BACnet® TESTING LABORATORIES
ADDENDA

# Addendum ay to BTL Test Package 16.1

**Revision 4**
**Revised September 30, 2020**

Approved by the BTL Working Group on June 25, 2020.
Approved by the BTL Working Group Voting Members on September 30, 2020.
Published on October 1, 2020.

**[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 16.1 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-16.1ay-1: SCHED-AVM-A and VM-A for Timer Test Changes - BTLWG-84

**Overview:**

The wording in SCHED-AVM-A and SCHED-VM-A is updated for the Timer object added in PR_17, and the long-untouched Testing Hints in two existing Base Requirements Test Plan entries is replaced with Testing Directives instead, using wording similar to what was adopted on 07-Feb-2019 in the DS-V-A and DS-M-A.

There are five existing Checklist line-items in SCHED-AVM-A all with Conditionality Code: R. The references to other BIBBs, as Verify SCHED-VM-A, and Verify DM-OCD-A by convention in BTL Checklist are always mentioned with a dedicated section, wherever that testing is applicable. But the testing which is located in the last two are here moved, along with the new tests, into Base Requirements and the extraneous final two Checklist line-items in SCHED-AVM-A with Conditionality Code: R, are removed.

**Changes:**

[In BTL Checklist, modify the SCHED-AVM-A entries as directed below.]

| Support | Listing | Option |
|---|---|---|
| **Scheduling - Advanced View Modify - A** | | |
| | R | Base Requirements |
| | R | Supports SCHED-VM-A |
| | R | Supports DM-OCD-A |
| | ~~R~~ | ~~Is able to read a Schedule object that is self-inconsistent with regard to the scheduled datatype, and modify it to be consistent~~ |
| | ~~R~~ | ~~Is able to change the datatype that a Schedule object schedules~~ |

[In BTL Test Plan, Update the Test Directives and remove the Testing Hints as shown below]

### 6.1 Scheduling - Advanced View and Modify - A

#### 6.1.1 Base Requirements

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

| **135.1-2013 - 8.18.3 - Reading and Presenting Properties** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Repeat the test for each of the standard object types, except Timer object for devices claiming Protocol_Revision 16 and below, and associated properties specified in Table K-20 in the 135 standard. The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries. |
| | **Testing Hints** | ~~Verify for Weekly_Schedule, Exception_Schedule, Effective_Period , Date_List , Schedule_Default, List_Of_Object_Property_References, Priority_For_Writing, and Out_Of_Service properties. See Addendum L for requirements details.~~ ~~The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries.~~ |
| **135.1-2013 - 8.22.4 - Accepting Input and Modifying Properties** | | |

| | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Repeat the test for each of the standard object types, except Timer object for devices claiming Protocol_Revision 16 and below, and associated properties specified in Table K-20 in the 135 standard.<br>The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries. |
| | **Testing Hints** | ~~Verify for Weekly_Schedule, Exception_Schedule, Effective_Period , Date_List , Schedule_Default, List_Of_Object_Property_References, Priority_For_Writing, and Out_Of_Service properties. See Addendum L for requirements details.~~<br>~~The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries.~~ |
| **135.1-2013 - 13.10.6 - Modify a Self-inconsistent Schedule to be Consistent** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | The IUT is not required to present the self-inconsistent schedule, only that the IUT write out a consistent schedule after the modification. Ideally, the IUT should involve the user in the process of modifying the schedule to be consistent, presenting the full data of the schedule and allowing the user to edit the data to correct the inconsistencies. If the IUT modifies the schedule automatically to make it consistent, it should at least notify the user that the schedule has been modified. It is not acceptable for the IUT to modify the schedule without any indication to the user that this was done. |
| **135.1-2013 - 13.10.7 - ~~Is able to change the datatype~~ *Change the Datatype* that a Schedule *Object Schedules* ~~object schedules~~** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Change the datatype of the Schedule to each schedule datatype required by the BIBB, as well as any additional which are supported by the IUT. |
| | **Testing Hints** | ~~The tester should attempt to change the datatype of the Schedule to each schedule datatype supported by the IUT.~~ |
| **BTL - 13.10.X8 - Modify a Self-inconsistent Timer to be Consistent** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | The IUT is not required to present the self-inconsistent timer, only that the IUT write out a consistent timer after the modification. Ideally, the IUT should involve the user in the process of modifying the timer to be consistent, presenting the full data of the timer and allowing the user to edit the data to correct the inconsistencies. If the IUT modifies the timer automatically to make it consistent, it should at least notify the user that the timer has been modified. It is not acceptable for the IUT to modify the timer without any indication to the user that this was done. |
| **BTL - 13.10.X9 - Change the Datatype that a Timer Object References** | | |
| | **Test Conditionality** | Must be executed if the IUT claims Protocol_Revision 17 or higher. |
| | **Test Directives** | Change the datatype of the Timer to each timer datatype required by the BIBB, as well as any additional which are supported by the IUT. |
| | **Testing Hints** | |

