BACnet® TESTING LABORATORIES ADDENDA

Addendum ca to BTL Test Package 23.3

Revision final
Revised 10/3/2024

Approved by the BTL Working Group on September 5, 2024;
Approved by the BTL Working Group Voting Members on October 2, 2024;
Published on October 7, 2024.

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

FOREWORD

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

In the following document, language to be added to existing clauses within the BTL Test Package 23.3 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout.

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

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

BTL-23.3 ca-1: Add New Checklist Entries for Addenda ca [BTLWG-1451]

Overview:

Add new checklist items per 135.1-2020 Addenda ca

Changes:

Checklist Changes

Lighting Output Object		
	R	Base Requirements
	R	Supports command prioritization
	R	Supports all BACnetLightingOperations
	S	Supports configurable Out_Of_Service property
	O	Supports blink-warn
	O	Supports Transition property
	O	Supports Feedback_Value property
	O	Supports Min_Actual_Value and Max_Actual_Value properties
	O	Supports the value source mechanism.
	O[1,~~2~~]	Supports Color_Reference property
	O[1,~~2~~]	Supports Color_Override property
	O[1,~~2~~]	Supports the Trim_Fade_Time property
	O[~~3~~2]	Supports intrinsic reporting
[1] Protocol_Revision 24 or higher must be claimed. [~~2~~] ~~Contact BTL for interim tests for this functionality.~~ [~~3~~2] Protocol_Revision 21 or higher must be claimed.		

Binary Lighting Output Object		
	R	Base Requirements
	R	Supports command prioritization
	S	Supports configurable Out_Of_Service property
	O	Supports blink-warn
	O	Supports writable Polarity property
	O	Supports strike count tracking
	O	Supports elapsed active time tracking
	O	Supports the value source mechanism.
	O[1,~~2~~]	Supports Color_Reference property
	O[1,~~2~~]	Supports Color_Override property
[1] Protocol_Revision 24 or higher must be claimed [~~2~~] ~~Contact BTL for interim tests for this functionality~~		

Test Plan Changes

None

Specified Test Changes

None

BTL-23.3 ca-2: Add New Color Object Type [BTLWG-1260]

Overview:

This WI adds testing for the Color object type

Changes:

Checklist Changes

[Replace the Color Object section]

Color Object		
	R	Base Requirements
	O	Supports Transition property
	O	Supports a configurable Default_Fade_Time
	O	Supports the value source mechanism

[Add color objects to Data Management - Object Creation and Deletion A and B sections]

Device Management - Object Creation and Deletion - A		
	…	…
	C[2,8]	Can create and delete Color objects
… [8] Protocol_Revision 24 or higher must be claimed.		
Device Management - Object Creation and Deletion - B		
	…	…
	C[1,6]	Supports object creation and deletion of the Color object
… [6] Protocol_Revision 24 or higher must be claimed.		

Test Plan Changes

[Replace the Color Object section with the below]

3.65 Color Object

3.65.1 Base Requirements

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

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify the IUT claims support for DS-WP-B
	Testing Hints	
BTL - 7.3.2.X67.1 - Color Object Present_Value Startup Test		
	Test Conditionality	Must be executed.
	Test Directives	If the IUT supports configuring the startup color between any specific color and the 'previous color that was in effect prior to restart' then this test shall be run twice, once in each configuration.
	Testing Hints	
BTL - 7.3.2.X67.2 - Transition NONE Test		
	Test Conditionality	This test shall be skipped if the Transition property is supported and cannot be set to NONE.
	Test Directives	
	Testing Hints	

BTL - 7.3.2.X67.3 - Color Object Present_Value Out Of Range Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.4 - Color Object Color_Command Out Of Range Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.5 - Invalid Color_Command Operations Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.6 - FADE_TO_COLOR Color Command Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.8 - Interrupting a Fade In Progress		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.11 - Color_Command Fade-time Out Of Range Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.65.2 Supports Transition Property

The IUT supports the Transition property in Color objects.

BTL - 7.3.2.X67.22 - Color Commands Ignore Transition When Fade-Time is Specified		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

3.65.3 Supports a Configurable Default_Fade_Time

The IUT supports a configurable Default_Fade_Time.

BTL - 7.3.2.X67.31 - Configuring Default_Fade_Time Within Allowable Range		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X67.32 - Writing Default_Fade_Time Positive Test		
	Test Conditionality	Must be executed if the Default_Fade_Time is writable.
	Test Directives	
	Testing Hints	
BTL - 9.22.2.4 - Writing with a Property Value that is Out of Range		
	Test Conditionality	Must be executed if the Default_Fade_Time is writable.
	Test Directives	Execute this test against Default_Fade_Time twice; once using 99 and again using 86400001.
	Testing Hints	

3.65.1 Supports the Value Source Mechanism

The IUT supports the Value Source Mechanism in color objects.

BTL - 7.3.1.X42.Y2 - Non-commandable Value_Source Property Test		
	Test Conditionality	Must be executed.
	Test Directives	

	Testing Hints	

[Add section for DM-OCD-A and DM-OCD-B]

8.21.68 Can Create and Delete Color Objects

The IUT can create and delete Color objects. The IUT shall not restrict the instance number which can be used to create the Color object.

135.1-2023 - 8.16.1 - Creating Objects by Specifying the Object Identifier with no Initial Values		
	Test Conditionality	Must be tested on the Color Object
	Test Directives	
	Testing Hints	
135.1-2023 - 8.17 - Delete Object Service		
	Test Conditionality	Must be tested on the Color Object
	Test Directives	
	Testing Hints	

8.22.68 Supports Object Creation and Deletion of the Color Object

The Color object can be created and deleted within the IUT. The Color object that is created must be the object that can be deleted using the delete service.

135.1-2023 - 9.16.1.1 - Creating Objects by Specifying the Object Type with No Initial Values		
	Test Conditionality	Must be executed.
	Test Directives	Execute using the Color Object.
	Testing Hints	
BTL - 9.16.1.2 - Creating Objects by Specifying the Object Identifier with No Initial Values		
	Test Conditionality	Must be executed.
	Test Directives	Execute using the Color Object.
	Testing Hints	
135.1-2023 - 9.17.1.1 - Successful Deletion of an Object		
	Test Conditionality	Must be executed.
	Test Directives	Execute using the Color Object.
	Testing Hints	

Specified Test Changes

[Add a new test section into BTL Specified Tests]

7.3.2.X67 Color Object Tests

7.3.2.X67.1 Color Object Present_Value Startup Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the Color object's Present_Value goes to either the last Tracking_Value or a default color on startup, depending on the color in the Default_Color property.

Test Concept: The IUT is restarted and Present_Value is verified to go either to Default_Color or the previous color in effect prior to restart if Default_Color is (0,0). The color output in Tracking_Value is verified to go to either Present_Value or the previous color before the restart.

Configuration Requirements: The IUT is not performing any color commands or fades at the beginning of this test. The starting Present_Value, PV1, shall be set to something other than the Default_Color, DC.

Test Steps:

1. VERIFY In_Progress = IDLE

2. READ PV1 = Present_Value
3. READ DC = Default_Color
4. CHECK (PV1 does not equal DC)
5. MAKE (the IUT restart)
6. WAIT (for the IUT to restart)
7. IF (DC = (0,0)) THEN
   VERIFY Present_Value = PV1
   VERIFY In_Progress = IDLE
   ELSE
   VERIFY Present_Value = DC
8. IF (the IUT's color output is updated on startup) THEN
   VERIFY Tracking_Value = (the Present_Value read in the previous step)
   VERIFY In_Progress = IDLE
   ELSE
   VERIFY In_Progress = NOT_CONTROLLED

7.3.2.X67.2 Transition NONE Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when Transition is NONE or not supported, writing to the Present_Value is set to the target color immediately.

Test Concept: Transition is verified as NONE or not supported. Tracking_Value is read. A different value is written to Present_Value and Tracking_Value is read back as equal to Present_Value.

Configuration Requirements: The IUT is not performing any color commands or fades at the start of this test.

