Present_Value = 100
17. WAIT **Internal Processing Fail Time**
18. VERIFY Tracking_Value = 100

-- Write a RAMP_TO command (operation, target-value, priority, ramp-rate)
19. WRITE Lighting_Command = (RAMP_TO, V2, PTY1, any valid rate)
20. WAIT **Internal Processing Fail Time**
21. VERIFY Priority_ Array = V2, ARRAY INDEX = PTY1
22. VERIFY Present_Value = V2

-- Check In_Progress while ramping up
23. VERIFY In_Progress = RAMP_ACTIVE,

-- Make sure that Tracking_Value decreases with the ramp-rate
24. WHILE (In_Progress <> RAMP_ACTIVE) DO {
25. VERIFY Tracking_Value < 0 > V2
26. CHECK (Tracking_Value is decreasing with the ramp-rate)}

-- Check In_Progress and Tracking_Value
27. VERIFY In_Progress = IDLE
28. VERIFY Tracking_Value = V2

7.3.2.X54.34 Lighting Command Operation STEP_UP Test

Purpose: To verify the correct operation of STEP_UP lighting command by observing the value of Present_Value, In_Progress and Tracking_Value.

Test Concept: The TD writes to Present_Value at 0%, and then writes to the Lighting Command Operation with STEP_UP and any step increment. The Tracking_Value shall remain at 0% to ignore the operation. Next, the TD writes to Present_Value at 1%, and then writes to the Lighting Command Operation with STEP_UP and a step increment greater than 99%, the Tracking_Value shall be 100%. The TD writes to Present_Value at 1%, and then writes to the Lighting Command Operation with STEP_UP and a step increment less than 99%, the Tracking_Value shall be 1% plus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:
-- Start with 0% Present_Value
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a STEP_UP command (operation, priority, step-increment)
5. WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking_Value was 0
7. VERIFY Priority_ Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 0
9. VERIFY Tracking_Value = 0

-- Now test with Tracking_Value >0
10. WRITE Present_Value = 1, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 1
12. WAIT **Internal Processing Fail Time**
13. VERIFY Tracking_Value = 1

-- Keep stepping up while continuously checking Priority_Array, Present_Value and Tracking_Value
14. REPEAT X = (1 through (100 - step-increment) by step-increment) DO{
        WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
        WAIT **Internal Processing Fail Time**
        VERIFY Priority_ Array = X + step-increment, ARRAY INDEX = PTY1
        VERIFY Present_Value = X + step-increment
        VERIFY Tracking_Value = X + step-increment
    }
-- Now step up one more time to confirm that the values will not exceed 100
15. WRITE Lighting_Command = (STEP_UP, PTY1, any valid value)
16. WAIT **Internal Processing Fail Time**
17. VERIFY Priority_ Array = 100, ARRAY INDEX = PTY1
18. VERIFY Present_Value =100
19. VERIFY Tracking_Value = 100

7.3.2.X54.35 Lighting Command Operation STEP_ DOWN Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking_Value, that writes resulting in a step below 1% are limited to 1%, and that this command is ignored if the Tracking_Value is 0.0%.

Test Concept: The TD writes to Present_Value at 0%, and then writes to the Lighting Command Operation with STEP_DOWN and any step increment. The Tracking_Value shall remain at 0%. The TD writes to Present_Value at 100%, and then writes to the Lighting Command Operation with STEP_DOWN and a step increment greater than 99%, the Tracking_Value shall be 1%. The TD writes to Present_Value at 100%, and then writes to the Lighting Command Operation with STEP_DOWN and a step increment less than 99%, the Tracking_Value shall be 100% minus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:
-- Start with 0% Present_Value
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a STEP_DOWN command (operation, priority, step-increment)
5. WRITE Lighting_Command = (STEP_ DOWN, PTY1, any valid value)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking_Value was 0
7. VERIFY Priority_ Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 0
9. VERIFY Tracking_Value = 0

-- Now test with Tracking_Value = 100
10. WRITE Present_Value = 100, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 100
12. WAIT **Internal Processing Fail Time**
13. VERIFY Tracking_Value =100

-- Keep stepping down while continuously checking Priority_Array, Present_Value and Tracking_Value

14. REPEAT X = (100 through (1 + step-increment) by step-increment) DO{
    WRITE Lighting_Command = (STEP_ DOWN, PTY1, any valid value)
    WAIT **Internal Processing Fail Time**
    VERIFY Priority_ Array = X - step-increment, ARRAY INDEX = PTY1
    VERIFY Present_Value = X - step-increment
    VERIFY Tracking_Value = X - step-increment
    }

-- Now step down one more time to confirm that the values will not go down below 1

15. WRITE Lighting_Command = (STEP_ DOWN, PTY1, any valid value)
16. WAIT **Internal Processing Fail Time**
17. VERIFY Priority_ Array = 1, ARRAY INDEX = PTY1
18. VERIFY Present_Value =1
19. VERIFY Tracking_Value = 1

7.3.2.X54.36 Lighting Command Operation STEP_ON Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking_Value, that this command will set the Tracking_Value to 1% if the Tracking_Value is 0.0%, and that it otherwise adheres to STEP_UP.

Test Concept: The TD writes to Present_Value at 0%, and then writes to the Lighting Command Operation with STEP_UP and any step increment. The Tracking_Value shall be 1%. The TD writes to Present_Value at 1%, and then writes to the Lighting Command Operation with STEP_UP and a step increment greater than 99%, the Tracking_Value shall be 100%. The TD writes to Present_Value at 1%, and then writes to the Lighting Command Operation with STEP_UP and a step increment less than 99%, the Tracking_Value shall be 1% plus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

-- Start with 0% Present_Value

1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a STEP_ON command (operation, priority, step-increment)

5. WRITE Lighting_Command = (STEP_ON, PTY1, any valid values)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the Present_Value and Tracking_Value became 1

7. VERIFY Priority_ Array = 1, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 1
9. VERIFY Tracking_Value = 1

-- Keep stepping on while continuously checking Priority_Array, Present_Value and Tracking_Value

10. REPEAT X = (1 through (100 – step-increment)) DO {
    WRITE Lighting_Command = (STEP_ON, PTY1, any valid values)
    WAIT **Internal Processing Fail Time**
    VERIFY Priority_ Array = X + step-increment, ARRAY INDEX = PTY1
    VERIFY Present_Value = X + step-increment
    VERIFY Tracking_Value = X + step-increment
    }

-- Now step on one more time to confirm that the values will not exceed 100

11. WRITE Lighting_Command = (STEP_ON, PTY1, any valid values)
12. WAIT **Internal Processing Fail Time**
13. VERIFY Priority_ Array = 100, ARRAY INDEX = PTY1
14. VERIFY Present_Value =100
15. VERIFY Tracking_Value = 100

7.3.2.X54.37 Lighting Command Operation STEP_ OFF Test

Purpose: To verify that writing this Lighting Command Operation is reflected in the Tracking_Value, that writes resulting in a step below 1% are limited to 1%, and that this command is ignored if the Tracking_Value is 0.0%.