[In BTL Test Plan, Remove sections 6.1.4 and 6.1.5 as shown below.]

### ~~6.1.4 Is able to Read a Schedule Object that is Self-inconsistent with regard to the Scheduled Datatype, and Modify it to be Consistent~~

~~The IUT is capable of modifying schedule objects that are self-inconsistent with regards to the scheduled datatype.~~

| ~~135.1-2013 - 13.10.6 - Modify a Self-inconsistent Schedule to be Consistent~~ | | |
|---|---|---|
| | ~~**Test Conditionality**~~ | ~~Must be executed.~~ |
| | ~~**Test Directives**~~ | |
| | ~~**Testing Hints**~~ | ~~The IUT is not required to present the self-inconsistent schedule, only that the IUT write out a consistent schedule after the modification. Ideally, the IUT should involve the user in the process of modifying the schedule to be consistent, presenting the full data of the schedule and allowing the user to edit the data to correct the inconsistencies. If the IUT modifies the schedule automatically to make it consistent, it should at least notify the user that the schedule has been modified. It is not acceptable for the IUT to modify the schedule without any indication to the user that this was done.~~ |

### ~~6.1.5 Is able to Change the Datatype that a *of* Schedule *and Timer Objects* Object Schedules~~

~~The IUT supports the ability for the user to change the datatype of the schedule object.~~

| ~~135.1-2013 - 13.10.7 - Is able to change the datatype *Change the Datatype* that a Schedule *Object Schedules* object schedules~~ | | |
|---|---|---|
| | ~~**Test Conditionality**~~ | ~~Must be executed.~~ |
| | ~~**Test Directives**~~ | |
| | ~~**Testing Hints**~~ | ~~The tester should attempt to change the datatype of the Schedule to each schedule datatype supported by the IUT.~~ |

### 6.1.3 Supports DM-OCD-A

The IUT supports DM-OCD-A for the Timer, Calendar, and Schedule objects.

| **Verify Checklist** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify the IUT claims support for the BIBB DM-OCD-A and that the Calendar, and Schedule objects can be created and deleted. If the IUT claims Protocol_Revision 17 or higher, verify that the creation and deletion of Timer objects is also claimed. |
| | **Testing Hints** | |

## 6.2 Scheduling - View and Modify - A

### 6.2.1 Base Requirements

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

| **135.1-2013 - 8.18.3 - Reading and Presenting Properties** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed *unless the IUT also* ~~This test may be skipped if the IUT~~ claims support for SCHED-AVM-A. |
| | **Test Directives** | Repeat the test for <u>each</u> of the standard object types, defined in the claimed Protocol_Revision, and associated properties specified by SCHED-VM-A. The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples |

| | | |
|---|---|---|
| | | per day. The Calendar Date_List used in this test should contain 32 calendar entries. |
| | **Testing Hints** | ~~Verify for Weekly_Schedule, Exception_Schedule, Effective_Period and Date_List properties. See Addendum L for requirements details.~~<br><br>~~The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries.~~ |
| **135.1-2013 - 8.22.4 - Accepting Input and Modifying Properties** | | |
| | **Test Conditionality** | Must be executed unless the IUT also ~~This test may be skipped if the IUT~~ claims support for SCHED-AVM-A. |
| | **Test Directives** | Repeat the test for each of the standard object types, defined in the claimed Protocol_Revision, and associated properties specified by SCHED-VM-A.<br>The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries. |
| | **Testing Hints** | ~~Verify for Weekly_Schedule, Exception_Schedule, Effective_Period and Date_List properties. See Addendum L for requirements details.~~<br><br>~~The reference schedule used during this test should include an Exception_Schedule that contains 255 entries and contain 12 BACnetTimeValue tuples per entry. The reference schedule should also contain a Weekly_Schedule which contains 6 BACnetTimeValue tuples per day. The Calendar Date_List used in this test should contain 32 calendar entries.~~ |