Test Steps:

1. IF (Transition property is supported) THEN
   VERIFY Transition = NONE
2. READ TV = Tracking_Value
3. WRITE Present_Value = (C1: any valid color supported by the IUT, other than TV)
4. VERIFY Tracking_Value = C1
5. VERIFY In_Progress = IDLE

7.3.2.X67.3 Color Object Present_Value Out Of Range Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a color which is out of range is written to its Present_Value, depending on the color value written.

Test Concept: Present_Value is read, then written to with a value outside the allowed range of (0,0) to (1,1). An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then a value which is within the allowed range, but outside of the range supported by the IUT, is written to Present_Value. Either an error is returned and Present_Value unchanged, or the value is accepted but the Present_Value is changed to the closest color supported. This is repeated if the IUT does not support the entire CIE chromaticity curve using a value within the curve, but outside of the range supported by the IUT.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1. READ PV1 = Present_Value
2. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Present_Value,
   'Property Value' = (any value outside of the range (0,0) to (1,1))
3. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
4. VERIFY Present_Value = PV1
5. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Present_Value,
   'Property Value' = (any value outside of the curved space of the CIE chromaticity diagram and within the range (0,0) to (1,1))

6. IF (the IUT clamps color values outside the CIE chromaticity curve but within the range (0,0) to (1,1)) THEN
   RECEIVE BACnet-SimpleACK-PDU
   READ PV1 = Present_Value
   IF (Transition is present and set to FADE) THEN
   WAIT (Default_Fade_Time milliseconds)
   CHECK (that PV1 and Tracking_Value are the closest supported color to what was written)
   ELSE
   RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
   VERIFY Present_Value = PV1
7. IF (the IUT does not support all color values within the CIE chromaticity curve) THEN
   TRANSMIT WriteProperty-Request,
   'Property Identifier' = Present_Value,
   'Property Value' = (any value unsupported by the IUT and within the curved space of the CIE chromaticity curve)
   IF (the IUT clamps unsupported color values) THEN
   RECEIVE BACnet-SimpleACK-PDU
   READ PV1 = Present_Value
   IF (Transition is present and set to FADE) THEN
   WAIT (Default_Fade_Time milliseconds)
   CHECK (that PV1 and Tracking_Value are the closest supported color to what was written)
   ELSE
   RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
   VERIFY Present_Value = PV1

7.3.2.X67.4 Color Object Color_Command Out Of Range Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a target color which is out of range is written to its Color_Command, depending on the color written.

Test Concept: Present_Value is read, then Color_Command is written to with a target value outside the allowed range of (0,0) to (1,1). An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then a Color_Command with a target value which is within the allowed range, but outside of the range supported by the IUT, is written to the IUT. Either an error is returned and Present_Value unchanged, or the value is accepted but the Present_Value is changed to the closest color supported. This is repeated if the IUT does not support the entire CIE chromaticity curve.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

1. READ PV1 = Present_Value
2. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Color_Command,
   'Property Value' = (FADE_TO_COLOR, any value outside of the range (0,0) to (1,1))
3. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
4. VERIFY Present_Value = PV1
5. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Color_Command,
   'Property Value' = (FADE_TO_COLOR, any value outside of the curved space of the CIE chromaticity diagram and within the range (0,0) to (1,1), 100)
6. IF (the IUT clamps color values outside the CIE chromaticity curve but within the range (0,0) to (1,1)) THEN
   RECEIVE BACnet-SimpleACK-PDU
   WAIT (100 milliseconds)
   CHECK (Present_Value and Tracking_Value are equal to each other and within the range of colors supported by the IUT)
   ELSE

RECEIVE BACnet-Error-PDU,
Error Class = PROPERTY
Error Code = VALUE_OUT_OF_RANGE
VERIFY Present_Value = PV1
7. IF (the IUT does not support all color values within the CIE chromaticity curve) THEN
READ PV2 = Present_Value
TRANSMIT WriteProperty-Request,
'Property Identifier' = Color_Command,
'Property Value' = (FADE_TO_COLOR, any value unsupported by the IUT and within the curved space of the CIE chromaticity diagram, 100)
IF (the IUT clamps unsupported color values) THEN
RECEIVE BACnet-SimpleACK-PDU
WAIT (100 milliseconds)
CHECK (Present_Value and Tracking_Value are equal to each other and within the range of colors supported by the IUT)
ELSE
RECEIVE BACnet-Error-PDU,
Error Class = PROPERTY
Error Code = VALUE_OUT_OF_RANGE
VERIFY Present_Value = PV2

7.3.2.X67.5 Invalid Color_Command Operations Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct error when invalid color commands are written to the Color_Command property of the Color object.

Test Concept: Present_Value and Color_Command are read, then Color_Command is written to with each unsupported CCT color command. An error is received each time, with Error Class = PROPERTY and Error Code = VALUE_OUT_OF_RANGE. When the error is received, TD verifies the Present_Value has not changed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1 READ CC = Color_Command
2. READ PV = Present_Value
3. REPEAT X = (each invalid Color_Command operation, including NONE and a value not defined) DO {
TRANSMIT WriteProperty-Request,
'Property Identifier' = Color_Command,
'Property Value' = (X)
RECEIVE BACnet-Error-PDU,
Error Class = PROPERTY
Error Code = VALUE_OUT_OF_RANGE
VERIFY Present_Value = PV
VERIFY Color_Command = CC
}

7.3.2.X67.6 FADE_TO_COLOR Color Command Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will accept a FADE_TO_COLOR color command when sent with minimum and maximum fade-times and without a specified fade-time.

Test Concept: Color_Command is written to with Operation = FADE_TO_COLOR and a valid target color, and the minimum fade-time allowed by the standard. Then another Color Command is written with the maximum fade-time allowed by the standard. TD verifies the color fade has started. A final color command is written without a fade-time parameter. TD verifies Present_Value and if Default_Fade_Time is large enough, also verifies In_Progress and Tracking_Value during the fade. After Default_Fade_Time has elapsed, TD verifies the fade is completed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1. WRITE Color_Command = (FADE_TO_COLOR, (C1: any valid color supported by the IUT), 100)
2. WAIT (100 milliseconds)
3. VERIFY In_Progress = IDLE
4. VERIFY Present_Value = C1
5. WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), 86400000)
6. VERIFY In_Progress = FADE_ACTIVE
7. VERIFY Present_Value = C2

-- Send a color command without a fade-time, overwriting the previous one, to verify usage of Default_Fade_Time

8. READ FT = Default_Fade_Time
9. WRITE Color_Command = (FADE_TO_COLOR, C1)
10. IF (FT is large enough for TD to read properties before elapsing) THEN
        BEFORE (Default_Fade_Time milliseconds)
            VERIFY In_Progress = FADE_ACTIVE
            VERIFY Tracking_Value <> C1
11. VERIFY Present_Value = C1
12. WAIT (Default_Fade_Time milliseconds)
13. VERIFY Tracking_Value = C1
14. VERIFY In_Progress = IDLE

7.3.2.X67.8 Interrupting a Fade In Progress

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a fade in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to begin a fade to color with a specified fade-time, then it interrupts the fade by writing to the Present_Value property. The fade should immediately stop and go to the color written in Present_Value, depending on the presence, and value of, the Transition property. Then TD writes the same color command to begin another fade to color. Before this fade elapses, TD interrupts the fade with a color command to go to a different color. TD then interrupts this final color command by writing STOP to the Color_Command property. TD verifies that the final state of Present_Value, In_Progress, and Color_Command.

Configuration Requirements: The IUT should not have a fade in progress at the beginning of this test. If Transition is configurable, it shall be configured to FADE.

Notes to Tester: This test can be made easier by selecting three distinct colors that the IUT supports.

