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

FOREWORD

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

BTL-TP14.0h-1: Minimum on off Tests, pg 2. [wID0553]
BTL-TP14.0h-2: Epics Consistency Test Changes, pg 14. [wID0922]
BTL-TP14.0h-3: Active_COV_Subscriptions SubscribeCOV Test Changes, pg 16. [wID0923]
BTL-TP14.0h-4: Remove duplicate EPICS Consistency Testing, pg 19. [wID0939]
BTL-TP14.0h-5: ReadProperty Test Plan Conditionality Change, pg 24. [wID0959]

In the following document, language to be added to existing clauses within the BTL Test Package 14.0 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout.

In addition, changes to BTL Specified Tests might also contain a yellow highlight to indicate the changes made by this addendum.

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

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

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

BTL-TP14.0h-1: Minimum on off Tests

Overview:
Review of the test plan regarding Minimum-On-Time and Minimum-Off-Time properties revealed that some functionality was not being tested. The review also identified a few errors in existing tests.

Changes:

[In BTL Test Plan, Change sections 3.6.7 (Binary Output Object) and 3.7.6 (Binary Value Object)]

Supports Minimum_Off_Time

The object contains Minimum_Off_Time property.

135.1-~~2009~~2013 - 7.3.1.4 - Minimum_Off_Time		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-~~2009~~2013***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6*.1* - Override of Minimum Time		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.2 - Minimum Off Time – Writing at priorities numerically greater than 6		
	Test Method	Manual
	Configuration	As per **BTL Specified Tests**.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.4 – Minimum Off Time – Writing at priorities numerically lesser than 6		
	Test Method	Manual
	Configuration	As per **BTL Specified Tests**.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.6 – Minimum_Off_Time – Clock is not affected by additional write operations		
	Test Method	Manual
	Configuration	As per **BTL Specified Tests**.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.8 – Ensuring Minimum_Off_Time starts at transition to INACTIVE		

	Test Method	Manual
	Configuration	As per **BTL Specified Tests**.
	Test Conditionality	If Minimum_On_Time and Minimum_Off_Time properties are present, this test must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.10 – Ensuring Minimum Times Are Not A Effected By Time Changes		
	Test Method	Manual
	Configuration	As per **BTL Specified Tests**.
	Test Conditionality	If the property is present and the device can execute TimeSynchronization or UTCTimeSynchronization requests, this test must be executed
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Test Plan, Change sections 3.6.8 (Binary Output Object) and 3.7.7 (Binary Value Object)]

Supports Minimum_On_Time

The object contains Minimum_On_Time property.

135.1-~~2009~~2013 - 7.3.1.5 - Minimum_On_Time		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-~~2009~~2013***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.*1* - Override of Minimum Time		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.3 - Minimum On Time – Writing at priorities numerically greater than 6		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.5 - Minimum On Time – Writing at priorities numerically lesser than 6		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	

	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.7 - Minimum_On_Time – Clock is not affected by additional write operations		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present, it must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.9 - Ensuring Minimum_On_Time starts at transition to ACTIVE		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If Minimum_On_Time and Minimum_Off_Time properties are present, this test must be executed.
	Test Directives	
	Testing Hints	
	Notes & Results	
BTL - 7.3.1.6.10 – Ensuring Minimum Times Are Not A Effected By Time Changes		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the property is present and the device can execute TimeSynchronization or UTCTimeSynchronization requests, this test must be executed
	Test Directives	
	Testing Hints	
	Notes & Results	

[In BTL Specified Tests, add new section header for min on/off times and change test 7.3.1.6]

7.3.1.6 Minimum On/Off Time Tests

7.3.1.6.*1* Override of Minimum Time

Reason for Change: ~~The notes for the tester were incorrect. There is no SSPC proposal for this change~~. ***T□e test □□s □e-□□itte□t□ □e□ □□e t□e depe□de□ce □□t□e p□ese□ce □f t□e □ i□i□ □□ □□ff□Ti□φ□pe□t□.***

Dependencies: ReadProperty Service Execution Tests, 9.15; WriteProperty Service Execution Tests, 9.19.

BACnet Reference Clause: 19.