Test Concept: The TD writes to Present_Value at 0%, and then writes to the Lighting Command Operation with STEP_DOWN and any step increment. The Tracking_Value shall remain at 0%. The TD writes to Present_Value at 100%, and then writes to the Lighting Command Operation with STEP_DOWN and a step increment greater than 99%, the Tracking_Value shall be 1%. The TD writes to Present_Value at 100%, and then writes to the Lighting Command Operation with STEP_DOWN and a step increment less than 99%, the Tracking_Value shall be 100% minus the step increment.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:
-- Start with 0% Present_Value
1. WRITE Present_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking_Value = 0

-- Write a STEP_OFF command (operation, priority, step-increment)
5. WRITE Lighting_Command = (STEP_ OFF, PTY1, step-increment)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking_Value was 0
7. VERIFY Priority_ Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 0
9. VERIFY Tracking_Value = 0

-- Now test with Tracking_Value = 100
10. WRITE Present_Value = 100, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 100
12. WAIT **Internal Processing Fail Time**
13. VERIFY Tracking_Value =100

-- Keep stepping off while continuously checking Priority_Array, Present_Value and Tracking_Value
14. REPEAT X = (100 through (1 + step-increment)) DO {
        WRITE Lighting_Command = (STEP_ OFF, PTY1, step-increment)
        WAIT **Internal Processing Fail Time**
        VERIFY Priority_ Array = X - step-increment, ARRAY INDEX = PTY1
        VERIFY Present_Value = X - step-increment
        VERIFY Tracking_Value = X - step-increment
    }
-- Confirm that the Present_Value and Tracking_Value become 0 when STEP OFF command is executed while Tracking_Value is 1

15. WRITE Lighting_Command = (STEP_ OFF, PTY1, step-increment)
16. WAIT **Internal Processing Fail Time**
17. VERIFY Priority_ Array = 0, ARRAY INDEX = PTY1
18. VERIFY Present_Value = 0
19. VERIFY Tracking_Value = 0

7.3.2.X54.41 Transition None test

Purpose: To verify that the Tracking_Value property immediately follows the Present_Value property if Transition is NONE.

Test Concept: Setup a Lighting Output object, O1, to use its complete supported value range. Set Present_Value to the highest supported value, and then to the lowest supported value, verifying that there is no delay in the transitions.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. If present, Min_Actual_Value shall be set to 1, and Max_Actual_Value shall be set to 100. Transition shall be set to NONE.

Test Steps:
1. VERIFY Transition = NONE
2. VERIFY In_Progress = IDLE
3. WRITE Present_Value = 100, ARRAY INDEX = PTY1
4. VERIFY In_Progress = IDLE
5. VERIFY Tracking_Value = 100
6. WRITE Present_Value = 1, ARRAY INDEX = PTY1
7. VERIFY In_Progress = IDLE
8. VERIFY Tracking_Value = 1

7.3.2.X54.42 Transition Test

Purpose: To verify that the Lighting Output object transitions using the configured function and transitions at the configured speed when Transition is set to either FADE or RAMP.

Test Concept: Setup a Lighting Output object, O1, to use fading or ramping as the default transition method. Present_Value is changed to V1 which is larger than the initial Present_Value, V0, so that the output will fade or ramp up. Halfway through the process, verify that Tracking_Value is approximately equal to the value halfway between V0 and V1. The physical output shall also be verified that it is fading or ramping from V0 to V1. When the process completes, verify that Tracking_Value reached V1. Repeat the process fading or ramping down from V1 to V2.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Transition property is set to FADE or RAMP, Present_Value is V0 and In_Progress is IDLE.

To test FADE functionality, T is FADE, A is FADE_ACTIVE, W1 and W2 are (Default_Fade_Time / 2), and Default_Fade_Time is sufficiently large so as to allow the intermediate progress checks.

To Test RAMP functionality, T is RAMP, A is RAMP_ACTIVE, W1 is (((V1 – V0) / Default_Ramp_Rate) / 2), W2 is (((V1 – V2) / Default_Ramp_Rate) / 2), and Default_Ramp_Rate is sufficiently small so as to allow the intermediate progress checks.

Test Steps:
1. VERIFY Transition = T
2. VERIFY In_Progress = IDLE
3. V0 = READ Present_Value
4. WRITE Present_Value = V1, ARRAY INDEX = PTY1
5. VERIFY Present_Value = V1
6. WAIT W1
7. VERIFY Tracking_Value ~= (V1 + V0) / 2
8. VERIFY In_Progress = A
9. CHECK (the physical output is fading from V0 to V1)
10. WAIT W1
11. VERIFY In_Progress = IDLE
12. VERIFY Tracking_Value = V1
13. WRITE Present_Value = V2, ARRAY INDEX = PTY1
14. VERIFY Present_Value = V2
15. WAIT W2
16. VERIFY Tracking_Value ~= (V2 + V1) / 2
17. VERIFY In_Progress = A
18. CHECK (the physical output is fading V1 to V2)
19. WAIT W2
20. VERIFY In_Progress = IDLE
21. VERIFY Tracking_Value = V2

7.3.2.X54.51 Feedback_Value Clamping Test

Purpose: To verify that the Feedback_Value remains in the normalized range when the physical lighting output is outside the normalized range.

Test Concept: Set the normalized range to be the largest range supported by the device. Make the physical output be above the normalized range by setting it to the maximum supported value and then shrinking the normalized range. The Feedback_Value is immediately tested to verify that it takes on the value 100.

Reset the normalized range. Make the physical output be below the normalized range by setting it to the minimum supported value and then shrinking the normalized range. The Feedback_Value is immediately tested to verify that it takes on the value 1.

Configuration Requirements: The Lighting Output object, O1, shall be configured to transition slowly when Present_Value changes, such as by ramping, fading or stepping, if possible.

O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1.

Test Steps:

-- Verify Feedback_Value when output is above Max_Actual_Value
1. WRITE Max_Actual_Value = 100
2. WRITE Min_Actual_Value = 1
3. WRITE Present_Value = 100, PRIORITY = PTY1
4. WHILE In_Progress <> IDLE {}
5. WRITE Max_Actual_Value = (Lowest supported Max_Actual_Value)
6. VERIFY Feedback_Value = 100

-- Verify Feedback_Value when output is below Min_Actual_Value
7. WRITE Max_Actual_Value = 100
8. WRITE Min_Actual_Value = 1
9. WRITE Present_Value = 1, PRIORITY = PTY1
10. WHILE In_Progress <> IDLE {}
11. WRITE Min_Actual_Value = (Highest supported Min_Actual_Value)
12. VERIFY Feedback_Value = 1

7.3.2.X54.61 Min_Actual_Value and Max_Actual_Value Test

Purpose: To verify that Min_Actual_Value remains less than Max_Actual_Value and within the allowable range when either is written to a value that would violate these conditions.

Test Concept: Write a value to Min_Actual_Value which is larger than Max_Actual_Value. Verify that Max_Actual_Value became equal to Min_Actual_Value. Next, write a value to Max_Actual_Value which is less than Min_Actual_Value. Verify that Min_Actual_Value became equal to Max_Actual_Value.

Verify that neither Min_Actual_Value nor Max_Actual_Value will accept a value outside the range 1.0 to 100.0.

Configuration Requirements: The IUT shall be configured with a lighting output, O1. Min_Actual_Value shall be set to a value less than Max_Actual_Value, and Max_Actual_Value shall be within the allowable range for Min_Actual_Value and not equal to Min_Actual_Value's maximum supported value. If the IUT cannot be configured to meet these requirements, then this test shall be skipped.

Test Steps:
1. V1 = READ Max_Actual_Value
2. WRITE Min_Actual_Value = V2, a value greater than V1
3. VERIFY Max_Actual_Value = V2
4. WRITE Max_Actual_Value = V3, a value less than V2
5. VERIFY Min_Actual_Value = V3

6. TRANSMIT WritePropertyRequest
   'Object Identifier' = O1,
   'Property Identifier' = Min_Actual_Value,
   'Property Value' = (any value outside the range 1.0 to 100.0)
7. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE
8. TRANSMIT WritePropertyRequest
   'Object Identifier' = O1,
   'Property Identifier' = Max_Actual_Value,
   'Property Value' = (any value outside the range 1.0 to 100.0)
9. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE

7.3.2.X54.62 Min_Actual_Value and Max_Actual_Value Scaling Test

Purpose: To verify that the physical output level changes to the expected scaled value as Present_Value changes.

Test Concept: Set Min_Actual_Value to a value other than the lowest supported minimum value, and set Max_Actual_Value to a value other than the highest support value but larger than Min_Actual_Value.