Test Steps:

1. READ C1 = Present_Value
2. VERIFY In_Progress = IDLE
3. WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), 86400000)
4. VERIFY In_Progress = FADE_ACTIVE
5. VERIFY Tracking_Value <> C2
6. VERIFY Present_Value = C2

-- Interrupt the color command fade by writing to Present_Value

7. WRITE Present_Value = (C3: any valid color supported by the IUT other than C1 and C2)
8. IF (Transition property is not present, or set to NONE) THEN
        VERIFY Tracking_Value = C3
        VERIFY In_Progress = IDLE
    ELSE
        VERIFY In_Progress = FADE_ACTIVE

-- Interrupt the fade or start a new one depending on Transition's value

9. WRITE Color_Command = (FADE_TO_COLOR, C1, 86400000)
10. VERIFY Present_Value = C1
11. VERIFY In_Progress = FADE_ACTIVE
12. VERIFY Tracking_Value <> C1

-- Send a different color command, to interrupt the previous one

13. WRITE Color_Command = (FADE_TO_COLOR, C2, 86400000)

-- Interrupt the fade with the STOP command

14. READ TV = Tracking_Value

15. WRITE Color_Command = STOP
16. VERIFY Present_Value = (a value approximately equal to, or equal to, TV)
17. VERIFY In_Progress = IDLE
18. VERIFY Color_Command = STOP

7.3.2.X67.11 Color_Command Fade-time Out Of Range Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a fade-time which is out of range is written to its Color_Command.

Test Concept: Present_Value is read, then a color command is written with a fade-time smaller than the minimum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then another color command is written with a target value which is larger than the maximum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. After each write, TD verifies the Present_Value is unchanged.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

1. READ PV1 = Present_Value
2. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Color_Command,
   'Property Value' = (FADE_TO_COLOR, C1: any valid color, 99)
3. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
4. VERIFY Present_Value = PV1
5. TRANSMIT WriteProperty-Request,
   'Property Identifier' = Color_Command,
   'Property Value' = (FADE_TO_COLOR, , C1, 86400001)
6. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY
   Error Code = VALUE_OUT_OF_RANGE
7. VERIFY Present_Value = PV1

7.3.2.X67.22 Color Commands Ignore Transition When Fade-Time is Specified

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property, writes to Color_Command with a specified fade-time will use that fade time instead of the fade time that is specified by Transition.

Test Concept: TD writes a Color_Command with a fade-time, that is different from Default_Fade_Time and verifies the color fade did not end after Default_Fade_Time milliseconds.

Configuration Requirements: Default_Fade_Time must not be set to 86400000.

Test Steps:

1. VERIFY Transition = FADE | NONE
2. READ C1 = Present_Value
3. WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), 86400000)
4. WAIT (Default_Fade_Time milliseconds)
5. VERIFY In_Progress = FADE_ACTIVE
6. VERIFY Tracking_Value <> C1
7. VERIFY Tracking_Value <> C2
8. WRITE Color_Command = STOP

7.3.2.X67.31 Configuring Default_Fade_Time Within Allowable Range

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT supports a configurable Default_Fade_Time.

Test Concept: The IUT is configured with a different Default_Fade_Time, FT1. If Transition is supported and set to FADE, TD writes to Present_Value and the Tracking_Value is verified to only be equal to the written color after FT1 milliseconds have passed. Otherwise, a color command with fade-time = FT2 is written and Tracking_Value is verified to only be equal to the written color after FT2 milliseconds have passed.

Configuration Requirements: There are no configuration requirements for this test.

Notes to Tester: Sufficiently large fade times should be used when selecting FT1 or FT2, in order to allow TD to read the Tracking_Value after FT0 but before FT1 or FT2 has passed.

Test Steps:

1. READ FT0 = Default_Fade_Time
2. MAKE (configure the IUT such that Default_Fade_Time = FT1: a different fade time longer than FT0)
3. VERIFY FT1 = Default_Fade_Time
4. READ C1 = Present_Value

-- Write to Present_Value to verify Default_Fade_Time gets applied

5. IF (Transition is present and equal to FADE) THEN
   WRITE Present_Value = (C2, a different color than C1)
   VERIFY In_Progress = FADE_ACTIVE
   WAIT (FT0 milliseconds)
   VERIFY Tracking_Value <> C2
   WAIT (FT1 - FT0 milliseconds)
   VERIFY Tracking_Value = C2
   VERIFY In_Progress = IDLE
   ELSE
   WRITE Color_Command = (FADE_TO_COLOR, (C2: any valid color supported by the IUT other than C1), (FT2: a different fade time longer than FT1))
   VERIFY In_Progress = FADE_ACTIVE
   WAIT (FT1 milliseconds)
   VERIFY Tracking_Value <> C2
   WAIT (FT2 - FT1 milliseconds)
   VERIFY Tracking_Value = C2
   VERIFY In_Progress = IDLE

7.3.2.X67.32 Writing Default_Fade_Time Positive Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT can be configured with a Default_Fade_Time at the bounds of the allowable value range using BACnet services.

Test Concept: Default_Fade_Time is written with a value equal to 100 milliseconds. Then Default_Fade_Time is written with a value equal to 86400000 milliseconds.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1. WRITE Default_Fade_Time = 100
2. VERIFY Default_Fade_Time = 100
3. WRITE Default_Fade_Time = 86400000
4. VERIFY Default_Fade_Time = 86400000

BTL-23.3 ca-3: Add Color-Reference Properties [BTLWG-1262]

Overview:

This WI adds testing for the color-reference properties of LO and BLO objects.

Changes:

Checklist Changes

[Modify the Lighting Output and Binary Lighting Output Object sections]

Lighting Output Object		
	R	Base Requirements
	R	Supports command prioritization
	R	Supports all BACnetLightingOperations
	S	Supports configurable Out_Of_Service property
	O	Supports blink-warn
	O	Supports Transition property
	O	Supports Feedback_Value property
	O	Supports Min_Actual_Value and Max_Actual_Value properties
	O	Supports the value source mechanism.
	O[1,2 ~~1,4~~]	Supports Color_Reference property
	O[1,~~2~~]	~~Supports Color_Override property~~ *Supports color override*
	O[1,2]	Supports the Trim_Fade_Time property
	O[3]	Supports intrinsic reporting
[1] Protocol_Revision 24 or higher must be claimed [2] Contact BTL for interim tests for this functionality [3] Protocol_Revision 21 or higher must be claimed *[4] Must be claimed if the IUT supports color override*		
Binary Lighting Output Object		
	R	Base Requirements
	R	Supports command prioritization
	S	Supports configurable Out_Of_Service property
	O	Supports blink-warn
	O	Supports writable Polarity property
	O	Supports strike count tracking
	O	Supports elapsed active time tracking
	O	Supports the value source mechanism.
	O[1,2]	Supports Color_Reference property
	O[1]	~~Supports Color_Override property~~ *Supports color override*
[1] Protocol_Revision 24 or higher must be claimed *[2] Must be claimed if the IUT supports color override*		

Test Plan Changes

[Replace sections Lighting Output Objects sections 3.54.10, 3.54.11 and Binary Lighting Output Objects sections 3.55.9, 3.55.10 with the below]

3.54.10 Supports Color_Reference Property

The IUT supports the Color_Reference property.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify the IUT claims support for either the Color or Color Temperature object.

	Testing Hints	

3.54.11 Supports Color Override

The IUT supports overriding the companion color.

BTL - 7.3.1.X54.1 - Color Override Test		
	Test Conditionality	Must be executed.
	Test Directives	Execute this test using the Lighting Output object as O1.
	Testing Hints	

3.55.9 Supports Color_Reference Property

The IUT supports the Color_Reference property.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify the IUT claims support for either the Color or Color Temperature object.
	Testing Hints	

3.55.10 Supports Color Override

The IUT supports overriding the companion color.

BTL - 7.3.1.X54.1 - Color Override Test		
	Test Conditionality	Must be executed.
	Test Directives	Execute this test using the Binary Lighting Output object as O1.
	Testing Hints	

Specified Test Changes

[Add new tests into BTL Specified Tests]

7.3.1.X54.1 Color Override Test
Reason for Change: No test exists for this functionality.

Purpose: This test ensures that a color override will follow the existing transition or no transition, but to the new commanded color or color temperature.