[In BTL Specified Tests, add two tests 13.10.X8 and 13.10.X9]

### 13.10.X8 Modify a Self-inconsistent Timer to be Consistent

Purpose: Demonstrate that the IUT can read a Timer that is self-inconsistent with regard to datatype and modify it to be consistent. This capability is required in order to be able to fix Timer objects that may be in a self-inconsistent state when the process of changing datatype in an earlier attempt, was interrupted.

Test Configuration: A reference device contains any valid Timer object T that supports modifying the datatype.

Test Steps:

1. MAKE (the IUT modify T so that the datatype which State_Change_Values holds, and which List_Of_Object_Property_References values points to, is consistent)

2. CHECK (Did the IUT write a consistent timer?)

Notes to Tester: A consistent timer is one that meets all of these criteria:

1. All non-NULL values used in the State_Change_Values property shall be of the same datatype.

2. All properties referenced by the List_Of_Object_Property_References are writable with that datatype.

Until those conditions are met, the inconsistency can be detected by reading the Reliability property which will have the value CONFIGURATION_ERROR.

### 13.10.X9 Change the Datatype that a Timer Object References

Purpose: Verify that the IUT can alter the datatype of a Timer object.

Test Configuration: A reference device contains any valid Timer object that supports modifying the datatype.

Test Steps:

1. MAKE (the State_Change_Values have, and/or List_Of_Object_Property_References point to a scheduled datatype that is different)
2. CHECK (Did the IUT write the modified properties correctly?)

Notes to Tester: It is acceptable that the IUT modify the Timer one property at a time, so there may be a time period when the Timer is in a self-inconsistent state during reconfiguration.

## BTL-16.1ay-2: Add Timer Object Testing and CHANGE_OF_TIMER algorithm Testing - BTLWG-81

**Overview:**

Testing for the Timer object added in PR_17 is added, and two BIBBs AE-N-I-B and AE-N-E-B are updated for the CHANGE_OF_TIMER algorithm associated with Timer object.

**Changes:**

[In Checklist, remove the reference by footnote to "Contact BTL for this algorithm." from the CHANGE_OF_TIMER entry]

| **Support** | **Listing** | **Option** |
|---|---|---|
| **Alarm and Event Management - Notification - Internal - B** | | |
| | R | Base Requirements |
| | … | ... |
| | C[3,~~5~~,7] | Implements the CHANGE_OF_TIMER algorithm |
| | | [1] Required if EventNotifications with service parameter AckRequired = True can be issued.<br>[2] At least one of these options must be supported to claim support for this BIBB.<br>[3] At least one of these options must be supported to claim support for this BIBB. It is recommended that a standard BACnet algorithm be used instead of a proprietary algorithm whenever possible.<br>[4] At least one of these options must be supported to claim support for this BIBB. The BACnetDateTime form of the timestamp is the recommended option.<br>[5] Contact BTL for interim tests for this algorithm.<br>[6] Protocol_Revision 16 or higher must be claimed.<br>[7] Protocol_Revision 17 or higher must be claimed.<br>[8] Protocol_Revision 18 or higher must be claimed. |

[In Checklist, remove the reference by footnote to "Contact BTL for this algorithm." from the CHANGE_OF_TIMER entry]

| **Support** | **Listing** | **Option** |
|---|---|---|
| **Alarm and Event Management - Notification - External - B** | | |
| | R | Base Requirements |
| | … | ... |
| | C[1,~~2~~,4] | Implements the CHANGE_OF_TIMER algorithm |
| | | [1] One of these options must be supported to claim support for this BIBB. It is recommended that a standard BACnet algorithm be used instead of a proprietary algorithm whenever possible.<br>[2] Contact BTL for interim tests for this algorithm.<br>[3] Protocol_Revision 16 or higher must be claimed.<br>[4] Protocol_Revision 17 or higher must be claimed.<br>[5] Protocol_Revision 18 or higher must be claimed. |