Then write 1.0 to Present_Value and measure the physical output. Repeat the procedure to measure the physical output after writing 100.0 to Present_Value. After obtaining these upper and lower bound values, write a value between 1.0 and 100.0, measure the physical output, and confirm that the measured value is approximately the same as the expected scaled value.

Configuration Requirements: The IUT shall be configured with a lighting output, O1 that can be observed during the test. O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1 and Out_Of_Service = FALSE.

Test Steps:
1. WRITE Min_Actual_Value = (a supported value that is not the lowest supported value)
2. WRITE Max_Actual_Value = (a supported value which is not the highest support value)
3. WRITE Present_Value = 1.0, ARRAY INDEX = PTY1
4. CHECK(the value of the physical output is Min_Actual_Value)
5. WRITE Present_Value = 100.0, ARRAY INDEX = PTY1
6. CHECK(the value of the physical output is Max_Actual_Value)
7. WRITE Present_Value = (V1, a value between 1.0 and 100.0 exclusive), ARRAY INDEX = PTY1
8. MAKE(measure the value of the physical output and record in MV)
9. CHECK (MV ~= Min_Actual_Value + (V1 / 100) * (Max_Actual_Value – Min_Actual_Value))

[In BTL Specified Tests, modify the tests 8.2.1, 8.2.2, 8.3.1, 8.3.2 to test against Lighting Output]

8.2.1 Change of Value Notification from an Analog Input, Analog Output, *Lighting Output*, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value Object's* Present_Value Property

Reason for Change: Add more primitive value objects. Updated description of the 'List of Values' to improve readability. Updated 'Configuration Requirements'.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Analog Input, Analog Output, *Lighting Output*,~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value 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 is then changed by an amount greater than the COV increment and a notification shall be received. The Present_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by an Analog Input object. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not

possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

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:

REPEAT X = (one supported object of each type from the set Analog Input, Analog Output, *Lighting Output,* ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value*) DO {

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = X,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = X,
   'Property Identifier' = COV_Increment
6. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = X,
   'Property Identifier' = COV_Increment,
   'Property Value' = (a value "increment" that will be used below)
7. IF (Out_Of_Service is writable) THEN
   WRITE X, 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' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (*ReportedPV* = ~~any value appropriate for~~ *the current* Present_Value, ~~and~~ new Status_Flags)
   TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present_Value is now writable) THEN
   WRITE X, Present_Value = (any value that differs ~~from "initial Present_Value"~~ *ReportedPV* by less than "increment")
   ELSE
   MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by less than "increment")
9. WAIT **Notification Fail Time**
10. CHECK (verify that no COV notification was transmitted)
11. IF (Present_Value is now writable) THEN
    WRITE X, Present_Value = (any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by an amount greater than "increment")
    ~~RECEIVE BACnet-SimpleACK-PDU~~
    ELSE
    MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by an amount greater than "increment")
12. BEFORE **NotificationFailTime**
    RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the new Present_Value and new Status_Flags)

13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
    WRITE X, Out_Of_Service = FALSE
    ~~RECEIVE BACnet-SimpleACK-PDU~~

8.2.2 Change of Value Notification from an Analog Input, Analog Output, *Lighting Output*,~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object*'s* Status_Flags Property

Reason for Change: Add more primitive value objects. Updated 'Configuration Requirements'. Removed extraneous SimpleACKs after WRITE statements. Updated descriptive text for 'List of Value' property.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags property of Analog Input, Analog Output, *Lighting Output*,~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags 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

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:

REPEAT X = (one supported object of each type from the set Analog Input, Analog Output, *Lighting Output*,~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value*) DO {

1. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
    'Monitored Object Identifier' = X,
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = X,
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU

5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from initial Status_Flags) |
    MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. ~~IF (WriteProperty is used in step 5) THEN~~
    ~~RECEIVE BACnet-SimpleACK-PDU~~

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' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (~~the initial~~*the current* Present_Value and new Status_Flags)
7. TRANSMIT BACnet-SimpleACK-PDU
8. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Out_Of_Service was changed in step 5) THEN
    WRITE X, Out_Of_Service = FALSE
    ~~RECEIVE BACnet-SimpleACK-PDU~~

8.3.1 Change of Value Notification from ~~an Analog Input, Analog Output, and Analog Value~~ *a Numeric* Object*'s* Present_Value Property

Reason for Change: Addenda 135-2008w-1, and 135-2010i-1. Add more primitive value objects and the Lighting Output object.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value property of Analog Input, Analog Output, *Lighting Output,*~~and~~ Analog Value*, Large Analog Value, Integer Value, and Positive Integer Value* objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.1 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

8.3.2 Change of Value Notification from ~~an Analog~~ ~~Input, Analog Output, and Analog Value~~ *a Numeric* Object*'s* Status_Flags Property

Reason for Change: Addenda 135-2008w-1, and 135-2010i-1. Add more primitive value objects and the Lighting Output object.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status_Flags property of Analog Input, Analog Output, *Lighting Output*~~and~~ Analog Value*, Large Analog Value, Integer Value, and Positive Integer Value* objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.2 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[In BTL Test Plan, modify all existing references to 8.2.1]

BTL - 8.2.1 Change of Value Notification from an Analog Input, Analog Output, Lighting Output, Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object's Present_Value Property
~~**BTL - 8.2.1 Change of Value Notification from an Analog Input, Analog Output, Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object Present_Value Property**~~

[In BTL Test Plan, modify all existing references to 8.2.2]

BTL - 8.2.2 Change of Value Notification from an Analog Input, Analog Output, Lighting Output,and Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object's Status_Flags Property
~~**BTL - 8.2.2 Change of Value Notification from an Analog Input, Analog Output, Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object Status_Flags Property**~~

[In BTL Test Plan, modify all existing references to 8.3.1]

BTL - 8.3.1 - Change of Value Notification from a Numeric Object's Present_Value Property
~~**BTL - 8.3.1 Change of Value Notification from an Analog Input, Analog Output, and Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object Present_Value Property**~~

[In BTL Test Plan, modify all existing references to 8.3.2]

BTL - 8.3.2 - Change of Value Notification from a Numeric Object's Status_Flags Property
~~**BTL - 8.3.2 Change of Value Notification from an Analog Input, Analog Output, and Analog Value, Large Analog Value, Integer Value, and Positive Integer Value Object Status_Flags Property**~~

BTL-15.2i-2: Non-Zero Value Test Updates [BTLWG-52]

Overview:

Addendum 135-2012*az*-2 specifies similar behavior when non-zero values are written to Change_Of_State_Count and Elapsed_Active_Time.

These changes are not contained in any SSPC proposal.

Changes:

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Change of State Tracking" in the 3.5 Binary Input Object section.]

3.5.4 Supports Change of State Tracking

The properties of ~~binary~~ objects that collectively track state changes, function as required.

~~135.1-2013~~*BTL* - 7.3.1.8 - ~~Binary Object~~ Change Of State Test~~s~~		
	Test Conditionality	~~If all of the state change properties are supported, it must be executed.~~ *If no Binary Input object contains a writable Elapsed_Active_Time then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Input object.*
	Testing Hints	
BTL - 7.3.1.X18 - Non-zero writable State Count Test		
	Test Conditionality	*If no Binary Input object contains a writable Change_Of_State_Count that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Input object.*
	Testing Hints	

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Elapsed Active Time Tracking" in the 3.5 Binary Input Object section.]

3.5.5 Supports Elapsed Active Time Tracking

The properties of ~~binary~~ objects that collectively track active time, function as required.

BTL - 7.3.1.9 - ~~Binary Object~~ Elapsed Active Time Test~~s~~		
	Test Conditionality	~~If all of the active time properties are supported, it must be executed.~~ *If no Binary Input object contains a writable Elapsed_Active_Time then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Input object.*
	Testing Hints	
BTL - 7.3.1.X19 - Non-zero writable Elapsed Active Time Test		
	Test Conditionality	*If no Binary Input object contains a writable Elapsed_Active_Time that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Input object.*
	Testing Hints	

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Change of State Tracking" in the 3.6 Binary Output Object section.]