Test Concept: A companion Color or Color Temperature object (COLOR1) is referenced by the Color_Reference property of the lighting output object (O1). A different Color or Color Temperature object (COLOR2) is referenced by the Override_Color_Reference property of O1. No transitions or fades are in progress. Color override is enabled and the output of O1 is verified to match the color referenced by COLOR2. Color override is disabled and the output of O1 is verified to match the color referenced by COLOR1. A Color_Command is written to COLOR1 which would cause a transition to begin. During the transition period, color override is enabled and the output is verified to go to the color referenced by COLOR2 immediately. The override is again disabled and TD verifies that the color returns to the color that would have been in effect had the override not occurred. One more override is enabled and a Color_Command is written to COLOR2 which would cause another transition to begin. During this transition color override is disabled and the TD verifies that the output matches COLOR1's Present_Value.

Configuration Requirements: COLOR1 is referenced by O1's Color_Reference property. COLOR2 is referenced by O1's Override_Color_Reference property. Color_Override is False. No fades or transitions are taking place at the start of the test. COLOR1 and COLOR 2 exist in the IUT, are of the same object type, and have different Present_Values at the start of the test.

Notes to tester: Select Present_Values for COLOR1 and COLOR2 which are measurably different from each other, and color commands which result in measurable changes, to facilitate a successful test run. The CHECK steps will need to be defined by the vendor if a physical output device is not provided.

Test Steps:

1. READ C1 = (COLOR1), Present_Value
2. READ C2 = (COLOR2), Present_Value
3. WRITE (O1), Color_Override = TRUE
4. CHECK (that the color output of O1 is represented by C2)
5. WRITE (O1), Color_Override = FALSE
6. CHECK (that the color output of O1 is represented by C1)
7. WRITE (COLOR1), Color_Command = (any valid operation for the color object with a specified fade time T1 which will still be in effect after step 10)
8. WRITE (O1), Color_Override = TRUE
9. CHECK (that the color output of O1 is represented by C2 and is not doing any fading)
10. WRITE (O1), Color_Override = FALSE
11. CHECK (that the color output of O1 is now fading in the direction of the last color command written to COLOR1)
12. WAIT (T1 seconds) -- Wait for the transition to finish
13. CHECK (the color output from O1 is represented by the last color command written to COLOR1)
14. WRITE (O1), Color_Override = TRUE
15. WRITE (COLOR2), Color_Command = (any valid operation for the color object with a specified fade time T2 which will still be in effect after step xyz)
16. CHECK (that the color output of O1 is fading to a color that represents the last color command written to COLOR2)
17. BEFORE (T2 has elapsed)
    WRITE (O1), Color_Override = FALSE
18. CHECK (that the color output of O1 is represented by the last color command written to COLOR1 and is not doing any fading)

BTL-23.3 ca-4: Add New Color Temperature Object [BTLWG-1261]

Overview:

Add new Color Temperature Object Type.

Changes:

Checklist Changes

[Replace the Color Temperature Object section]

Color Temperature Object		
	R	Base Requirements
	O	Supports Transition property
	O	Supports the value source mechanism

[Add color temperature objects to Data Management - Object Creation and Deletion A and B sections]

Device Management - Object Creation and Deletion - A		
	…	…
	C[2]	Can create and delete Color Temperature objects
…		

Device Management - Object Creation and Deletion - B		
	…	…
	C[1,6]	Supports object creation and deletion of the Color Temperature object
…		
[6] Protocol_Revision 24 or higher must be claimed.		

Test Plan Changes

[Replace contents of 3.66 Color Temperature Object with the below]

3.66.1 Base Requirements

Base requirements must be met by any IUT that can contain Color Temperature objects.

Verify Checklist		
	Test Conditionality	Must be executed.
	Test Directives	Verify the IUT claims support for DS-WP-B
	Testing Hints	
BTL - 7.2.X1 - Numeric Bounds Test		
	Test Conditionality	Must be executed if the IUT does not support Min_Pres_Value and Max_Pres_Value.
	Test Directives	Execute this test on Present_Value, with an upper bound of 30000 and lower bound of 1000.
	Testing Hints	
BTL - 7.3.2.X68.1 - Color Temperature Object Present_Value Startup Test		
	Test Conditionality	Must be executed.
	Test Directives	If Default_Color_Temperature is configurable, repeat this test with Default_Color_Temperature = 0 and a valid non-zero value.
	Testing Hints	
BTL - 7.3.2.X68.2 - Transition NONE Test		
	Test Conditionality	This test shall be skipped if the Transition property is supported and cannot be set to NONE.

	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.3 - Color Temperature Object Present_Value Clamping Test		
	Test Conditionality	Must be executed if the IUT supports Min_Pres_Value and Max_Pres_Value.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.4 - Color Temperature Object Color_Command Out Of Range Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.5 - Invalid Color_Command Operations Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.6 - Valid Color Command Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.8 - Interrupting a Fade In Progress		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.9 - Interrupting a Ramp In Progress		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.11 - Color_Command Optional Parameter Out Of Range Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.22 - Default_Fade_Time Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.23 - Default_Ramp_Rate Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.24 - Default_Step_Size Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.35 - Configuring Default_Step_Increment Within Allowable Range		
	Test Conditionality	Must be executed if Default_Step_Increment is configurable.
	Test Directives	
	Testing Hints	

3.66.2 Supports Transition Property

The IUT supports the Transition property in Color Temperature objects.

BTL - 7.3.2.X68.25 - Transition FADE Test		
	Test Conditionality	This test shall be skipped if the Transition property is supported and cannot be set to FADE.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X68.26 - Transition RAMP Test		

	Test Conditionality	This test shall be skipped if the Transition property is supported and cannot be set to RAMP.
	Test Directives	
	Testing Hints	

3.66.3 Supports the Value Source Mechanism

The IUT supports the Value Source Mechanism in color temperature objects.

135.1-2023 - 7.3.1.28.2 - Non-commandable Value_Source Property Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	

[Add section for DM-OCD-A and DM-OCD-B]

8.21.68 Can Create and Delete Color Temperature Objects

The IUT can create and delete Color Temperature objects. The IUT shall not restrict the instance number which can be used to create the Color object.

135.1-2023 - 8.16.1 - Creating Objects by Specifying the Object Identifier with no Initial Values		
	Test Conditionality	Must be tested on the Color Temperature Object
	Test Directives	
	Testing Hints	
135.1-2023 - 8.17 - DeleteObject Service Initiation Tests		
	Test Conditionality	Must be tested on the Color Temperature Object
	Test Directives	
	Testing Hints	

8.22.68 Supports Object Creation and Deletion of the Color Temperature Object

The Color Temperature object can be created and deleted within the IUT. The Color Temperature object that is created must be the object that can be deleted using the delete service.

135.1-2023 - 9.16.1.1 - Creating Objects by Specifying the Object Type with No Initial Values		
	Test Conditionality	**Must be executed.**
	Test Directives	**Execute using the Color Temperature Object.**
	Testing Hints	
135.1-2023 - 9.16.1.2 - Creating Objects by Specifying the Object Identifier with No Initial Values		
	Test Conditionality	**Must be executed.**
	Test Directives	**Execute using the Color Temperature Object.**
	Testing Hints	
135.1-2023 - 9.17.1.1 - Successful Deletion of an Object		
	Test Conditionality	**Must be executed.**
	Test Directives	**Execute using the Color Temperature Object.**
	Testing Hints	

Specified Test Changes

[Add a new test section into BTL Specified Tests]

7.3.2.X68 Color Temperature Object Tests

7.3.2.X68.1 Color Temperature Object Present_Value Startup Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the Color Temperature object's Present_Value goes to either the last Present_Value or a default color temperature on startup, depending on the color in the Default_Color_Temperature property.

Test Concept: The IUT is restarted and Present_Value is verified to go either to Default_Color_Temperature or the previous color temperature in effect prior to restart if Default_Color_Temperature is 0. The color temperature output in Tracking_Value is verified to go to either Present_Value or the previous color temperature before the restart.

Configuration Requirements: The IUT is not performing any color temperature commands or fades at the beginning of this test. The starting Present_Value, PV1, shall be set to something other than the Default_Color_Temperature,.