[In Checklist, completely replace existing **Timer Object** entry with the below entry.]

| **Support** | **Listing** | **Option** |
|---|---|---|
| **Timer Object** | | |
| | R | Base Requirements |
| | O | Supports Writable Out_Of_Service Property |
| | O | Supports Writable Present_Value while Out_Of_Service is FALSE |
| | O[1] | Supports Update_Time |

| Support | Listing | Option |
|---|---|---|
| | O | Supports Expiration_Time |
| | O | Supports Default_Timeout |
| | O | Supports Priority_For_Writing |
| | O | Supports writable Priority_For_Writing and List_Of_Object_Property_References |
| | O | Can contain an Object with Reliability_Evaluation_Inhibit Property |
| | [1] Required if the IUT supports intrinsic reporting for Timer object type. | |

[In Checklist, completely replace existing **Scheduling - Timer - Internal - B** with below entry.]

| Support | Listing | Option |
|---|---|---|
| **Scheduling - Timer - Internal - B** | | |
| | R | Base Requirements |
| | R | Supports writable Priority_For_Writing property |
| | R | Supports writable Default_Timeout property |
| | | |

[In Checklist, completely replace the **Scheduling - Timer - External - B** entry with the below entry.]

| Support | Listing | Option |
|---|---|---|
| **Scheduling - Timer - External - B** | | |
| | R | Base Requirements |
| | R | Supports SCHED-TMR-I-B |
| | R | Supports DS-WP-A |
| | R | Supports writable List_Of_Object_Property_References property |
| | R | Supports references to objects in external devices |
| | | |

[In Test Plan, replace existing placeholder section 3.57 with the following.]

## 3.57 Timer Object

### 3.57.1 Base Requirements

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

| **BTL - 7.3.2.X57.1.1 - Timer State_Change_Values** | | |
|---|---|---|
| | **Test Conditionality** | If the IUT does not support a State_Change_Values property in any Timer object, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.2 - Timer Running then Expired Test** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.5 - Restarting An Expired Timer** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object, this test shall be skipped. |

| | | |
|---|---|---|
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.6 - Already Running Timer restarted by writing the Present_Value** | | |
| | **Test Conditionality** | If the IUT supports writable Present_Value in Timer object while Out_Of_Service is FALSE, there is no need to repeat this test. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.7 - Already Running Timer restarted with Default_Timeout** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.10 - Forcing Timer Expiration by writing Zero** | | |
| | **Test Conditionality** | If Present_Value is never writable, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.3 - Invalid Property Writing in a Timer** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.4 - Expired Timer Ignores Writing Zero** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.5 - Expired Timer Ignores Writing FALSE** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.6 - Idle Timer Ignores Writing Zero** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.7 - Idle Timer Ignores Writing FALSE** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.8 - Idle Timer Ignores Writing IDLE** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.2 Supports Writable Out_Of_Service Property

The Out_Of_Service property in Timer objects is writable.

| | | |
|---|---|---|
| **135.1-2013 - 7.3.1.1 - Writing Out_Of_Service, Status_Flags, and Reliability Test** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.4 - Running Timer by writing the Present_Value** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | |
| | **Testing Hints** | If Present_Value is only writable while Out_Of_Service is TRUE, then the presence of Min_Pres_Value and Max_Pres_Value is optional. |
| **BTL - 7.3.2.X57.1.11 - Forcing Timer Expiration by writing FALSE** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.12 - Forcing Timer Expiration by writing IDLE** | | |

| | | |
|---|---|---|
| | **Test Conditionality** | If the IUT does not support a writable Timer_State, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.13 - Resetting Timer by writing IDLE** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_State this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.3 Supports Writable Present_Value while Out_Of_Service is FALSE

The Present_Value property in a Timer object is writable while Out_Of_Service is FALSE.