3.6.5 Supports Change of State Tracking

The properties of ~~binary~~ objects that collectively track state changes, function as required.

~~135.1-2013~~*BTL* - 7.3.1.8 - ~~Binary Object~~ Change Of State Tests		
	Test Conditionality	~~If all of the state change properties are supported, it must be executed.~~ *If no Binary Output object contains a writable Elapsed_Active_Time then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Output object.*
	Testing Hints	
BTL - 7.3.1.X18 - Non-zero writable State Count Test		
	Test Conditionality	*If no Binary Output object contains a writable Change_Of_State_Count that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Output object.*
	Testing Hints	

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Elapsed Active Time Tracking" in the 3.6 Binary Output Object section.]

3.6.6 Supports Elapsed Active Time Tracking

The properties of ~~binary~~ objects that collectively track active time, function as required.

BTL - 7.3.1.9 - ~~Binary Object~~ Elapsed Active Time Tests		
	Test Conditionality	~~If all of the active time properties are supported, it must be executed.~~ *If no Binary Output object contains a writable Elapsed_Active_Time then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Output object.*
	Testing Hints	
BTL - 7.3.1.X19 - Non-zero writable Elapsed Active Time Test		
	Test Conditionality	*If no Binary Output object contains a writable Elapsed_Active_Time that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped*
	Test Directives	*This test shall be performed using a Binary Output object.*
	Testing Hints	

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Change of State Tracking" in the 3.7 Binary Value Object section.]

3.7.3 Supports Change of State Tracking

The properties of ~~binary~~ objects that collectively track state changes, function as required.

~~135.1-2013~~*BTL* - 7.3.1.8 - ~~Binary Object~~ Change Of State Tests		
	Test Conditionality	~~If all of the state change properties are supported, it must be executed.~~ *If no Binary Value object contains a writable Elapsed_Active_Time then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Value object.*
	Testing Hints	
BTL - 7.3.1.X18 - Non-zero writable State Count Test		

	Test Conditionality	*If no Binary Value object contains a writable Change_Of_State_Count that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Value object.*
	Testing Hints	

[In BTL Test Plan, Modify the reference to one test and add a new test to section "Supports Elapsed Active Time Tracking" in the 3.7 Binary Value Object section.]

3.7.4 Supports Elapsed Active Time Tracking

The properties of ~~binary~~ objects that collectively track active time, function as required.

BTL - 7.3.1.9 - ~~Binary Object~~ Elapsed Active Time Tests~~		
	Test Conditionality	~~If all of the active time properties are supported, it must be executed.~~ *If no Binary Value object contains a writable Elapsed_Active_Timethen this test shall be skipped..*
	Test Directives	*This test shall be performed using a Binary Value object.*
	Testing Hints	
BTL - 7.3.1.X19 - Non-zero writable Elapsed Active Time Test		
	Test Conditionality	*If no Binary Value object contains a writable Elapsed_Active_Time that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.*
	Test Directives	*This test shall be performed using a Binary Value object.*
	Testing Hints	

[The Changes to the Binary Lighting Output Object for COV-B have been applied in section i-3 of this addenda]
[In BTL Specified Tests, add test 7.3.1.8 from 135.1-2013 and modify as noted below.]

7.3.1.8 ~~Binary Object~~ Change of State Tests~~

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

Reason for Change: Renamed test from the 135.1-2013 version and modified steps to express using READ, WRITE and VERIFY commands.

~~BACnet Reference Clauses: 12.6.14, 12.6.15, 12.6.16, 12.7.14, 12.7.15, 12.7.16, 12.8.12, 12.8.13, and 12.8.14.~~

Purpose: To verify that the properties of ~~binary~~ objects that collectively track state changes (changes in Present_Value) function as required. ~~If the Change_Of_State_Count, Change_Of_State_Time, and Time_Of_State_Count_Reset properties are not supported this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.~~

Test Concept: The Present_Value of the ~~binary~~ object under test is changed. The Change_Of_State_Count property is checked to verify that it has been incremented and the Change_Of_State_Time property is checked to verify that it has been updated. The Change_Of_State_Count is reset and Time_Of_State_Count_Reset is checked to verify that it has been updated appropriately.

Configuration Requirements: The object being tested shall be configured such that the Present_Value and Change_Of_State_Count properties are writable or another means of changing these properties shall be provided.

Test Steps:

1. *READ PV = Present_Value*
2. *READ N = Change_of_State_Count*
3. *IF (PV = ACTIVE) THEN*
   IF (Present_Value is writable) THEN

WRITE Present_Value = INACTIVE
VERIFY Present_Value = INACTIVE
ELSE
MAKE (Present_Value = INACTIVE)
ELSE
IF (Present_Value is writable) THEN
WRITE Present_Value = ACTIVE
VERIFY Present_Value = ACTIVE
ELSE
MAKE (Present_Value = ACTIVE)

4. *VERIFY (Change_of_State_Count = N+1)*
5. *VERIFY (Time_Of_State_Count_Time ~= the current local date and time)*
6. *IF (Change_of_State_Count is writable) THEN*
WRITE Change_of_State_Count = 0
ELSE
MAKE (Change_of_State_Count = 0)
7. *VERIFY Time_Of_State_Count_Reset ~= (the current local date and time)*

~~1. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Present_Value~~
~~2. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Present_Value,~~
~~'Property Value' = ACTIVE | INACTIVE~~
~~3. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Change_Of_State_Count~~
~~4. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Change_Of_State_Count,~~
~~'Property Value' = (any valid value, N)~~
~~5. IF (Present_Value is writable) THEN~~
~~IF (the value returned in step 2 was ACTIVE) THEN~~
~~WRITE Present_Value = INACTIVE~~
~~VERIFY Present_Value = INACTIVE~~
~~ELSE~~
~~WRITE Present_Value = ACTIVE~~
~~VERIFY Present_Value = ACTIVE~~
~~ELSE~~
~~MAKE (Present_Value change to the opposite state)~~
~~6. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Date~~
~~7. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local Date,~~
~~'Property Value' = (the current local date, D)~~
~~8. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time~~
~~9. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time,~~
~~'Property Value' = (the current local time, T_{LOC})~~
~~10. WAIT **Internal Processing Fail Time**~~
~~11. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Change_Of_State_Time~~
~~12. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~

~~'Property Identifier' = Change_Of_State_Time,~~
~~'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})~~
~~13. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Change_Of_State_Count~~
~~14. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Change_Of_State_Count,~~
~~'Property Value' = N + 1~~
~~15. IF (Change_Of_State_Count is writable) THEN~~
~~WRITE Change_Of_State_Count = 0~~
~~VERIFY Change_Of_State_Count = 0~~
~~ELSE~~
~~MAKE (Change_Of_State_Count = 0)~~
~~16. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time~~
~~17. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time,~~
~~'Property Value' = (the current local time, T_{LOC})~~
~~18. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Time_Of_State_Count_Reset~~
~~19. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Time_Of_State_Count_Reset,~~
~~'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})~~

[In BTL Specified Tests, add test 7.3.1.9 from 135.1-2013 and modify as noted below.]

7.3.1.9 ~~Binary Object~~ Elapsed Active Time Tests

Reason for Change: Errors were pointed out via BTL-CR-0253, and in order to express using READ, WRITE and VERIFY commands.

~~Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.~~

~~BACnet Reference Clauses: 12.6.17, 12.6.18, 12.7.17, 12.7.18, 12.8.15, and 12.8.16.~~

Purpose: To verify that the properties of ~~binary~~ objects that collectively track active time function properly. ~~If the Elapsed_Active_Time and Time_Of_Active_Time_Reset properties are not supported then this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.~~

Test Concept: The Present_Value of the ~~binary~~ object being tested is set to INACTIVE. The Elapsed_Active_Time property is checked to verify that it does not accumulate time while the object is in an INACTIVE state. The Present_Value is then set to ACTIVE. The Elapsed_Active_Time property is checked to verify that it is accumulating time while the object is in an ACTIVE state. The Present_Value is then set to INACTIVE and the Elapsed_Active_Time is reset. The Time_Of_Active_Time_Reset property is checked to verify that it has been updated.