Purpose: To verify that higher priority commands override minimum on or off times. If neither minimum on time or minimum off time is supported this test shall be omitted. This test applies to Binary Output and ***c□□ □ □□d□□e*** Binary Value objects.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE and it is controlled at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off and/or minimum on time to have expired. The Present_Value is written to with a value of ACTIVE at priority 7. The value of slot 6 of the Priority_Array is monitored to verify that it contains the value ACTIVE. Before the minimum on time expires the Present_Value is written to with a value of INACTIVE and a priority numerically lower (higher priority) than 6. This overrides the minimum on time and immediately initiates the minimum off time algorithm.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically less than 7 have a value of NULL and no internal algorithms are issuing commands to this object at a priority numerically

lesser (higher priority) than the priority that is currently controlling Present_Value. *Minimum_On_Time must be configured with a large enough value to allow execution of all test steps before it expires.*

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
4. BEFORE Minimum_On_Time
   WRITE Present_Value = INACTIVE, PRIORITY = (any value numerically lower than 6 (higher priority))
5. VERIFY Present_Value = INACTIVE
~~6. VERIFY Priority_Array = INACTIVE, PRIORITY = 6~~
6. VERIFY Priority_Array <> ACTIVE, ARRAY_INDEX = 6

Notes to Tester: If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1*, 2, 3 and 6* ~~through 3~~ and ACTIVE in steps 4 ~~through 7~~*6 and 5*, *and by using the Minimum_Off_Time in Step 4*.

[In BTL Specified Tests – add new tests]

7.3.1.6.2 Minimum Off Time – Writing at priorities numerically greater than 6

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

Purpose: To verify that commands written at a lower priority than 6 will not affect Present_Value while Minimum_Off_Time is in effect.

Test Concept: The initial Present_Value of the object tested is set to ACTIVE and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than P9 (P9 > 7). The object has been in this state long enough for any minimum on time to have expired. The Present_Value of the object is set to INACTIVE at a priority P9. Before Minimum_Off_Time expires, Present_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority_Array is monitored to verify that it contains the appropriate values and Present_Value is monitored to verify that it does not change before Minimum_Off_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Active	Inactive	Inactive	Inactive	Inactive	Active
PA_Index = 6	Null	Inactive	Inactive	Inactive	Inactive	<>Inactive
PA_Index = P7	Null	Null	Null	**Active**	Active	Active
PA_Index = P9	Null	**Inactive**	**Null**	Null	**Active**	Active
Relinquish_Default	Active	Active	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum_Off_Time

End of Test

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array from P9 and higher (numerically lesser) have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_Off_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = P9

2. VERIFY Present_Value = INACTIVE
3. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P9
5. VERIFY Present_Value = INACTIVE
6. VERIFY Priority_ Array = INACTIVE, ARRAY_INDEX = 6

--…(Steps 4-6:Check that a NULL value at P9 does not affect ARRAY_INDEX = 6 or PV)

7. WRITE Present_Value = ACTIVE, PRIORITY = P7 (6 < P7 < P9)
8. VERIFY Present_Value = INACTIVE
9. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = P7
10. VERIFY Priority_ Array = INACTIVE, ARRAY_INDEX = 6

--…(Steps 7-10:Check that an ACTIVE value at P7 does not affect ARRAY_INDEX = 6 or PV)

11. WRITE Present_Value = ACTIVE, PRIORITY = P9
12. VERIFY Present_Value = INACTIVE
13. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = P9
14. VERIFY Priority_ Array = INACTIVE, ARRAY_INDEX = 6

--…(Steps 11-14:Check that an ACTIVE value at P9 does not affect ARRAY_INDEX = 6 or PV)

15. WAIT (Minimum_Off_Time + Internal Processing Fail Time)
16. VERIFY Present_Value = ACTIVE

7.3.1.6.3 Minimum On Time – Writing at priorities numerically greater than 6

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

Purpose: To verify that commands written at a lower priority than 6 will not affect Present_Value while Minimum_On_Time is in effect.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than P9 (P9 > 7). The object has been in this state long enough for any minimum off time to have expired. The Present_Value of the object tested is set to ACTIVE at a priority P9. Before Minimum_On_Time expires, Present_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority_Array is monitored to verify that it contains the appropriate values and Present_Value is monitored to verify that it does not change before Minimum_On_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Inactive	Active	Active	Active	Active	Inactive
PA_Index = 6	Null	Active	Active	Active	Active	<>Active
PA_Index = P7	Null	Null	Null	**Inactive**	Inactive	Inactive
PA_Index = P9	Null	**Active**	**Null**	Null	**Inactive**	Inactive
Relinquish_Default	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

End of Test

Configuration Requirements: The object to be tested shall be configured such that all slots from P9 and higher (numerically lesser) in the Priority_Array have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_On_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = P9
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P9

5. VERIFY Present_Value = ACTIVE
6. VERIFY Priority_ Array = ACTIVE, ARRAY_INDEX = 6

--…(Steps 4-6:Check that a NULL value at P9 does not affect ARRAY_INDEX = 6 or PV)