Test Steps:

1. VERIFY In_Progress = IDLE
2. READ PV1 = Present_Value
3. READ DCT = Default_Color_Temperature
4. CHECK (PV1 <> DCT)
5. MAKE (the IUT restart)
6. WAIT (for the IUT to restart)
7. IF (DCT = 0) THEN {
8. IF (Present_Value is preserved over a power cycle) THEN {
9. VERIFY Present_Value = PV1
10. VERIFY In_Progress = IDLE
}
11. ELSE {
12. VERIFY In_Progress = NOT_CONTROLLED
}
}
13. ELSE {
14. VERIFY Present_Value = DCT
15. VERIFY Tracking_Value = DCT
16. VERIFY In_Progress = IDLE
}

7.3.2.X68.2 Transition NONE Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when Transition is NONE or not supported, writing to the Present_Value is set to the target color temperature immediately.

Test Concept: Transition is verified as NONE or not supported. Tracking_Value is read. A different value is written to Present_Value and Tracking_Value is read back as equal to Present_Value.

Configuration Requirements: The IUT is not performing any color temperature commands or fades at the start of this test.

Test Steps:

1. IF (Transition property is supported) THEN
2. VERIFY Transition = NONE
3. READ TV = Tracking_Value
4. WRITE Present_Value = (C1: any valid color temperature supported by the IUT, other than TV)
5. VERIFY Tracking_Value = C1
6. VERIFY In_Progress = IDLE

7.3.2.X68.3 Color Temperature Object Present_Value Clamping Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a color temperature which is within the standard range for Present_Value but out of range of Min_Pres_Value and Max_Pres_Value is written.

Test Concept: A Color Temperature object's (O1) Present_Value is read, then written to using values T1 and T2, where T1 is a value between 1000 Kelvin and Min_Pres_Value and T2 is a value between Max_Pres_Value and 30000 Kelvin.

Configuration Requirements: The color temperature object should not be executing any fades.

Notes to Tester: Configuring Transition to NONE, or minimizing Default_Fade_Time and maximizing Default_Ramp_Rate will assist in reducing the time it takes to execute this test.

Test Steps:

1. READ PV1 = Present_Value
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = Present_Value,
   'Property Value' = 999
3. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE
4. VERIFY Present_Value = PV1
5. TRANSMIT WriteProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = Present_Value,
   'Property Value' = 30001
6. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE
7. VERIFY Present_Value = PV1
8. IF (the IUT supports Min_Pres_Value and Max_Pres_Value) THEN {
9. IF (Min_Pres_Value > 1000) THEN {
10. WRITE Present_Value = T1
11. VERIFY Present_Value = Min_Pres_Value
12. IF (Transition is present and set to FADE) THEN {
13. WAIT (Default_Fade_Time milliseconds)
    }
14. IF (Transition is present and set to RAMP) THEN {
15. WAIT (((PV1 - Min_Pres_Value) / Default_Ramp_Rate) seconds)
    }
16. VERIFY Tracking_Value = Min_Pres_Value
    }
17. IF (Max_Pres_Value < 30000) THEN {
18. READ PV1 = Present_Value
19. WRITE Present_Value = T2
20. VERIFY Present_Value = Max_Pres_Value
21. IF (Transition is present and set to FADE) THEN {
22. WAIT (Default_Fade_Time milliseconds)
    }
23. IF (Transition is present and set to RAMP) THEN {
24. WAIT (((Max_Pres_Value - PV1) / Default_Ramp_Rate) seconds)
    }
25. VERIFY Tracking_Value = Max_Pres_Value
    }
}

7.3.2.X68.4 Color Temperature Object Color_Command Out Of Range Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when a target color temperature which is out of range is written to its Color_Command, depending on the color temperature written.

Test Concept: A Color Temperature object's (O1) Present_Value is read, then Color_Command is written to with a target value outside the allowed range of 1000 to 30000. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then, if the IUT supports Min_Pres_Value and Max_Pres_Value, a Color_Command with a target value which is within the allowed range but outside of the range supported by the IUT, is written to the IUT. An error is returned and Present_Value is clamped to the Min_Pres_Value or Max_Pres_Value.

Configuration Requirements: The IUT is not performing any color commands or fades.

Test Steps:

```
1.  REPEAT X = (each valid Color_Command operation) DO {
2.      READ PV1 = Present_Value
3.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, additional parameters which would result in a color temperature below 1000K)
4.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
5.      READ PV1 = Present_Value
6.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X, additional parameters which would result in a color temperature above 30000K)
7.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
8.      VERIFY Present_Value = PV1
9.      IF (the IUT supports Min_Pres_Value and Max_Pres_Value) THEN {
10.         IF (Min_Pres_Value > 1000) THEN {
11.             WRITE Color_Command = (X, additional parameters which would result in a color temperature between
1000 and Min_Pres_Value)
12.             VERIFY Present_Value = Min_Pres_Value
13.             WHILE (In_Progress <> IDLE) { }  -- Do nothing
14.             VERIFY Tracking_Value = Min_Pres_Value
            }
15.         IF (Max_Pres_Value < 30000) THEN {
16.             WRITE Color_Command = (X, additional parameters which would result in a color temperature between
Max_Pres_Value and 30000)
17.             VERIFY Present_Value = Max_Pres_Value
18.             WHILE (In_Progress <> IDLE) { -- Do nothing }
19.             VERIFY Tracking_Value = Max_Pres_Value
            }
        }
    }
```

7.3.2.X68.5 Invalid Color_Command Operations Test

Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct error when invalid color commands are written to the Color_Command property of the Color Temperature object.

Test Concept: A Color Temperature object's (O1) Present_Value and Color_Command are read, then Color_Command is written to with each unsupported CCT color command. An error is received each time, with Error Class = PROPERTY and Error Code = VALUE_OUT_OF_RANGE. When the error is received, TD verifies the Present_Value has not changed.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

```
1   READ CC = Color_Command
2.  READ PV = Present_Value
3.  REPEAT X = (each invalid Color_Command operation, including NONE and a value not defined) DO {
4.      TRANSMIT WriteProperty-Request,
            'Object Identifier' = O1,
            'Property Identifier' = Color_Command,
            'Property Value' = (X)
5.      RECEIVE BACnet-Error-PDU,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE
6.      VERIFY Present_Value = PV
7.      VERIFY Color_Command = CC
    }
```

7.3.2.X68.6 Valid Color Command Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will accept all valid color commands when sent with minimum and maximum parameters and without optional parameters present.

Test Concept: Each valid Color Command Operation is written with valid target color temperatures, exercising the optional fields of each operation. TD also verifies that when writing a Color Command without the optional field, that the default parameter is used. TD verifies the fade is completed once enough time has elapsed. This process is repeated for all remaining valid Color Commands.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1. REPEAT X = (each valid Color_Command operation) DO {
2. WRITE Color_Command = (X, (C1: any valid target color temperature supported by the IUT, or absent if X does not support it), MIN: the minimum allowable value for this parameter for X)
3. WAIT (until the color command has finished)
4. VERIFY In_Progress = IDLE
5. VERIFY Present_Value = (the color output determined by X and C1's presence)
6. WRITE Color_Command = (X, (C2: any valid target color temperature supported by the IUT other than C1, or absent if X does not support it), MAX: the maximum allowable value for this parameter for X)
7. VERIFY In_Progress = (an appropriate state for X)
8. VERIFY Present_Value = (the color output determined by X and C2's presence)

-- Write Color Command without the optional parameter, interrupting the last one to verify use of the default property corresponding to the optional parameter in the Color Command

9. READ T1 = (the 'default' value corresponding to the optional parameter in X)
10. WRITE Color_Command = (X, (C1, or absent if X does not support it))
11. VERIFY Present_Value = (the color output determined by X and C1's presence)
12. IF (the color command will finish within a reasonable timeframe based on X and T1) THEN {
13. WAIT (for the fade to finish based on T1)
14. VERIFY Tracking_Value = (the color output determined by X and C1's presence)
15. VERIFY In_Progress = IDLE
    }
16. ELSE {
17. WRITE Color_Command = STOP
    }
    }

7.3.2.X68.8 Interrupting a Fade In Progress
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a fade in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to a Color Temperature object's (O1) which starts a fade to a color temperature (C2) with a specified fade-time, then it interrupts the fade by writing to the Present_Value property. The fade should immediately stop and go to the color temperature written in Present_Value, depending on the presence, and value of, the Transition property. Then for each valid color command, TD writes a color command to begin a fade to color temperature. Before the operation completes, TD interrupts the fade with a different color command. TD verifies that Color_Command matches the command that was written and that Present_Value and In_Progress have appropriate values.