Configuration Requirements: The object being tested shall be configured such that the Present_Value and Elapsed_Active_Time properties are writable or another means of changing these properties shall be provided.

Test Steps:

1. *IF (Present_Value is writable) THEN*
   WRITE Present_Value = INACTIVE
   VERIFY Present_Value = INACTIVE
   ELSE
   MAKE (Present_Value = INACTIVE)
2. *READ AT = Elapsed_Active_Time*

3. *WAIT more than **Internal_Processing Fail Time** + at least 1 second*
4. *VERIFY (Elapsed_Active_Time = AT)*
5. *IF (Present_Value is writable) THEN*
   WRITE Present_Value = ACTIVE
   VERIFY Present_Value = ACTIVE
   ELSE
   MAKE (Present_Value = ACTIVE)
6. *WAIT more than **Internal_Processing Fail Time** + at least 1 second*
7. *VERIFY (Elapsed_Active_Time > AT)*

~~1. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = INACTIVE~~
~~VERIFY Present_Value = INACTIVE~~
~~ELSE~~
~~MAKE (Present_Value = INACTIVE)~~
~~2. TRANSMIT ReadProperty Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time~~
~~3. RECEIVE ReadProperty ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time,~~
~~'Property Value' = (the elapsed active time, $T_{ELAPSED}$ in seconds)~~
~~4. WAIT (1 minute)~~
~~5. TRANSMIT ReadProperty Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time~~
~~6. RECEIVE ReadProperty ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time,~~
~~'Property Value' = (the same $T_{ELAPSED}$ as step 3)~~
~~7. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = ACTIVE~~
~~VERIFY Present_Value = ACTIVE~~
~~ELSE~~
~~MAKE (Present_Value = ACTIVE)~~

8. WAIT (**Internal Processing Fail Time** + 30 seconds)
9. *IF (Present_Value is writable) THEN*
   WRITE Present_Value = INACTIVE
   VERIFY Present_Value = INACTIVE
   ELSE
   MAKE (Present_Value = INACTIVE)
10. *VERIFY Elapsed_Active_Time ~= ($T_{ELAPSED}$ + 30)*

~~10. TRANSMIT ReadProperty Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time~~
~~11. RECEIVE ReadProperty ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Elapsed_Active_Time,~~
~~'Property Value' = (T: ($T_{ELAPSED}$ + 30) ≤ T ≤ ($T_{ELAPSED}$ 30 + **Internal Processing Fail Time**))~~
~~11. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = INACTIVE~~
~~VERIFY Present_Value = INACTIVE~~
~~ELSE~~
~~MAKE (Present_Value = INACTIVE)~~

11~~12~~. IF (Elapsed_Active_Time is writable) THEN
WRITE Elapsed_Active_Time = 0
VERIFY Elapsed_Active_Time = 0
ELSE
MAKE (Elapsed_Active_Time = 0)

12. *VERIFY Time_Of_Active_Time_Reset ~= (the current local date and time)*

~~13. TRANSMIT ReadProperty Request,~~

~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Date~~
~~14. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local Date,~~
~~'Property Value' = (the current local date, D)~~
~~15. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time~~
~~16. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Local_Time,~~
~~'Property Value' = (the current local time, T_{LOC})~~
~~17. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Time_Of_Active_Time_Reset~~
~~18. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the object being tested),~~
~~'Property Identifier' = Present_Value*Time_Of_Active_Time_Reset*,~~
~~'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})~~

[In BTL Specified Tests, Add the following new test.]

7.3.1.X18 Non-zero writable State Count Test
Reason for Change: Additional behavior was specified in 135-2012*az*-1 and 135-2012*az*-2 when the Change_of_State_Count property accepts writes of non-zero values.

Purpose: To verify that the properties of objects that count the number of transitions and the time when that number of the transitions tracking started function properly.

Test Concept: The Change_of_State_Count property is set with a non-zero value. The Time_Of_State_Count_Reset property is checked to verify that it has not been updated. The Time_Of_State_Count_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall contain Change_of_State_Count and Time_Of_State_Count_Reset properties, and its Change_of_State_Count property must accept writes of non-zero values.

Test Steps:

1. READ TSCR = Time_Of_State_Count_Reset
2. WRITE Change_of_State_Count = (a value > 0)
3. VERIFY (Time_Of_State_Count_Reset = TSCR, i.e. is unchanged)
4. WRITE Time_Of_State_Count_Reset = (a valid value, T1)
5. VERIFY Time_Of_State_Count_Reset = T1

BTL-15.2i-3: Binary Lighting Output Object Tests [BTLWG-616]

Overview:

Add testing for the Binary Lighting Output Object.

This addenda replaces the following sections in the Interim Tests document: BTL-TP15.0-2.1.0, BTL-TP15.1-2.2.0, BTL-TP15.1-2.3.0, BTL-TP15.1-2.4.0
This addenda moves and updates sections BTL-TP15.0-2.1.0, BTL-TP15.1-2.2.0, BTL-TP15.1-2.3.0, and BTL-TP15.1-2.4.0 from the Interim Tests document into the Test Plan, Checklist and Specified Tests documents.

Changes:

[In Interim Tests, remove the following sections: BTL-TP15.0-2.1.0, BTL-TP15.1-2.2.0, BTL-TP15.1-2.3.0, BTL-TP15.1-2.4.0**]**

[In BTL Checklist, modify Binary Lighting Output object type to Section 3, Objects]

~~Support~~	~~Listing~~	**Option**
Binary Lighting Output Object		
	R~~[1,2]~~	Base Requirements
	R	*Supports command prioritization*
	S	*Supports writable Out_Of_Service properties*
	O	*Supports blink-warn*
	O	*Supports writable Polarity property*
	O	*Supports strike count tracking*
	O	*Supports elapsed active time tracking*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
~~[1]Contact BTL for interim tests for this object.~~ ~~[2]Protocol_Revision 16 or higher must be claimed~~		

[In BTL Test Plan, modify Binary Lighting Output object tests in section 3.55]

3.55 Binary Lighting Output Object

3.55.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Binary Lighting Output objects. There are no base requirements for this object.*

[In BTL Test Plan, add the following all new sections to Binary Lighting Output Object.]

3.55.2 Supports Command Prioritization

135.1-2013 - 7.3.1.2 - Relinquish Default Test		
	Test Conditionality	If no object can be made to meet the configuration requirements, this test shall be skipped.
	Test Directives	

	Testing Hints	
135.1-2013 - 7.3.1.3 - Command Prioritization Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.55.3 Supports Writable Out_Of_Service Property

The Out_Of_Service property in Binary Lighting Output objects contained in the IUT are writable.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	If Out_Of_Service is writable, this test must be executed.
	Test Directives	
	Testing Hints	

3.55.4 Supports Blink-Warn

The IUT supports blink-warn in the Binary Lighting Output object. The Blink_Warn_Enable property is true or can be set to true.

BTL - 7.3.1.X41.Y1 - Blink-Warn WARN Command Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y2 - Blink-Warn WARN_OFF Command Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y3 - Blink-Warn WARN_RELINQUISH Command Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y4 - Blink-Warn STOP Command Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test with WARN_OFF and WARN_RELINQUISH commands
	Testing Hints	
BTL - 7.3.1.X41.Y5 - Blink-Warn WARN Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	Repeat the test with WARN_OFF and WARN_RELINQUISH commands
	Testing Hints	
BTL - 7.3.1.X41.Y6 - Blink-Warn WARN_OFF Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y7 - Blink-Warn WARN_RELINQUISH Command Failure Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y8 - Blink-Warn WARN_OFF Command Halted Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X41.Y9 - Blink-Warn WARN_RELINQUISH Command Halted Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.55.5 Supports Writable Polarity Property

The IUT supports a writable Polarity property in the Binary Output object.