7. WRITE Present_Value = INACTIVE, PRIORITY = P7 (6 < P7 < P9)
8. VERIFY Present_Value = ACTIVE
9. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = P7
10. VERIFY Priority_ Array = ACTIVE, ARRAY_INDEX = 6

--…(Steps 7-10:Check that an INACTIVE value at P7 does not affect ARRAY_INDEX = 6 or PV)

11. WRITE Present_Value = INACTIVE, PRIORITY = P9
12. VERIFY Present_Value = ACTIVE
13. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = P9
14. VERIFY Priority_ Array = ACTIVE, ARRAY_INDEX = 6

--…(Steps 11-14:Check that an INACTIVE value at P9 does not affect ARRAY_INDEX = 6 or PV)

15. WAIT (Minimum_On_Time + Internal Processing Fail Time)
16. VERIFY Present_Value = INACTIVE

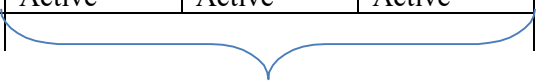
7.3.1.6.4 Minimum Off Time – Writing at priorities numerically lesser than 6

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

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum off time is in effect.

Test Concept: The initial Present_Value of the object tested is set to ACTIVE and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum on time to have expired. The Present_Value of the object tested is set to INACTIVE at a priority P5 (P5 < 6). Before Minimum_Off_Time expires, Present_Value is written with values of NULL and ACTIVE and the Present_Value and Priority_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Active	Inactive	Inactive	Active
PA_Index = P5	Null	**Inactive**	**Null**	**Active**
PA_Index = 6	Null	Inactive	Inactive	<>Inactive
Relinquish_Default	Active	Active	Active	Active



Note: Bold font indicates the change invoked by write operation

Minimum Off Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_Off_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = P5
2. VERIFY Present_Value = INACTIVE
3. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P5
5. VERIFY Present_Value = INACTIVE
6. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
7. VERIFY Priority_ Array = NULL, ARRAY_INDEX = P5

--…(Steps 4-7:Check that a NULL value at P5 will NOT change ARRAY_INDEX = 6 or PV)

8. WRITE Present_Value = ACTIVE, PRIORITY = P5
9. VERIFY Present_Value = ACTIVE
10. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = P5

11. VERIFY Priority_ Array <> INACTIVE, ARRAY_INDEX = 6
--…(Steps 8-11:Check that an ACTIVE value at P5 will change ARRAY_INDEX = 6 and PV)

7.3.1.6.5 Minimum On Time – Writing at priorities numerically lesser than 6

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

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum on time is in effect.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off time to have expired. The Present_Value of the object tested is set to ACTIVE at a priority P5 (P5 < 6). Before Minimum_On_Time expires, Present_Value is written with values of NULL and INACTIVE and the Present_Value and Priority_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Inactive	Active	Active	Inactive
PA_Index = P5	Null	**Active**	**Null**	**Inactive**
PA_Index = 6	Null	Active	Active	<>Active
Relinquish_Default	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_On_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = P5
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P5
5. VERIFY Present_Value = ACTIVE
6. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
7. VERIFY Priority_ Array = NULL, ARRAY_INDEX = P5

--…(Steps 4-7:Check that a NULL value at P5 will NOT change ARRAY_INDEX = 6 or PV)

8. WRITE Present_Value = INACTIVE, PRIORITY = P5
9. VERIFY Present_Value = INACTIVE
10. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = P5
11. VERIFY Priority_ Array <> ACTIVE, ARRAY_INDEX = 6

--…(Steps 8-11:Check that an INACTIVE value at P5 will change ARRAY_INDEX = 6 and PV)

7.3.1.6.6 Minimum_Off_Time – Clock is not affected by additional write operations

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

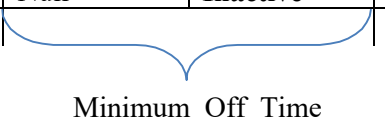
Purpose: To verify that the Minimum_Off_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE at priority P8, such that present-value and slot 6 in the priority-array change to INACTIVE. At time T1, which occurs before minimum off time expires, another write request, at priority P9, with a value of INACTIVE, is executed by the device. After minimum off time expires but before T1

+ Minimum_Off_Time, slot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Active	Inactive	Inactive	Inactive
PA_Index = P6	Null	Inactive	Inactive	Null
PA_Index = P~~X~~8	Null	**Inactive**	Inactive	Inactive
PA_Index = P~~¥~~9	Null	Null	**Inactive**	Inactive

Note: Bold font indicates the change invoked by write operation



Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE and slot 6 in the Priority_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6
3. WRITE Present_Value = INACTIVE, PRIORITY = P8
4. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6

--…(Execute step 5 at time T1)

5. WRITE Present_Value = INACTIVE, PRIORITY = P~~¥~~9

--…(Execute steps 6 and 7 before Minimum_Off_Time expires)

6. VERIFY Present_Value = INACTIVE
7. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
8. WAIT for Minimum_Off_Time to expire

--…(Execute step 9 before T1 + Minimum_Off_Time)

9. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6

Notes to Tester: P8 and P9 may assume any value in the Priority_Array (except 6) and may be equal.