| | | |
|---|---|---|
| **Verify EPICS** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify that Min_Pres_Value and Max_Pres_Value properties are present in the object. |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.4 - Running Timer by writing the Present_Value** | | |
| | **Test Conditionality** | Must be executed. Note that this test is included in multiple Test Plan sections for the Timer object but it needs to only be executed once. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.6 - Already Running Timer restarted by writing the Present_Value** | | |
| | **Test Conditionality** | Must be executed. Note that this test is included in multiple Test Plan sections for the Timer object but it needs to only be executed once. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.11 - Forcing Timer Expiration by writing FALSE** | | |
| | **Test Conditionality** | Must be executed. Note that this test is included in multiple Test Plan sections for the Timer object but it needs to be only executed once. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.12 - Forcing Timer Expiration by writing IDLE** | | |
| | **Test Conditionality** | Must be executed. Note that this test is included in multiple Test Plan sections for the Timer object but it needs to be only executed once. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.13 - Resetting Timer by writing IDLE** | | |
| | **Test Conditionality** | Must be executed. Note that this test is included in multiple Test Plan sections for the Timer object but it needs to be only executed once. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.4 Supports Update_Time

The IUT supports the Update_Time property in a Timer object.

| | | |
|---|---|---|
| **Verify EPICS** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify that Local_Date and Local_Time properties are present in the Device object. |
| | **Testing Hints** | |

### 3.57.5 Supports Expiration_Time

The IUT supports the Expiration_Time property in a Timer object.

| | | |
|---|---|---|
| **Verify EPICS** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify that Local_Date and Local_Time properties are present in the Device object. |
| | **Testing Hints** | |

| **BTL - 7.3.2.X57.1.14 - Timer Object Operation Unaffected by Changes to Local_Time and Local_Date** | | |
|---|---|---|
| | **Test Conditionality** | Must be Executed. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.6 Supports Default_Timeout

The IUT supports the Default_Timeout property in a Timer object.

| **BTL - 7.3.2.X57.1.3 - Default_Timeout Test** | | |
|---|---|---|
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object which contains a Default_Timeout, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.5 - Restarting An Expired Timer** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object which contains a Default_Timeout, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.7 - Already Running Timer restarted with Default_Timeout** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object which contains a Default_Timeout, this test shall be skipped. If every Timer only goes into RUNNING state with an Initial_Value equal to Default_Value, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.1.16 - Changing Default_Timeout Test** | | |
| | **Test Conditionality** | If the IUT does not support a writable Timer_Running in any Timer object which contains a writable Default_Timeout, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.9 - Default_Timeout Written Outside Supported Range** | | |
| | **Test Conditionality** | If Default_Timeout property is not writable in any Timer object, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.7 Supports Priority_For_Writing

The IUT supports the Priority_For_Writing property in a Timer object.

| **BTL - 7.3.2.X57.1.15 - Changes made by State_Change_Values are at Correct Priority** | | |
|---|---|---|
| | **Test Conditionality** | If List_Of_Object_Property_References cannot reference a commandable property, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |

### 3.57.8 Supports Writable Priority_For_Writing and List_Of_Object_Property_References

The IUT supports writable Priority_For_Writing and List_Of_Object_Property_References properties in a Timer object.

| **BTL - 7.3.2.X57.1.8 - Timer accepts all the required datatypes in an Internal Reference** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test shall be executed with a Timer object that can be configured to monitor a property within the IUT. Repeat the test with the List_Of_Object_Property_References making its references to, and the State_Change_Values property containing non-NULL values of each of these datatypes: NULL, BOOLEAN, Unsigned, INTEGER, REAL, and ENUMERATED. Support for writing to properties with other datatypes is optional. |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.1 - Writing Timer with an Unsupported External Reference** | | |

| | | |
|---|---|---|
| | **Test Conditionality** | If the IUT claims support for SCHED-TMR-E-B, this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.2.X57.2.2 - Writing an Unsupported Datatype to State_Change_Values** | | |
| | **Test Conditionality** | If there is no datatype which the IUT does not support, then this test shall be skipped. |
| | **Test Directives** | Write a value using a datatype which is not supported in the Timer instance. |
| | **Testing Hints** | |

### 3.57.9 Can Contain an Object with Reliability_Evaluation_Inhibit Property

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

| | | |
|---|---|---|
| **BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test** | | |
| | **Test Conditionality** | If no Timer object exists in the IUT for which fault conditions can be detected, then this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |
| **BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test** | | |
| | **Test Conditionality** | If no Timer object exists in the IUT for which fault conditions can be detected and which supports intrinsic reporting, then this test shall be skipped. |
| | **Test Directives** | |
| | **Testing Hints** | |

[In Test Plan, in existing sections 5.2.41 add four new tests each.]