135.1-2013 - 7.3.2.6.3 - Polarity Property Tests		
	Test Conditionality	Must be executed
	Test Directives	
	Testing Hints	

3.55.6 Supports Strike Count Tracking

The properties of the Binary Lighting Output object that collectively track strike counts, function as required.

BTL - 7.3.2.X41.Y10 - Strike Count Tests		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X20 - Non-zero writable Strike Count Test		
	Test Conditionality	If no Binary Lighting Output object contains a writable Strike_Count that accepts writes of non-zero values then this test shall be executed. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.
	Test Directives	This test shall be performed using a Binary Lighting Output object.
	Testing Hints	

3.55.7 Supports Elapsed Active Time Tracking

The properties of objects that collectively track active time, function as required.

BTL - 7.3.1.9 - Elapsed Active Time Test		
	Test Conditionality	If no Binary Lighting Output object contains a writable Elapsed_Active_Time then this test shall be skipped.
	Test Directives	This test shall be performed using a Binary Lighting Output object.
	Testing Hints	
BTL - 7.3.1.X19 - Non-zero writable Elapsed Active Time Test		
	Test Conditionality	If no Binary Lighting Output object contains a writable Elapsed_Active_Time that accepts writes of non-zero values then this test shall be skipped. If IUT claims Protocol_Revision less than 16, then this test shall be skipped.
	Test Directives	This test shall be performed using a Binary Lighting Output object.
	Testing Hints	

3.55.8 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	If no object exists in the IUT for which fault conditions can be generated, then this test shall be skipped.
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	If no object exists in the IUT for which fault conditions can be generated, then this test shall be skipped.
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Blink-Warn specific tests in section 7.3.1]

7.3.1.X41.Y1 Blink-Warn WARN Command Test

Purpose: To verify the correct operation of the blink-warn WARN command.

Test Concept: Select an object O1 that supports blink-warn WARN command. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority_Array value at priority PTY1 remains.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN	-1.0 if PROP_REF = Present_Value, otherwise WARN
V1	ON	>1.0

Test Steps:
1. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Active = FALSE
4. WRITE PROP_REF = C1, PRIORITY = PTY1
5. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
6. VERIFY Egress_Active = FALSE
7. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1

7.3.1.X41.Y2 Blink-Warn WARN_OFF Command Test

Purpose: To verify the correct operation of the blink-warn WARN_OFF command.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN_OFF command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority_Array value at priority PTY1 after Egress_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Active is FALSE, and Egress_Time is a non-zero value.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_OFF	-3.0 if PROP_REF = Present_Value, otherwise WARN_OFF
V1	ON	>1.0
V2	OFF	0.0

Test Steps:
1. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0

4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
8. WHILE (Egress_Active = TRUE)
   VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY (T1 – T2) ~= Egress_Time +/- **Internal Processing Fail Time**
11. VERIFY Priority_ Array = V2, ARRAY INDEX = PTY1

7.3.1.X41.Y3 Blink-Warn WARN_RELINQUISH Command Test

Purpose: To verify the correct operation of the blink-warn WARN_RELINQUISH commands.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN_RELINQUISH command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority_Array value at priority PTY1 after Egress_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, and Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_OFF
V0	NULL or OFF	NULL or 0.0
V1	ON	>1.0
V2	OFF	0.0

Test Steps:
1. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
8. WHILE (Egress_Active = TRUE)
   VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY (T1 – T2) ~ = Egress_Time +/- **Internal Processing Fail Time**
11. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1

7.3.1.X41.Y4 Blink-Warn STOP Command Test

Purpose: To verify the correct operation of the blink-warn STOP command.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate that blink-warn occurs. Before the Egress_Time times out, STOP the egress process and validate the Priority_Array value at PTY1 remains equal to V1 after Egress_Time.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, and Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Lighting_Command
C1	WARN_RELINQUISH or WARN_OFF	WARN_RELINQUISH or WARN_OFF
V0	NULL or OFF	NULL or 0.0
V1	ON	>1.0
V2	OFF	0.0

Test Steps:
1. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
8. VERIFY Egress_Active = TRUE
9. WAIT less than Egress_Time
   WRITE PROP_REF = STOP, PRIORITY = PTY1
10. T2 = current local time
11. WAIT **Internal Processing Fail Time**
12. VERIFY Egress_Active = FALSE
13. WAIT Egress_Time – (T2 – T1) + **Internal Processing Fail Time**
14. VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1

7.3.1.X41.Y5 Blink-Warn WARN Command Failure Test

Purpose: To verify blink-warn WARN command does not occur when, the specified priority is not the highest active priority, the value at the specified priority is off or Blink_Warn_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority_Array is not affected.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Select a priority, PTY2, which is numerically less than PTY1 and not equal to 6. Blink_Warn_Enable is TRUE, Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN	-1.0 if PROP_REF = Present_Value, otherwise WARN
V1, V2	ON	>1.0
V3	OFF	0.0

Test Steps:
-- Test for the specified priority is not the highest active priority
1. VERIFY Blink_Warn_Enable = TRUE
2. WRITE Present_Value = V1, PRIORITY = PTY1
3. VERIFY Egress_Active = FALSE
4. WRITE Present_Value = V2, PRIORITY = PTY2

5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. WAIT **Internal Processing Fail Time**
   CHECK (blink-warn did not occur)
7. VERIFY Egress_Active = FALSE
8. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
9. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is either OFF or 0.0
10. WRITE Present_Value = V3, PRIORITY = PTY1
11. WRITE PROP_REF = C1, PRIORITY = PTY1
12. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
13. VERIFY Egress_Active = FALSE
14. VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1
15. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for Blink_Warn_Enable is FALSE
16. IF (Blink_Warn_Enable is writable) THEN
    WRITE Blink_Warn_Enable = FALSE
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_ Array = V1, ARRAY INDEX = PTY1

7.3.1.X41.Y6 Blink-Warn WARN_OFF Command Failure Test

Purpose: To verify blink-warn WARN_OFF command does not occur when the specified priority is not the highest active priority, the Present_Value is either 0.0 or OFF, or Blink_Warn_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value and Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_OFF	-3.0 if PROP_REF = Present_Value, otherwise WARN_OFF
V1, V2	ON	>1.0
V3	OFF	0.0

Test Steps:
-- Test for the specified priority is not the highest active priority
1. VERIFY Blink_Warn_Enable = TRUE
2. VERIFY Egress_Time > 0
3. WRITE Present_Value = V1, PRIORITY = PTY1
4. VERIFY Egress_Active = FALSE
5. WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP_REF = C1, PRIORITY = PTY1
7. WAIT **Internal Processing Fail Time**
   CHECK (blink-warn did not occur)
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1
10. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for the Present_Value is OFF or 0.0
11. WRITE Present_Value = V3, PRIORITY = PTY2, a value not equal to 6 and less than PTY1

12. WRITE PROP_REF = C1, PRIORITY = PTY1
13. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
14. VERIFY Egress_Active = FALSE
15. VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1
16. WRITE Present_Value = NULL, PRIORITY = PTY2
17. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for Blink_Warn_Enable is FALSE
18. IF (Blink_Warn_Enable is writable) THEN
    WRITE Blink_Warn_Enable = FALSE
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1

7.3.1.X41.Y7 Blink-Warn WARN_RELINQUISH Command Failure Test

Purpose: To verify blink-warn WARN_RELINQUISH command does not occur when the specified priority is not the highest active priority, the value at the specified priority is V0, the value of the next highest non-NULL priority, including Relinquish_Default, is V1, or Blink_Warn_Enable is FALSE.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command would generate a blink-warn except set the specified failure conditions. Verify blink-warn does not occur and the Priority_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_RELINQUISH
V0	OFF or NULL	0.0 or NULL
V1	ON	>1.0
V2	OFF	0.0