7.3.1.6.7 Minimum_On_Time – Clock is not affected by additional write operations

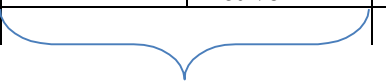
Reason for Change: This test is not specified in any SSPC proposal.

Purpose: To verify that the Minimum_On_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present_Value of the object being tested is set to INACTIVE and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to ACTIVE, at priority P8, such that present-value and slot 6 in the priority-array change to ACTIVE. At time T1, which occurs before minimum on time expires, another write request, at priority P9, with a value of ACTIVE, is executed by the device. After minimum on time expires but before T1 + Minimum_On_Time, ~~S~~slot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Inactive	Active	Active	Active
PA_Index = P6	Null	Active	Active	Null
PA_Index = P8	Null	**Active**	Active	Active
PA_Index = P9	Null	Null	**Active**	Active

Note: Bold font indicates the change invoked by write operation



Minimum On Time

Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE and slot 6 in the Priority_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Test Steps:

1. VERIFY Present_Value = INACTIVE
2. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6
3. WRITE Present_Value = ACTIVE, PRIORITY = P8
4. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6

--…(Execute step 5 at time T1)

5. WRITE Present_Value = ACTIVE, PRIORITY = P9

--…(Execute steps 6 and 7 before Minimum_On_Time expires)

6. VERIFY Present_Value = ACTIVE
7. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
8. WAIT for Minimum_On_Time to expire

--…(Execute step 9 before T1 + Minimum_On_Time)

9. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6

Notes to Tester: P8 and P9 may assume any value in the Priority_Array (except 6) and may be equal.

7.3.1.6.8 Ensuring Minimum_Off_Time starts at transition to INACTIVE

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

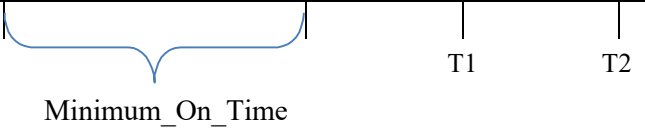
Purpose: To verify that Minimum_Off_Time does not start immediately after a write operation while Minimum_On_Time is in effect and present-value is ACTIVE.

Test Concept: The initial Present_Value of the object being tested is set to INACTIVE and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to ACTIVE at P9, where P9 is a priority between 7 and

16, such that present-value and slot 6 in the priority-array change to ACTIVE. Before Minimum_On_Time expires, Present_Value is written to INACTIVE at P7, where P7 is a priority between 7 and P9, such that Present_Value would change if Minimum_On_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:
T1 = the time when the INACTIVE request is executed by the device + Minimum_Off_Time
T2 = the time when the ACTIVE request is executed by the device + Minimum_On_Time + Minimum_Off_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Inactive	Active	Active	Inactive	Inactive	Inactive
PA_Index = 6	Null	Active	Active	Inactive	Inactive	Null
PA_Index = P7	Null	Null	**Inactive**	Inactive	Inactive	Inactive
PA_Index = P9		**Active**	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation



Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE and slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_On_Time and Minimum_Off_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum_On_Time and Minimum_Off_Time properties, this test shall be skipped.

Test Steps:

1. VERIFY Present_Value = INACTIVE
2. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6
3. WRITE Present_Value = ACTIVE, PRIORITY = P9
4. VERIFY Present_Value = ACTIVE
5. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6

--…(Execute steps 6 through 7 before Minimum_On_Time expires)

6. WRITE Present_Value = INACTIVE, PRIORITY = P7
7. VERIFY Present_Value = ACTIVE
8. VERIFY Priority_Array = ACTIVE, ARRAY_INDEX = 6
9. WAIT for Minimum_On_Time to expire

--…(Execute steps 10 and 11 before T1)

10. VERIFY Present_Value = INACTIVE
11. VERIFY Priority_ Array = INACTIVE, ARRAY_INDEX = 6

--…(Execute step 12 between T1 and T2

12. VERIFY Present_Value = INACTIVE
13. VERIFY Priority_ Array = INACTIVE, ARRAY_INDEX = 6

--..(Execute step 14 and 15 after T2)

14. VERIFY Present_Value = INACTIVE
15. VERIFY Priority_Array = NULL, ARRAY_INDEX = 6

Notes to Tester: P9 and P7 may be equal.