## 5.2 Alarm and Event Management - Notification - Internal - B

### 5.2.41 Implements the CHANGE_OF_TIMER Algorithm

~~Contact the BTL for interim tests for this algorithm.~~ The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_TIMER.

| | | |
|---|---|---|
| **BTL - 8.4.10.X1 - CHANGE_OF_TIMER ConfirmedEventNotification Test** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_TIMER. |
| | **Testing Hints** | |
| **BTL - 8.4.10.X2 - CHANGE_OF_TIMER Offnormal-to-Offnormal ConfirmedEventNotification** | | |
| | **Test Conditionality** | If pAlarmValues cannot be configured with two different values to which pMonitored will become equal, this test shall be skipped. |
| | **Test Directives** | This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_TIMER. |
| | **Testing Hints** | |
| **BTL - 8.5.10.X1 - CHANGE_OF_TIMER UnconfirmedEventNotification Test** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_TIMER. |
| | **Testing Hints** | |
| **BTL - 8.5.10.X2 - CHANGE_OF_TIMER Offnormal-to-Offnormal UnconfirmedEventNotification** | | |
| | **Test Conditionality** | If pAlarmValues cannot be configured with two different values to which pMonitored will become equal, this test shall be skipped. |
| | **Test Directives** | This test must be repeated once for each object type that is capable of generating event notifications with an Event_Type of CHANGE_OF_TIMER. |

| | | |
|---|---|---|
| | **Testing Hints** | |

[In Test Plan, in existing sections 5.3.30 add four new tests each.]

## 5.3 Alarm and Event Management - Notification - External - B

### 5.3.30 Implements the CHANGE_OF_TIMER Algorithm

~~Contact the BTL for interim tests for this algorithm.~~ The IUT contains, or can be made to contain, an object that can generate EventNotifications with an Event_Type of CHANGE_OF_TIMER.

| **BTL - 8.4.10.X1 - CHANGE_OF_TIMER ConfirmedEventNotification Test** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT. |
| | **Testing Hints** | |
| **BTL - 8.4.10.X2 - CHANGE_OF_TIMER Offnormal-to-Offnormal ConfirmedEventNotification** | | |
| | **Test Conditionality** | If pAlarmValues cannot be configured with two different values to which pMonitored will become equal, this test shall be skipped. |
| | **Test Directives** | This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT. |
| | **Testing Hints** | |
| **BTL - 8.5.10.X1 - CHANGE_OF_TIMER UnconfirmedEventNotification Test** | | |
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT. |
| | **Testing Hints** | |
| **BTL - 8.5.10.X2 - CHANGE_OF_TIMER Offnormal-to-Offnormal UnconfirmedEventNotification** | | |
| | **Test Conditionality** | If pAlarmValues cannot be configured with two different values to which pMonitored will become equal, this test shall be skipped. |
| | **Test Directives** | This test shall be executed with an Event Enrollment object that is configured to monitor a property in a device other than the IUT. |
| | **Testing Hints** | |

[In Test Plan, replace existing placeholder sections 6.9 and 6.10 with the following.]

## 6.9 Scheduling - Timer - Internal - B

### 6.9.1 Base Requirements

There are no Base Requirements for this BIBB.

### 6.9.2 Supports Writable Priority_For_Writing Property

The IUT supports a writable Priority_For_Writing property in Timer objects.

| **Verify Checklist** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify the IUT claims support for Timer object option 'Supports Priority_For_Writing'. |
| | **Testing Hints** | |

### 6.9.3 Supports Writable Default_Timeout Property

The IUT supports a writable Default_Timeout property in Timer objects.

| **Verify Checklist** | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify the IUT claims support for Timer object option 'Supports Default_Timeout'. |
| | **Testing Hints** | |

## 6.10 Scheduling - Timer - External - B

### 6.10.1 Base Requirements

There are no Base Requirements for this BIBB.