Configuration Requirements: The IUT should not have a fade or ramp in progress at the beginning of this test. If Transition is configurable, it shall not be configured to NONE at the start of this test.

Test Steps:

1. READ C1 = Present_Value
2. VERIFY In_Progress = IDLE
3. WRITE Color_Command = (FADE_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), 86400000)
4. VERIFY In_Progress = FADE_ACTIVE

5. VERIFY Present_Value = C2
-- Interrupt the color command by writing to Present_Value
6. WRITE Present_Value = C1
7. REPEAT X = (each valid Color_Command operation including FADE_TO_CCT and STOP) DO {
8. WRITE Color_Command = (FADE_TO_CCT, C2, 86400000)
9. WRITE Color_Command = (X, (C1, or absent if X does not support a target color temperature), (the maximum value for this parameter, or absent if X does not support a fade-time or ramp-rate))
10. VERIFY Present_Value = (a value appropriate to X)
11. VERIFY In_Progress = (a value appropriate to X)
12. VERIFY Color_Command = (X)
}

7.3.2.X68.9 Interrupting a Ramp In Progress
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT will stop a ramp in progress when Present_Value is written to, when a new color command is written, or when STOP is written to the Color_Command property.

Test Concept: TD writes a color command to a Color Temperature object's (O1) which starts a ramp to a color temperature C2 with a specified ramp-time, then it interrupts the ramp by writing to the Present_Value property. The ramp should immediately stop and go to the color temperature written in Present_Value, depending on the presence, and value of, the Transition property. Then for each valid color command, TD writes a color command operation to begin a ramp to color temperature. Before the operation completes, TD interrupts the ramp with a different color command. TD verifies that Color_Command matches the command that was written and that Present_Value and In_Progress have appropriate values.

Configuration Requirements: The IUT should not have a ramp in progress at the beginning of this test. If Transition is configurable, it shall not be configured to NONE at the start of this test.

Test Steps:

1. READ C1 = Present_Value
2. VERIFY In_Progress = IDLE
3. WRITE Color_Command = (RAMP_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), 1)
4. VERIFY In_Progress = RAMP_ACTIVE
5. VERIFY Present_Value = C2
-- Interrupt the color command by writing to Present_Value
6. WRITE Present_Value = C1
7. REPEAT X = (each valid Color_Command operation including RAMP_TO_CCT and STOP) DO {
8. WRITE Color_Command = (RAMP_TO_CCT, C2, 1)
9. WRITE Color_Command = (X, (C2, or absent if X does not support a target color temperature), (the maximum value for this parameter, or absent if X does not support a fade-time or ramp-rate)
10. VERIFY Present_Value = (a value appropriate to X)
11. VERIFY In_Progress = (a value appropriate to X)
12. VERIFY Color_Command = (X)
}

7.3.2.X68.11 Color_Command Optional Parameter Out Of Range Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT responds with the correct behavior when Color Command is written which contains an optional parameter whose value is outside the allowed range.

Test Concept: Present_Value of a Color Temperature object (O1) is read, then for each valid color command operation, a color command is written with the optional parameter smaller than the minimum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. Then another color command is written with the optional parameter larger than the maximum allowed. An error is received with Error Class = PROPERTY, and Error Code = VALUE_OUT_OF_RANGE. After each write, TD verifies the Present_Value is unchanged.

Configuration Requirements: The IUT is not performing any color commands. If Transition is present and configurable, it shall be configured to NONE. If Min_Pres_Value and Max_Pres_Value are configurable, they shall not be configured to 1000 and 30000 respectively.

Test Steps:

1. READ Cx = Present_Value
2. REPEAT X = (each valid Color Command operation other than STOP) DO {
3. TRANSMIT WriteProperty-Request,
'Object Identifier' = O1,
'Property Identifier' = Color_Command,
'Property Value' = (X, (C1: any valid target color temperature other than Cx or absent), (a value smaller than the minimum allowed for this parameter))
4. RECEIVE BACnet-Error-PDU,
'Error Class' = PROPERTY,
'Error Code' = VALUE_OUT_OF_RANGE
5. VERIFY Present_Value = Cx
6. TRANSMIT WriteProperty-Request,
'Object Identifier' = O1,
'Property Identifier' = Color_Command,
'Property Value' = (X, (C1 or absent), (a value larger than the maximum allowed for this parameter))
7. RECEIVE BACnet-Error-PDU,
'Error Class' = PROPERTY,
'Error Code' = VALUE_OUT_OF_RANGE
8. VERIFY Present_Value = Cx
}
9. IF (the IUT supports the Min_Pres_Value and Max_Pres_Value properties) THEN {
10. IF (Min_Pres_Value > 1000) THEN {
11. WRITE Color_Command = (STEP_DOWN_CCT, (S1: a value such that 1000 <= Cx - S1 < Min_Pres_Value))
12. VERIFY Present_Value = Min_Pres_Value
}
13. IF (Max_Pres_Value < 30000) THEN {
14. WRITE Color_Command = (STEP_UP_CCT, (S2: a value such that Max_Pres_Value < Present_Value + S2 <= 30000))
15. VERIFY Present_Value = Max_Pres_Value
}
}

7.3.2.X68.22 Default_Fade_Time Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified fade-time will use that fade-time instead of the Default_Fade_Time and when fade-time is not specified, Default_Fade_Time is used

Test Concept: TD writes a Color_Command with a fade-time, that is different from Default_Fade_Time and verifies the color temperature fade did not end after Default_Fade_Time milliseconds. A second color command is written without the fade-time parameter and TD verifies that the fade ends after Default_Fade_Time milliseconds, if Default_Fade_Time is a reasonable value.

Configuration Requirements: There are no configuration requirements for this test.

Test Steps:

1. READ C1 = Present_Value
2. WRITE Color_Command = (FADE_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), (Ft: a valid fade-time that is different than Default_Fade_Time and long enough for the next step to be executed))
3. IF (Ft > Default_Fade_Time) THEN {
4. WAIT (Default_Fade_Time milliseconds)
5. VERIFY In_Progress = FADE_ACTIVE
6. WAIT (Ft milliseconds - Default_Fade_Time)
7. VERIFY In_Progress = IDLE
}
8. ELSE { -- (Ft < Default_Fade_Time)

9. WAIT (Ft milliseconds)
10 VERIFY In_Progress = IDLE
}
11. WRITE Color_Command = (FADE_TO_CCT, C1)
12. IF (Default_Fade_Time is not excessively long) THEN {
13. WAIT (Default_Fade_Time milliseconds)
14. VERIFY Tracking_Value = C2
15. VERIFY In_Progress = IDLE
}

7.3.2.X68.23 Default_Ramp_Rate Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified ramp-rate will use that ramp rate instead of the Default_Ramp_Rate and when ramp-rate is not specified, Default_Ramp_Rate is used.

Test Concept: TD writes a Color_Command with a ramp-rate, that is different from Default_Ramp_Rate and verifies the color temperature ramp did not end after Default_Ramp_Rate. A second color command is written without the ramp-rate parameter and TD verifies that the ramp ends at the appropriate time.

Configuration Requirements: Default_Ramp_Time must not be set to 1. Rr shall be a valid ramp-rate that is different than Default_Ramp_Rate and large enough to allow the test to be executed. T1 shall be the ((absolute value of C1-C2/Default_Ramp_Rate). T2 shall be the ((absolute value of C1-C2/Rr).