7.3.1.6.9 Ensuring Minimum_On_Time starts at transition to ACTIVE

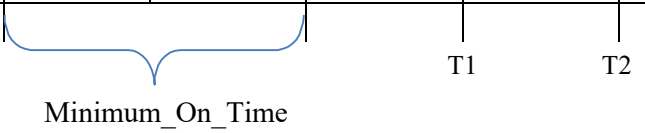
Reason for Change: This test is not specified in any SSPC proposal.

Purpose: To verify that Minimum_On_Time does not start immediately after a write operation while Minimum_Off_Time is in effect and present-value is INACTIVE.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE at P9, where P9 is a priority between 7 and 16, such that present-value and slot 6 in the priority-array change to INACTIVE. Before Minimum_Off_Time expires, Present_Value is written to ACTIVE at P7, where P7 is a priority between 7 and P9, such that Present_Value would change if Minimum_Off_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:
T1 = the time when the ACTIVE request is executed by the device + Minimum_On_Time
T2 = the time when the INACTIVE request is executed by the device + Minimum_Off_Time + Minimum_On_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Active	Inactive	Inactive	Active	Active	Active
PA_Index = 6	Null	Inactive	Inactive	Active	Active	Null
PA_Index = P7	Null	Null	**Active**	Active	Active	Active
PA_Index = P9		**Inactive**	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation



Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE and slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_On_Time and Minimum_Off_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum_On_Time and Minimum_Off_Time properties, this test shall be skipped.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6
3. WRITE Present_Value = INACTIVE, PRIORITY = P9
4. VERIFY Present_Value = INACTIVE
5. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6

--…(Execute steps 6 through 7 before Minimum_Off_Time expires)

6. WRITE Present_Value = ACTIVE, PRIORITY = P7
7. VERIFY Present_Value = INACTIVE
8. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
9. WAIT for Minimum_Off_Time to expire

--…(Execute steps 10 and 11 before T1)

10. VERIFY Present_Value = ACTIVE
11. VERIFY Priority_ Array = ACTIVE, ARRAY_INDEX = 6

--…(Execute step 12 between T1 and T2

12. VERIFY Present_Value = ACTIVE
13. VERIFY Priority_ Array = ACTIVE, ARRAY_INDEX = 6

--..(Execute step 14 and 15 after T2)

14. VERIFY Present_Value = ACTIVE
15. VERIFY Priority_Array = NULL, ARRAY_INDEX = 6

Notes to Tester: P9 and P7 may be equal.

7.3.1.6.10 Ensuring Minimum Times Are Not Effected By Time Changes

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

Purpose: To verify that minimum times are not affected by changing the time in a device via TimeSynchronization or UTCTimeSynchronization requests.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE such that present-value and slot 6 in the priority-array change to INACTIVE. Before Minimum_Off_Time expires, the time is changed to a value T1 which is more than Minimum_Off_Time in the future and Present_Value and Slot 6 in the priority-array are read to verify that they were not affected by the time change. After Minimum_Off_Time expires, slot 6 in the priority-array is read again to verify that it is no longer INACTIVE.

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE and slot 6 in the Priority_Array has a value of NULL. If the IUT does not support TimeSynchronization or UTC-TimeSynchronization, then this test shall be omitted.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_ Array = NULL, ARRAY_INDEX = 6
3. WRITE Present_Value = INACTIVE
4. VERIFY Present_Value = INACTIVE
5. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
6. TRANSMIT
   DA = GLOBAL BROADCAST,
   SA = TD
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = TimeSynchronization-Request,
   Date = T1,
   Time = T1
7. TRANSMIT
   DA = GLOBAL BROADCAST,
   SA = TD
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = UTC-TimeSynchronization-Request,
   Date = T1,
   Time = T1
8. VERIFY Priority_Array = INACTIVE, ARRAY_INDEX = 6
9. WAIT (the remainder of Minimum_Off_Time)
10. VERIFY Priority_Array <> INACTIVE, ARRAY_INDEX = 6

Notes to Tester: The test above is written for Minimum_Off_Time. To execute this test for Minimum_On_Time, use INACTIVE

BTL-TP14.0h-2: Epics Consistency Test Changes

Overview:

The purpose of this document is to improve the documentation for **Clause 135.1-2013 on test 5 EPICS Consistency** which is different from other tests in many ways. Because what's currently described in this clause isn't clear to explain the difference between this test and the rest of the tests, many readers seem to have problems understanding what this test actually is and they end up asking the similar questions to BTL.