### 6.10.2 Supports SCHED-TMR-I-B

The IUT supports SCHED-TMR-I-B.

| Verify Checklist | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify that the IUT claims support for SCHED-TMR-I-B. |
| | **Testing Hints** | |

### 6.10.3 Supports DS-WP-A

The IUT supports DS-WP-A.

| Verify Checklist | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify that the IUT claims support for DS-WP-A. |
| | **Testing Hints** | |

### 6.10.4 Supports Writable List_Of_Object_Property_References

The IUT supports a writable List_Of_Object_Property_References property.

| Verify Checklist | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | Verify the IUT claims support for Timer object option 'Supports writable Priority_For_Writing and List_Of_Object_Property_References' |
| | **Testing Hints** | |

### 6.10.5 Supports External Devices

The IUT supports a non-empty List_Of_Object_Property_References property which can contain references to objects in external devices.

| BTL - 7.3.2.X57.1.9 - Timer supports writing an External Device | | |
|---|---|---|
| | **Test Conditionality** | Must be executed. |
| | **Test Directives** | This test shall be executed with a Timer object that is capable of being configured to monitor a property in a device other than the IUT. |
| | **Testing Hints** | |

[Add these entirely new tests to BTL Specified Tests.]

**7.3.2.X57.1.1 Timer State_Change_Values**

Reason for Change: No test exists for this functionality.

Purpose: Verify all State_Change_Values property transitions.

Test Concept: Start this test in the IDLE state. Write Timer_Running and observe the object enter RUNNING state, and after time passes, observe it enters the EXPIRED state. Force it into the IDLE state. Write Timer_Running again observe it enter RUNNING state, then write Timer_Running again and observe it make the RUNNING_TO_RUNNING transition. Force it into the IDLE state again. Observe that per State_Change_Values transitions, all writes take place.

Configuration Requirements: The State_Change_Values property, if present, holds a valid configuration.

Test Steps:

```
1. WRITE Timer_Running = TRUE
2. CHECK (IUT exhibits any changes configured in IDLE_TO_RUNNING transition)
3. READ RT = Present_Value
4. WAIT RT
5. CHECK (IUT exhibits any changes configured in RUNNING_TO_EXPIRED transition)
6. WRITE Timer_State = IDLE
7. CHECK (IUT exhibits any changes configured in EXPIRED_TO_IDLE transition)
8. WRITE Timer_Running = TRUE
9. CHECK (IUT exhibits any changes configured in IDLE_TO_RUNNING transition)
10. BEFORE the timer expires
        WRITE Timer_Running = TRUE
11. CHECK (IUT exhibits any changes configured in RUNNING_TO_RUNNING transition)
```

```
12. WRITE Timer_State = IDLE
13. CHECK (IUT exhibits any changes configured in RUNNING_TO_IDLE transition)
14. VERIFY Present_Value = 0
```

**7.3.2.X57.1.2 Timer Running then Expired Test**
Reason for Change: No test exists for this functionality.

Purpose: Verify that Timer T1 when set to RUNNING, enters the EXPIRED state after the configured time.

Test Concept: Start this test in the IDLE state. Write Timer_Running and observe it enter RUNNING state, with its current configuration. After time passes, observe it enters the EXPIRED state. Observe that specified properties take their required values.

Configuration Requirements: T1 starts this test with the Timer_State equal to IDLE.