Test Steps:

1. READ C1 = Present_Value
2. WRITE Color_Command = (RAMP_TO_CCT, (C2: any valid color temperature supported by the IUT other than C1), Rr)
3. IF (Rr > Default_Ramp_Rate) THEN {
4. WAIT (T1 seconds)
5. VERIFY In_Progress = RAMP_ACTIVE
6. WAIT (T1 - T2 seconds)
7. VERIFY In_Progress = IDLE
}
8. ELSE { -- (Rr < Default_Ramp_Rate)
9. WAIT (T2 seconds)
10. VERIFY In_Progress = IDLE\
}
11. WRITE Color_Command = (RAMP_TO_CCT, C1)
12. IF (T1 is not excessively long) THEN {
13 WAIT (T1)
14 VERIFY Tracking_Value = C1
15 VERIFY In_Progress = IDLE
}

7.3.2.X68.24 Default_Step_Size Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that writes to Color_Command with a specified step-size will use that step size instead of the Default_Step_Size.

Test Concept: TD writes a Color_Command with a step-size, that is different from Default_Step_Size and verifies the color temperature did not change by Default_Step_Size.

Configuration Requirements: Present_Value must not be at the maximum allowable value for the Color Temperature object at the start of the test.

Test Steps:

1. READ C1 = Present_Value
2 WRITE Color_Command = (STEP_UP_CCT, (S1: any valid step size supported by the IUT other than Default_Step_Size, and will not cause clamping of Present_Value))
3. VERIFY In_Progress = IDLE

4. VERIFY Tracking_Value = C1 + S1
5. VERIFY Present_Value = C1 + S1
6 WRITE Color_Command = (STEP_DOWN_CCT, S1)
7. VERIFY In_Progress = IDLE
8. VERIFY Tracking_Value = C1
9. VERIFY Present_Value = C1

7.3.2.X68.25 Transition FADE Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property set to FADE, writes to Present_Value will use the Default_Fade_Time.

Test Concept: TD writes a Present_Value, verifies the color temperature fade is not continuing after Default_Fade_Time milliseconds.

Configuration Requirements: Transition shall be configured to FADE at the beginning of this test.

Test Steps:

1. VERIFY Transition = FADE
2. READ C1 = Present_Value
3. WRITE Present_Value = (C2: any valid color temperature supported by the IUT other than C1)
4. IF (TD can read In_Progress before Default_Fade_Time elapses) THEN {
5. VERIFY In_Progress = FADE_ACTIVE
   }
6. WAIT (Default_Fade_Time milliseconds)
7. VERIFY In_Progress = IDLE
8. VERIFY Tracking_Value = C2

7.3.2.X68.26 Transition RAMP Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that when the IUT supports the Transition property set to RAMP, writes to Present_Value will use the Default_Ramp_Rate.

Test Concept: TD writes a Present_Value, verifies the color temperature ramp is not continuing after Default_Ramp_Rate milliseconds.

Configuration Requirements: Transition shall be configured to RAMP at the beginning of this test.

Test Steps:

1. VERIFY Transition = RAMP
2. READ C1 = Present_Value
3. WRITE Present_Value = (C2: any valid color temperature supported by the IUT other than C1)
4. IF(TD can read In_Progress before the transition completes) THEN {
5. VERIFY In_Progress = RAMP_ACTIVE
   }
6. WAIT ((absolute value of C1-C2)/Rr seconds)
7. VERIFY In_Progress = IDLE
8. VERIFY Tracking_Value = C2

7.3.2.X68.35 Configuring Default_Step_Increment Within Allowable Range
Reason for Change: No test exists for this functionality.

Purpose: This test verifies that the IUT supports a configurable Default_Step_Increment.

Test Concept: The IUT is configured with a different Default_Step_Increment, S1. A color command with step-increment S2 is written and Present_Value is verified to only be equal to the written color temperature after that ramp is completed.

Configuration Requirements: Present_Value should not be set to the minimum or maximum color temperature supported by the IUT at the start of the test.

Test Steps:

1. READ S0 = Default_Step_Increment
2. MAKE (configure the IUT such that Default_Step_Increment = S1: a different step increment than S0)
3. VERIFY S1 = Default_Step_Increment
4. READ C1 = Present_Value
5. WRITE Color_Command = STEP_UP_CCT
6. VERIFY Present_Value = (C1 + S1)
7. WRITE Color_Command = STEP_DOWN_CCT
8. VERIFY Present_Value = C1

BTL-23.3 ca-5: Add High_End_Trim and Low_End_Trim Testing [BTLWG-1263]

Overview:

The High_End_Trim and Low_End_Trim properties for Lighting Output were added in 135-2020, Addendum ca-5. Changes need to be made and tests added to support these new properties.

Changes:

Checklist Changes

[Modify the Lighting Output Object section. Note The Color_Reference, Color_Override, Trim_Fade_Time, and High_End_Trim or Low_End_Trim properties required PR24. The Intrinsic Reporting requires PR21.]

Lighting Output Object		
	R	Base Requirements
	R	Supports command prioritization
	R	Supports all BACnetLightingOperations
	S	Supports writable Out_Of_Service property
	O	Supports blink-warn
	O	Supports Transition property
	O	Supports Feedback_Value property
	O	Supports Min_Actual_Value and Max_Actual_Value properties
	O	Supports the value source mechanism.
	O[1] ~~[2]~~	Supports Color_Reference property
	O[1] ~~[2]~~	Supports Color_Override property
	O[1] ~~[2]~~	Supports the Trim_Fade_Time property
	O ~~[2]~~	Supports intrinsic reporting
	O[1] ~~[2]~~	Supports High_End_Trim or Low_End_Trim properties
[1] Protocol_Revision 24 or higher must be claimed ~~[2] Contact BTL for interim tests for this functionality~~ [2] Protocol_Revision 21 or higher must be claimed		

Test Plan Changes

3.54 Lighting Output Object

[Add new section to Lighting Output Object]

3.54.14 Supports High_End_Trim or Low_End_Trim Properties

IUT supports the High_End_Trim or Low_End_Trim properties.

BTL - 7.3.2.39.X1 - Tracking_Value Clamping Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.3.2.39.X2 - Priority 1 and 2 Clamping Test		
	Test Conditionality	Must be executed.
	Test Directives	
	Testing Hints	
BTL - 7.2.X1 - Numeric Bounds Test		
	Test Conditionality	Must be executed if the High_End_Trim is present and writable and/or the Low_End_Trim is present and writable, otherwise this test shall be skipped.
	Test Directives	Execute this test on High_End_Trim, if present and writable, with an upper bound of 100 and an appropriate lower bound.

		Execute this test on Low_End_Trim, if present and writable, with a lower bound of 1 and an appropriate upper bound.
	Testing Hints	

Specified Test Changes

[Add 3 new tests to BTL Specified Tests]

7.3.2.39.X1 Tracking_Value Clamping Test
Reason for Change: No test exists for this functionality.

Purpose: To ensure that Tracking_Value is properly clamped when Present_Value is written to a value above High_End_Trim or below Low_End_Trim and In_Progress is equal TRIM_ACTIVE.

Test Concept: Throughout this test, the Present_Value shall be written at a priority between 3 and 16 and is the active priority. If High_End_Trim is present, write the Present_Value to a value greater than High_End_Trim. Verify In_Progress is equal to TRIM_ACTIVE, Track_Value is equal to the High_End_Trim, and Present_Value is equal to the value written. If Low_End_Trim is present, write the Present_Value to a value less than Low_End_Trim. Verify In_Progress is equal to TRIM_ACTIVE, Track_Value is equal to the Low_End_Trim, and Present_Value is equal to the value written.

Test Configuration: The Lighting Output object, O1, shall be configured such that In_Progress is IDLE and no processes are writing to the Present_Value. The High_End_Trim, if present, shall be less than 99. The Low_End_Trim, if present, shall be greater than 2.