Reason for Change:

1) Unlike the other tests, this **Clause 135.12013 on test 5 EPICS Consistency** has no test steps or test setup. However the current documentation does not describe that.
2) The whole section has generic statements only as if to say 'ensure it all works', but more detailed test directives as to what exactly are required in order to pass the test is needed for each section.

Changes:
[Add to BTL Specified Tests, Clause 5]

5. EPICS CONSISTENCY TESTS

These tests are static tests of the EPICS and do not involve interrogating the IUT. There are no Test Configuration or Test Step sections with TCSL in these tests because the tests are static tests of the EPICS and not tests of the IUT itself.

Each implementation shall be tested to ensure consistency among interrelated data elements

These tests shall include:

(a) All object types required by the specified BIBBs shall be indicated as supported in the Standard Object Types Supported section of the EPICS.

(b) A minimum of one instance of each object type required by the specified BIBBs shall be included in the test database.

(c) The *Protocol_Object_Types_Supported* property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.

(d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.

(e) The *Protocol_Services_Supported* property of the Device object in the test database shall indicate support for each application service for which the supported BIBBs requires support for execution of the service.

(f) The object types listed in the Standard Object Types Supported section of the EPICS shall have a one-to-one correspondence with object types listed in the *Protocol_Object_Types_Supported* property of the Device object contained in the test database.

(g) For each object type listed in the Standard Object Types Supported* section of the EPICS there shall be at least one object of that type in the test database. **

**An object type is supported if it can be made to exist in the IUT's database.*

***with the exception of the case where File objects are only present in the IUT during Backup and Restore. An object type is supported if it can be made to exist in the IUT's database.*

(h) There shall be a one-to-one correspondence between the objects listed in the Object_List property of the Device object and the objects included in the test database. The Object_List property and the test database shall both include all proprietary objects. Properties of proprietary objects that are not required by BACnet Clause 23.4.3 need not be included in the test database.

(i) For each object included in the test database, all required properties for that object as defined in Clause 12 of BACnet shall be present. *Standard properties which are not defined for the implemented Protocol_Revision shall not be present.* In addition, if any of the properties supported for an object require the conditional presence of other properties, their presence shall be verified.

(j) For each property that is required to be writable, or conditionality writable, that property shall be marked as writable, or conditionality writable, in the EPICS.

(k) The length of the Protocol_Services_Supported bitstring shall have the number of bits defined for BACnetProtocolServicesSupported for the IUT's declared protocol revision.

(l) The length of the Protocol_Object_Types_Supported bitstring shall have the number of bits defined for BACnetObjectTypesSupported for the IUT's declared protocol revision

(m) For each object included in the test database, any properties that are deprecated or removed, shall not appear after the Protocol_Revision in which the property was deprecated or removed.

(n) If the Protocol Revision property is present in the Device object and its value is greater than or equal to 14, the Property List property of each object included in the test database shall have one entry for each property present including non-standard properties with the exception of Object_Type, Object_Identifier, Object_Name and Property_List

(o) If the Segmentation_Supported property in the Device object is SEGMENTED_BOTH or SEGMENTED_RECEIVE, then the value of the Max_Segments_Accepted property of the Device object shall be greater than 1.

BTL-TP14.0h-3: Active_COV_Subscriptions SubscribeCOV Test Changes

Overview:

The existing test 135.1-2013 - 7.3.2.10.1 was accidentally edited into 135.1-2013 from 135.1-2009*d*-2, without including steps 15 through 21.

Changes:

[In BTL Specified Tests, derive from existing test 135.1-2013 - 7.3.2.10.1 This goes in an errata, not into an addenda.]

7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test

Purpose: This test case verifies that the IUT correctly updates the Active_COV_Subscriptions property when COV subscriptions are created, cancelled and timed-out using SubscribeCOV.

Configuration Requirements: In this test, the tester shall choose three standard objects, O_1, O_2, and O_3, for which the device supports SubscribeCOV. O_1, O_2, and O_3 are not required to refer to different objects. The tester shall also choose three non-zero unique process identifiers, P_1, P_2, and P_3, and three non-zero lifetimes L_1, L_2, and L_3. Lifetime L_1 shall be long enough to allow the initial part of the test to run through to step 14. Lifetimes L_2 and L_3 shall be long enough for the whole test to be completed without expiring.