Test Steps:
-- Test for the specified priority is not the highest active priority
1. VERIFY Blink_Warn_Enable = TRUE
2. VERIFY Egress_Time > 0
3. WRITE Present_Value = V1, PRIORITY = PTY1
4. VERIFY Egress_Active = FALSE
5. WRITE Present_Value = V1, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP_REF = C1, PRIORITY = PTY1
7. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is OFF or 0.0
11. WRITE Present_Value = V2 PRIORITY = PTY1
12. WRITE PROP_REF = C1, PRIORITY = PTY1
13. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)

14. VERIFY Egress_Active = FALSE
15. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1

-- Test for the value at the specified priority is NULL
16. WRITE Present_Value = NULL, PRIORITY = PTY1
17. WRITE PROP_REF = C1, PRIORITY = PTY1
18. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
19. VERIFY Egress_Active = FALSE
20. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1

-- Test for the value of the next highest non-NULL priority is ON or > 1.0
21. WRITE Present_Value = V1 PRIORITY = PTY1
22. WRITE Present_Value = V1, PRIORITY = PTY3, a value numerically greater than PTY1
23. WRITE PROP_REF = C1, PRIORITY = PTY1
24. WAIT **Internal Processing Fail Time**
    CHECK (blink-warn did not occur)
25. VERIFY Egress_Active = FALSE
26. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1
27. WRITE Present_Value = NULL, PRIORITY = PTY3

-- Test for the value of Relinquish_Default is ON or > 1.0
28. IF (Relinquish_Default is writable) THEN
    WRITE Present_Value = V1, PRIORITY = PTY1
    WRITE Relinquish_Default = V1
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT **Internal Processing Fail Time**
        CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1
    WRITE Relinquish_Default = V2

-- Test for Blink_Warn_Enable is FALSE
29. IF (Blink_Warn_Enable is writable) THEN
    WRITE Present_Value = V1, PRIORITY = PTY1
    WRITE Blink_Warn_Enable = FALSE
    WRITE PROP_REF = C1, PRIORITY = PTY1
    WAIT **Internal Processing Fail Time**
        CHECK (blink-warn did not occur)
    VERIFY Egress_Active = FALSE
    VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1

7.3.1.X41.Y8 Blink-Warn WARN_OFF Command Halted Test

Purpose: To verify blink-warn WARN_OFF execution is halted when a higher priority entry is written or the Present_Value at the specified priority is changed.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command will generate a blink-warn. Before the Egress timer expires, verify the specified actions clear the blink-warn properties and the Priority_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value and Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_OFF	-3.0 if PROP_REF = Present_Value, otherwise WARN_OFF
V1 to V3	ON	>1.0

V4	OFF	0.0

Test Steps:
-- Test for a higher priority entry is written to a non NULL value
1. WRITE Present_Value = V1, PRIORITY = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
7. BEFORE Egress_Active = FALSE
   WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_ Array = V4, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the Present_Value at the specified property is changed
11. WRITE Present_Value = V1, PRIORITY = PTY1
12. VERIFY Blink_Warn_Enable = TRUE
13. VERIFY Egress_Time > 0
14. VERIFY Egress_Active = FALSE
15. WRITE PROP_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**
    CHECK (blink-warn occurred)
17. BEFORE Egress_Active = FALSE
    WRITE Present_Value = V3, PRIORITY = PTY1
18. VERIFY Egress_Active = FALSE
19. VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1

7.3.1.X41.Y9 Blink-Warn WARN_RELINQUISH Command Halted Test

Purpose: To verify blink-warn WARN_RELINQUISH execution is halted when a higher priority entry is written or the Present_Value at the specified priority is changed.

Test Concept: Select an object O1 that supports blink-warn commands. Configure O1 such that a blink-warn command will generate a blink-warn. Before the Egress timer expires, verify the specified actions clear the blink-warn properties and the Priority_Array is correctly changed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V1 and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, Egress_Active is FALSE.

	Binary Lighting Output object	**Lighting Output object**
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_RELINQUISH
V0	OFF or NULL	0.0 or NULL
V1	OFF	0.0
V2	ON	>1.0
V3	OFF	any value >1.0 and not equal to V2

Test Steps:
-- Test for a higher priority entry is written to a non NULL value
1. WRITE Present_Value = V2, PRIORITY = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE

5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
7. BEFORE Egress_Active = FALSE
   WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_ Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the Present_Value at the specified property is changed
11. WRITE Present_Value = V2, PRIORITY = PTY1
12. VERIFY Blink_Warn_Enable = TRUE
13. VERIFY Egress_Time > 0
14. VERIFY Egress_Active = FALSE
15. WRITE PROP_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**
   CHECK (blink-warn occurred)
17. BEFORE Egress_Active = FALSE
   WRITE Present_Value = V3, PRIORITY = PTY1
18. VERIFY Egress_Active = FALSE
19. VERIFY Priority_ Array = V3, ARRAY INDEX = PTY1

[In BTL Specified Tests, add new Binary Light Output Object specific test for Strike Count to section 7.3.2]

7.3.2.X41.Y10 Strike Count Test

Purpose: To verify that the properties of an object (O1) that tracks strike counts.

Test Concept: The Present_Value or Feedback_Value of O1 can be used as the source S1 to increment Strike_Count. S1 is transitioned from OFF to ON. The Strike_Count property is checked to verify that it has been incremented. The Strike_Count is reset and Time_Of_Strike_Count_Reset is checked to verify that it has been updated appropriately. Strike_Count is set to a non-zero value and the Time_Of_Strike_Count_Reset is unchanged.

Configuration Requirements: O1 shall be configured such that the Present_Value property is writable or another means of changing these properties shall be provided.

Test Steps:
1. C1 = Strike_Count
2. MAKE (S1 transition OFF to ON)
3. VERIFY (Strike_Count = C1 + 1)
4. IF (Strike_Count is writable) THEN
   MAKE (Strike_Count = 0)
   VERIFY (Time_Of_Strike_Count_Reset = current local time)
5. IF (Strike_Count is writable to a non-zero value) THEN
   MAKE (Strike_Count > 0)
   VERIFY (Time_Of_Strike_Count_Reset is unchanged)

[In BTL Specified Tests, delete Notes to Tester from clause 8.5.X9.1
Strike through is not normally used for changes to BTL created tests; it is used here for clarity]

8.5.X9.1 CHANGE_OF_RELIABILITY with No Fault Algorithm

…

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

[In BTL Checklist, remove the “contact BTL” footnote and the Protocol Revision 16 footnote, on object type Binary Lighting Output in existing 4.9 DS-COV-A testing]

Support	Listing	Option
Data Sharing - Change of Value - A		
	R	Base Requirements
	C[2]	**...**
	C[2,~~4,5~~]	Can Subscribe for COV from Binary Lighting Output objects
	O	Can Subscribe for COV from proprietary objects
[1] At least one of these options is required in order to claim conformance to this BIBB. [2] At least one of these options is required in order to claim conformance to this BIBB. [3] Support for this option is suggested except in the case where the device is able to generate infinite subscriptions in which case it is required. [4]Contact BTL for interim tests for this object. ~~[5]Protocol_Revision 16 or higher must be claimed.~~		

[In BTL Checklist, remove the “contact BTL” footnote and the "Protocol Revision" footnote on object type Binary Lighting Output in section 4.10 DS-COV-B testing]

Support	Listing	Option
Data Sharing - Change of Value - B		
	R	Base Requirements
	C[1]	**...**
	C[1,~~2,3~~]	Supports COV for Binary Lighting Output objects
	O	Supports COV for proprietary Objects
[1] At least one of these options is required in order to claim conformance to this BIBB. [2]Contact BTL for interim tests for this object. ~~[3]Protocol_Revision 16 or higher must be claimed.~~		

[In BTL Test Plan, add the following tests into the existing section 4.10.33 found in DS-COV-B]

4.10.33 Supports COV for Binary Lighting Output Objects

The IUT supports change of value notifications for at least one object of type Binary Lighting Output.