Test Steps:

1. VERIFY In_Progress = IDLE
2. IF (High_End_Trim is present) THEN {
3. READ HET = High_End_Trim
4. WRITE Present_Value = (PV, a valid value > HET)
5. WHILE (In_Progress <> TRIM_ACTIVE) {}
6. VERIFY Tracking_Value = HET
7. VERIFY Present_Value = PV
   }
8. IF (Low_End_Trim is present) THEN {
9. READ LET = Low_End_Trim
10. IF (In_Progress = TRIM_ACTIVE) THEN {
11. WRITE Present_Value = (PV, a valid value < HET and > LET)
12. WHILE (In_Progress <> IDLE) {}
    }
13. WRITE Present_Value = (PV, a valid value < LET)
14. WHILE (In_Progress <> TRIM_ACTIVE) {}
15. VERIFY Tracking_Value = LET
16. VERIFY Present_Value = PV
    }

7.3.2.39.X2 Priority 1 and 2 Clamping Test
Reason for Change: No test exists for this functionality.

Purpose: To verify that Tracking_Value is not clamped to High_End_Trim or Low_End_Trim when Present_Value is written at priorities 1 and 2.

Test Concept: Throughout this test, the Present_Value shall be written at a priority 1 and 2 and is the active priority. If High_End_Trim is present, write the Present_Value to a value greater than High_End_Trim. Verify In_Progress is equal to IDLE and Track_Value and Present_Value are equal to the value written. If Low_End_Trim is present, write the Present_Value to a value less than Low_End_Trim. Verify In_Progress is equal to IDLE and Track_Value and Present_Value are equal to the value written.

Test Configuration: The Lighting Output object, O1, shall be configured such that In_Progress is IDLE and no processes are writing to the Present_Value. The High_End_Trim, if present, shall be less than 99. The Low_End_Trim, if present, shall be greater than 2.

Test Steps:

1. VERIFY In_Progress = IDLE
2. IF (High_End_Trim is present) THEN {
3. READ HET = High_End_Trim
4. WRITE Present_Value, = (PV, a valid value > HET), PRIORITY = 1
5. WHILE (In_Progress <> IDLE) {}
6. VERIFY Tracking_Value = PV
7. VERIFY Present_Value = PV
   }
8. IF (Low_End_Trim is present) THEN {
9. READ LET = Low_End_Trim
10. WRITE Present_Value, = (NULL), PRIORITY = 1
11. WHILE (In_Progress <> IDLE) {}
12. WRITE Present_Value = (PV, a valid value < LET), PRIORITY = 2
13. WHILE (In_Progress <> IDLE) {}
14. VERIFY Tracking_Value = PV
15. VERIFY Present_Value = PV
    }

7.2.X1 Numeric Bounds Test
Reason for Change: No test exists for this functionality.

Purpose: This test validates that the upper and lower bounds of a numeric property can be written, and values outside of the range return the correct error class and code.

Test Concept: Property (P1) in object (O1) is successfully written using the upper bound value (UB). P1 is written with a value greater than UB and an error response is verified. P1 is successfully written using the lower bound value (LB). P1 is written with a value less than LB and an error response is verified.

Configuration Requirements: If conditionally writable, Property P1, shall be made writable.

Test Steps:

1. IF (UB supported) THEN {
2. WRITE (O1), P1 = UB,
3. VERIFY (O1), P1 = UB
4. TRANSMIT WriteProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Value' = (UB + 1)
5. RECEIVE BACnet-Error-PDU,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE
6. VERIFY (O1), P1 = UB
   }
7. IF (LB supported) THEN {
8. WRITE (O1), P1 = LB,
9. VERIFY (O1), P1 = LB
10. TRANSMIT WriteProperty-Request,
    'Object Identifier' = O1,
    'Property Identifier' = P1,
    'Property Value' = (LB - 1)
11. RECEIVE BACnet-Error-PDU,
    Error Class = PROPERTY,
    Error Code = VALUE_OUT_OF_RANGE
12. VERIFY (O1), P1 = LB
    }

BTL-23.3 ca-6: Add Trim_Fade_Time Property Testing [BTLWG-1570]

Overview:

Testing for the Trim_Fade_Time property, added in 135-2020-ca, is needed.

Changes:

Checklist Changes

[Modify the Lighting Output Object section]

Lighting Output Object		
	…	…
	O[1,2]	Supports Color_Reference property
	O[1,2]	Supports Color_Override property
	~~O[1,2]~~	~~Supports the Trim_Fade_Time property~~
	O[3]	Supports intrinsic reporting
	O[1,2]	Supports High_End_Trim or Low_End_Trim properties
[1] Protocol_Revision 24 or higher must be claimed [2] Contact BTL for interim tests for this functionality [3] Protocol_Revision 21 or higher must be claimed		

Test Plan Changes

[Remove section 3.54.12 Supports Trim_Fade_Time property entirely and renumber remaining test sections]

~~3.54.12 Supports Trim_Fade_Time Property~~

[Add a test to section 3.54.13 Supports High_End_Trim or Low_End_Trim Properties]

BTL - 7.3.2.39.X3 - Trim_Fade_Time Test		
	Test Conditionality	Must be executed if the High_End_Trim is present and writable and/or the Low_End_Trim is present and writable, otherwise this test shall be skipped.
	Test Directives	
	Testing Hints	

Specified Test Changes

[Add a new test into BTL Specified Tests]
7.3.2.39.X3 Trim_Fade_Time Test
Reason for Change: No test exists for this functionality.

Purpose: This test verifies the IUT's Lighting Output object will fade using Trim_Fade_Time when the High_End_Trim or Low_End_Trim are changed such that the current value of Present_Value is outside of the Operating Range.

Test Concept: The Present_Value is made to be within the Operating Range. If the High_End_Trim property is present, it is changed to be a lower value such that Present_Value is now outside the Operating range and it is verified that it takes Trim_Fade_Time milliseconds for the Tracking_Value to fade to the new High_End_Trim value. The same steps are repeated using Low_End_Trim, if present.

Configuration Requirements: The IUT shall not be performing any fades at the start of this test and Present_Value shall be within the Operating Range. Default_Fade_Time and Trim_Fade_Time shall be configured to different values, if possible.

Test Steps:

```
1.  VERIFY In_Progress = IDLE
2.  READ DFT = Default_Fade_Time
3.  READ TFT = Trim_Fade_Time
4.  IF (High_End_Trim is present) THEN {
5.      READ HET1 = High_End_Trim
6.      WRITE Present_Value = HET1
7.      WRITE High_End_Trim = (a new value HET2, such that HET2 < PV)
8.      IF (TFT <= DFT) THEN {
9.          WAIT (TFT milliseconds)
10.         VERIFY Tracking_Value = HET2
11.         VERIFY In_Progress = TRIM_ACTIVE
12.         VERIFY Present_Value = HET1
        }
13.     IF (TFT > DFT) THEN {
14.         WAIT (DFT milliseconds)
15.         VERIFY Tracking_Value <> HET2
16.         WAIT (TFT - DFT milliseconds)
17.         VERIFY Tracking_Value = HET2
18.         VERIFY In_Progress = TRIM_ACTIVE
19.         VERIFY Present_Value = HET1
        }
        -- Reset test setup in case of Low_End_Trim being present
20.     WRITE Present_Value = HET2
21.     VERIFY In_Progress = IDLE
    }
22. IF (Low_End_Trim is present) THEN {
23.     READ LET1 = Low_End_Trim
24.     WRITE Present_Value = LET1
25.     WRITE Low_End_Trim = (a new value LET2, such that LET2 > PV)
26.     IF (TFT <= DFT) THEN {
27.         WAIT (TFT milliseconds)
28.         VERIFY Tracking_Value = HET2
29.         VERIFY In_Progress = TRIM_ACTIVE
30.         VERIFY Present_Value = HET1
        }
31.     IF (TFT > DFT) THEN {
32.         WAIT (DFT milliseconds)
33.         VERIFY Tracking_Value <> HET2
34.         WAIT (TFT - DFT milliseconds)
35.         VERIFY Tracking_Value = HET2
36.         VERIFY In_Progress = TRIM_ACTIVE
37.         VERIFY Present_Value = HET1
        }
    }
```