The IUT shall start the test with no entries in its Active_COV_Subscriptions property.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = P_1,
   'Monitored Object Identifier' = O_1,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L_1
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = P_1,
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O_1,
   'Time Remaining' = (a value approximately equal to L_1),
   'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT BACnet-SimpleACK-PDU
5. VERIFY Active_COV_Subscriptions = {{ {TD, P_1}, {O_1, Present_Value }, TRUE, (a value less than L_1), (a valid Increment if the property is REAL) }}
6. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = P_2,
   'Monitored Object Identifier' = O_2,
   'Issue Confirmed Notifications' = FALSE,
   'Lifetime' = L_2
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = P_2,
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O_2,
   'Time Remaining' = (a value approximately equal to L_2),
   'List of Values' = (values appropriate to the object type of the monitored object)
9. VERIFY Active_COV_Subscriptions = {{ {TD, P_1}, {O_1, Present_Value}, TRUE, (a value less than L_1), (a valid Increment if the property is REAL) },
   { {TD, P_2}, {O_2, Present_Value}, FALSE, (a value less than L_2), (a valid Increment if the property is REAL) }}
10. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = P_3,

'Monitored Object Identifier' = O_3,
'Issue Confirmed Notifications' = FALSE,
'Lifetime' = L_3

11. RECEIVE BACnet-SimpleACK-PDU
12. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = P_3,
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = O_3,
    'Time Remaining' = (a value approximately equal to L_3),
    'List of Values' = (values appropriate to the object type of the monitored object)
13. VERIFY Active_COV_Subscriptions = {{{TD, P_1}, {O_1, Present_Value}, TRUE, (a value less than L_1), (a valid Increment if the property is REAL)}, {{TD, P_2}, {O_2, Present_Value}, FALSE, (a value less than L_2), (a valid Increment if the property is REAL)}, {{TD, P_3}, {O_3, Present_Value}, FALSE, (a value less than L_3), (a valid Increment if the property is REAL)}}
14. WAIT L_1 + the IUT's timer granularity
15. ***ER Acti e S sc ipti s T 2 2 ese t e ALSE e ess t L 2***
    id c e e tif t e p pe t is REAL
    T 3 3 ese t e ALSE e ess t L 3
    id c e e tif t e p pe t is REAL
16. ***TRA S TS sc i e -Re est***
    S sc i e cess de tifie 3
    it ed ect de tifie 3
17. ***RE E EBA et-Si p eA -***
18. ***ER Acti e S sc ipti s T 2 2 ese t e ALSE e ess t L 2***
    id c e e tif t e p pe t is REAL
19. ***TRA S TS sc i e -Re est***
    S sc i e cess de tifie 2
    it ed ect de tifie 2
20. ***RE E EBA et-Si p eA -***
21. ***ER Acti e S sc ipti s***

[In BTL Test Plan, to existing test in sections 4.10.1 DS-COV-B Base Requirements, revise the test reference, as indicated.]

4.10 Data Sharing - COV -B

4.10.1 Base Requirements

Base requirements must be met by any IUT claiming conformance to this BIBB.

. . .		
~~135.1-2013~~**BTL - 7.3.2.10.1 - Active_COV_Subscriptions SubscribeCOV Test**		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests~~ASHRAE 135.1-2013~~*** .
	Test Conditionality	This test must be executed if the device claims conformance to protocol revision 1 or higher.
	Test Directives	
	Testing Hints	
	Notes & Results	

BTL-TP14.0h-4: Remove duplicate EPICS Consistency Testing

Overview:

The purpose of this document is to remove duplicate references to test 5 EPICS Consistency from the Test Plan. Currently, this test is referenced from 2.1.2 (Basic Functionality Test), 3.10.1 (Objects – Device object) and 3.18.1 (Objects – Proprietary object). It makes sense to retain its reference within 2.1.2 only and remove others.

Reason for Change:

On the Checklist under Section 2 Basic Bacnet Functionality, it currently says "**applies to all BACnet devices"** and EPICS Consistency Tests is selected by default as shown below. It appears to be very clear that this test must be executed for all IUTs. Having said that, it is unnecessary to have duplicate references from other sections. Removing them would clean up and simplify the Test Plan.

2 Basic BACnet Functionality

Support	Listing	Option
Basic Functionality (applies to all BACnet devices)		
X	R	Base Requirements
X	R	EPICS Consistency Tests
X	R	Supports DS-RP-B

Changes:
[In BTL Test Plan, modify the Device Object entry]

3.10 Device Object

3.10.1 Base Requirements

~~Base requirements must be met by any IUT.~~

B□se □e□i□e□e□ts □ □st □e □et □□ □□□□ T t□□tc□□ c□□t□i□□e□ice □□ects. T□e□e □e □□□se □e□i□e□e□tstests f□□t□is secti□□.