~~Contact BTL for interim tests for this object.~~

BTL - 8.2.3 - Change of Value Notification from a Discrete Valued Object Present_Value Property		
	Test Conditionality	*This may be skipped if 8.3.3 is executed against a Binary Lighting Output object.*
	Test Directives	*The selected object must be a Binary Lighting Output object.*
	Testing Hints	
BTL - 8.2.4 - Change of Value Notification from a Discrete Valued Object Status_Flags Property		
	Test Conditionality	*This may be skipped if 8.3.4 is executed against a Binary Lighting Output object.*
	Test Directives	*The selected object must be a Binary Lighting Output object.*
	Testing Hints	
BTL - 8.3.3 - Change of Value Notification from a Discrete Valued Object Present_Value Property		
	Test Conditionality	*This may be skipped if 8.2.3 is executed against a Binary Lighting Output object.*
	Test Directives	*The selected object must be a Binary Lighting Output object.*

	Testing Hints	
BTL - 8.3.4 - Change of Value Notification from a Discrete Valued Object Status_Flags Property		
	Test Conditionality	*This may be skipped if 8.2.4 is executed against a Binary Lighting Output object.*
	Test Directives	*The selected object must be a Binary Lighting Output object.*
	Testing Hints	

[In BTL Specified Tests, modify the existing tests 8.23, and 8.24.]

~~***8.2.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present_Value Property***~~

8.2.3 Change of Value Notification from a Discrete Valued Object's Present_Value Property

Reason for Change: Updated the 'Configuration Requirements'. Removed extraneous SimpleACKs that appear after WRITE statements. Modified descriptive text for 'List of Values' properties.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Binary ~~Input, Binary Output, and Binary Value~~ objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed and a notification shall be received. The Present_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by a Binary Input object. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

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

Test Steps:

REPEAT X = (one of each supported binary object type ~~Input, Binary Output, and Binary Value~~) DO {

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = X,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out_Of_Service is writable) THEN
   WRITE X, 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' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (~~the initial~~*ReportedPV = the current* Present_Value, ~~and~~ new *Current* Status_Flags)

TRANSMIT BACnet-SimpleACK-PDU

6. IF (Present_Value is now writable) THEN
   WRITE X, Present_Value = (any value that differs from ~~"initial Present_Value"~~ *ReportedPV*)
   ELSE
   MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV*)
7. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the new Present_Value and new Status_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out_Of_Service is writable) THEN
    WRITE X, Out_Of_Service = FALSE
    ~~RECEIVE BACnet-SimpleACK-PDU~~

~~**8.2.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property**~~

8.2.4 Change of Value Notification from a Discrete Valued Object Status_Flags Property

Reason for Change: Updated 'Test Concept' to include case if finite lifetime is not supported. Updated 'Configuration Requirements'.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags property of Binary ~~Input, Binary Output, and Binary Value~~ objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. ~~Removed extraneous SimpleACKs after WRITE statements.~~ *L shall be set to a value less than 8 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.

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

Test Steps:

REPEAT X = (one of each supported binary object type ~~Input, Binary Output, and Binary Value~~) DO {

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = X,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU

5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from initial Status_Flags) | MAKE (Status_Flags = any value that differs from initial Status_Flags)
~~6. IF (WriteProperty is used in step 5) THEN~~
~~RECEIVE BACnet-SimpleACK-PDU~~
~~7~~*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' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (~~the initial~~*the current* Present_Value, and new Status_Flags)
~~8~~*7*. TRANSMIT BACnet-SimpleACK-PDU
~~9~~*8*. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = X
~~10~~*9*.RECEIVE BACnet-SimpleACK-PDU
~~11~~*10* IF (Out_Of_Service was changed in step 5) THEN
WRITE X, Out_Of_Service = FALSE
~~RECEIVE BACnet-SimpleACK-PDU~~

[In BTL Specified Tests, add the below updated 135.1-2013 tests]

~~**8.3.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present_Value Property**~~

8.3.3 Change of Value Notification from a Discrete Valued Object Present_Value Property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value property of Binary ~~Input, Binary Output, and Binary Value~~ objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.3 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

~~**8.3.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property**~~

8.3.4 Change of Value Notification from a Discrete Valued Object Status_Flags Property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status_Flags property of Binary ~~Input, Binary Output, and Binary Value~~ objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.4 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

[In BTL Test Plan, modify all existing references to 8.2.3]

BTL - 8.2.3 Change of Value Notification from a Discrete Valued Object's Present_Value Property
~~**BTL - 8.2.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present_Value Property**~~

[In BTL Test Plan, modify all existing references to 8.2.4]

BTL - 8.2.4 Change of Value Notification from a Discrete Valued Object Status_Flags Property
~~**BTL - 8.2.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property**~~

[In BTL Test Plan, modify all existing references to 8.3.3]

BTL - 8.3.3 Change of Value Notification from a Discrete Valued Object Present_Value Property
~~**135.1 - 8.3.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present_Value Property**~~

[In BTL Test Plan, modify all existing references to 8.3.4]

BTL - 8.3.4 Change of Value Notification from a Discrete Valued Object Status_Flags Property
~~**135.1 - 8.3.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property**~~

[In BTL Specified Tests, Add the following 2 new tests for use in Binary Light Output Object testing]

7.3.1.X19 Non-zero writable Elapsed Active Time Test
Reason for Change: Additional behavior was specified in 135-2012*az*-1 and 135-2012*az*-2 when the Elapsed_Active_Time property accepts writes of non-zero values.

Purpose: To verify that Time_Of_Active_Time_Reset is writable when Elapsed_Active_Time accepts writes of non-zero values and does not automatically change when Elapsed_Active_Time is written to a non-zero value.

Test Concept: The Present_Value or Feedback_Value is made INACTIVE and the Elapsed_Active_Time is set with a non-zero value. The Time_Of_Active_Time_Reset property is checked to verify that it has not been updated. The Time_Of_Active_Time_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Elapsed_Active_Time and Time_Of_Active_Time_Reset properties are present, and in which the Elapsed_Active_Time property accepts writes of non-zero values.

Test Steps:

1. IF (Present_Value is writable) THEN
       WRITE Present_Value = INACTIVE
       VERIFY Present_Value = INACTIVE
   ELSE
       MAKE (Present_Value = INACTIVE)
2. READ TATR = Time_Of_Active_Time_Reset
3. WRITE Elapsed_Active_Time = a supported non-zero value
4. VERIFY (Time_Of_Active_Time_Reset = TATR, i.e. is unchanged)
5. WRITE Time_Of_Active_Time_Reset = (a valid value, T1)
6. VERIFY Time_Of_Active_Time_Reset = T1

7.3.1.X20 Non-zero writable Strike Count Test
Reason for Change: Additional behavior was specified in 135-2012*az*-1 when the Strike_Count property accepts writes of non-zero values.

Purpose: To verify that Time_Of_Count_Reset is writable when Strike_Count accepts writes of non-zero values and does not automatically change when Strike_Count is written to a non-zero value.

Test Concept: The Strike_Count property is set with a non-zero value. The Time_Of_Strike_Count_Reset property is checked to verify that it has not been updated. The Time_Of_Strike_Count_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Strike_Count and Time_Of_Strike_Count_Reset properties are present, and in which the Strike_Count property accepts writes of non-zero values.

Test Steps:
1. READ TSCR = Time_Of_Strike_Count_Reset
2. WRITE Strike_Count = (a value > 0)
3. VERIFY (Time_Of_Strike_Count_Reset = TSCR, i.e. is unchanged)
4. WRITE Time_Of_Strike_Count_Reset = (a valid value, T1)

5. VERIFY Time_Of_Strike_Count_Reset = T1