~~135.1-2013 - 5 - EPICS ConsistencyONSISTENCY Tests~~		
	~~Test Method~~	~~Manual~~
	~~Configuration~~	~~As per *ASHRAE 135.1-2013*.~~
	~~Test Conditionality~~	~~Must be executed~~
	~~Test Directives~~	
	~~Testing Hints~~	
	~~Notes & Results~~	

3.10.2 Supports Database_Revision Property

Ensures that the Database_Revision property increments correctly.

135.1-2013 - 7.3.2.10.3 - Successful Increment of the Database_Revision Property after Creating an Object		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-2013***.
	Test Conditionality	If the device implements protocol revision 2 or higher, this test must be executed. If the IUT does not support object creation through any means, not just through BACnet services, this test may be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	
135.1-2013 - 7.3.2.10.4 - Successful Increment of the Database_Revision Property after Deleting an Object		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-2013***.
	Test Conditionality	If the device implements protocol revision 2 or higher, this test must be executed. If the IUT does not support object deletion through any means, not just through BACnet services, this test may be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	
135.1-2013 - 7.3.2.10.5 - Successful Increment of the Database_Revision Property after Changing the Object_Name Property of an Object		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-2013***.
	Test Conditionality	If the device implements protocol revision 2 or higher, this test must be executed. If the IUT does not support a changeable Object_Name property in any object, this test may be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	

BTL - 7.3.2.10.6 - Successful Increment of the Database_Revision Property after Changing the Object_Identifier Property of an Object		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***.
	Test Conditionality	If the device implements protocol revision 2 or higher, this test must be executed. If the IUT does not support a changeable Object_Identifier property in any object, this test may be skipped.
	Test Directives	
	Testing Hints	
	Notes & Results	

3.10.3 Supports Time_Synchronization_Recipients

The Time_Synchronization_Recipients property is present in the Device Object.

135.1-2013 - 13.2.1 - TimeSynchronization Recipients Test, Protocol_Revision < 7		
	Test Method	Manual
	Configuration	As per ***ASHRAE 135.1-2013.***
	Test Conditionality	Must be executed if Protocol_Revision < 7.
	Test Directives	
	Testing Hints	
	Notes & Results	

3.10.4 Supports UTC_Time_Synchronization_Recipients

The UTC_Time_Synchronization_Recipients property is present in the Device Object.

Verify Checklist		
	Test Method	Manual
	Configuration	
	Test Conditionality	Must be executed if Protocol_Revision ≥ 7.
	Test Directives	Verify that the IUT claims support for DM-ATS-A in the checklist.
	Testing Hints	
	Notes & Results	

3.10.5 Contains a Writable Local_Date Property

The IUT contains, or can be made to contain, a Device object that contains a writable Local_Date property.

BTL - 7.2.X4 - Date Non-Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	Must be executed.
	Test Directives	Apply to the Local_Date property.
	Testing Hints	
	Notes & Results	

3.10.6 Contains a Writable Local_Time Property

The IUT contains, or can be made to contain, a Device object that contains a writable Local_Time property.

BTL - 7.2.X5 - Time Non-Pattern Properties Test		
	Test Method	Manual
	Configuration	As per ***BTL Specified Tests***
	Test Conditionality	Must be executed.
	Test Directives	Apply to the Local_Time property.
	Testing Hints	
	Notes & Results	

3.18 Proprietary Objects

3.18.1 Base Requirements

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

B□se □e□i□e□e□ts □ □st □e □et □□ □□□□□T t□□tc□□c□□t□i□□□□p□iet□□□□□ects. T□e□e□□e □□□□se □e□i□e□e□tstests f□□ t□is secti□□.

~~135.1-2013_5_EPICS CONSISTENCY TESTS~~		
	~~Test Method~~	~~Manual~~
	~~Configuration~~	~~As per *ASHRAE 135.1-2013*.~~
	~~Test Conditionality~~	~~Must be executed~~
	~~Test Directives~~	
	~~Testing Hints~~	
	~~Notes & Results~~	

BTL-TP14.0h-5: ReadProperty Test Plan Conditionality Change

Overview:

Though the test 7.1.X3 within *Configuration Requirements*, itself has correct wording to skip if IUT < 14, to be in line with our normal usage we should also have that correct conditionality in the *Test Plan.*

Changes:

[In BTL Test Plan, revise the Test Conditionality, as indicated]

4.2Data Sharing - ReadProperty - B

4.2.1 Base Requirements

All devices must support this BIBB.

	. . .	
BTL - 7.1.X3 - Verifying Property_List against the EPICS		
	Test Method	Manual
	Configuration	As per *BTL Specified Tests*.
	Test Conditionality	Must be executed *if the IUT claims Protocol_Revision 14 or greater*.
	Test Directives	
	Testing Hints	
	Notes & Results	