Test Steps:

```
1. VERIFY Timer_State = IDLE
2. WRITE Timer_Running = TRUE
3. CHECK (IUT exhibits any changes configured in IDLE_TO_RUNNING transition)
4. IF (Default_Timeout property is present in T1) THEN {
        READ DV = Default_Timeout
        VERIFY Initial_Timeout = DV
        }
5. VERIFY Timer_State = RUNNING
6. READ RT = Present_Value
7. WAIT RT
8. CHECK (IUT exhibits any changes configured in RUNNING_TO_EXPIRED transition)
9. VERIFY Timer_State = EXPIRED
10. VERIFY Present_Value = 0
11. VERIFY Last_State_Change = RUNNING_TO_EXPIRED
12. IF (Update_Time property is present in T1) THEN {
        READ UT = Update_Time
        VERIFY UT ~= (the current date and time)
        IF (Expiration_Time property is present in T1) THEN
            VERIFY Expiration_Time = UT
    }
    ELSE IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time ~= (the current date and time)
```

**7.3.2.X57.1.3 Default_Timeout Test**
Reason for Change: No test exists for this functionality.

Purpose: Verify that starting the Timer running via Timer_Running uses Default_Timeout.

Test Concept: Start this test in the IDLE state. Default_Timeout is a valid non-zero value, different from the value which is Initial_Timeout at start of test. Write Timer_Running and observe the Timer enter RUNNING state. Observe that specified properties take their required values.

Configuration Requirements: Timer starts this test with the Timer_State equal to IDLE. IUT is configured with Initial_Timeout and Default_Timeout being different. If this cannot be done, then this test shall be skipped.

Test Steps:

```
1. READ PrevIT = Initial_Timeout
2. VERIFY Default_Timeout <> PrevIT
3. WRITE Timer_Running = TRUE
4. VERIFY Present_Value ~= Default_Timeout
```

**7.3.2.X57.1.4 Running Timer by writing the Present_Value**
Reason for Change: No test exists for this functionality.

Purpose: Start or Restart the Timer by writing the Present_Value property.

Test Concept: Start the Timer with a non-zero value PV1, written to the Present_Value property, and observe that the Timer counts down according to the new value written.

Configuration Requirements: This test can start with T1 in any of the IDLE, EXPIRED, or RUNNING states.

Test Steps:

```
1. IF (Default_Timeout property is present in T1) THEN
        READ DT = Default_Timeout
2. READ PrevIT = Initial_Timeout
3. WRITE Present_Value = (PV1, any valid non-zero value, sufficiently different from DT and PrevIT so that it is clear that countdown is according to the new value written)
4. VERIFY Present_Value <= PV1
```

**7.3.2.X57.1.5 Restarting An Expired Timer**
Reason for Change: No test exists for this functionality.

Purpose: Verify that writes to Timer_Running with TRUE while in the EXPIRED state are successful.

Test Concept: Start this test with the Timer_State equal to EXPIRED. Write TRUE to Timer_Running.

Configuration Requirements: T1 starts this test with the Timer_State equal to EXPIRED.

Test Steps:

```
1. VERIFY Timer_State = EXPIRED
2. WRITE Timer_Running = TRUE
3. VERIFY Timer_State = RUNNING
```

**7.3.2.X57.1.6 Already Running Timer restarted by writing the Present_Value**
Reason for Change: No test exists for this functionality.

Purpose: Verify Timer can be restarted while running by writing Present_Value.

Test Concept: Configure and run the Timer T1 as necessary to put it into RUNNING state. Then write the Timer with a different non-zero value written to the Present_Value property and observe that specified properties take their required values and all configured State_Change_Values transitions if any, take place. The timer counts down and observe that it operates according to the new value written.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING. If Present_Value is not writable, this test shall be skipped.

Test Steps:

```
1. VERIFY Timer_State = RUNNING
2. WRITE Present_Value = (any valid non-zero value)
3. CHECK (IUT exhibits any changes configured in RUNNING_TO_RUNNING transition)
4. VERIFY Timer_Running = TRUE
5. VERIFY Last_State_Change = RUNNING_TO_RUNNING
```

**7.3.2.X57.1.7 Already Running Timer restarted with Default_Timeout**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Timer_Running with TRUE while already in the RUNNING state.

Test Concept: Configure and run the Timer T1 as necessary to put it into RUNNING state with an Initial_Value different from Default_Value. Then write the Timer_Running property with TRUE, and observe that Present_Value restarts with the value from Default_Timeout.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING. In service of observing the change between step 3 and step 6, it is necessary that at the test start, the Timer went into RUNNING state with an Initial_Value different from Default_Value.

Test Steps:
1. VERIFY Timer_State = RUNNING
2. READ DV = Default_Timeout
3. VERIFY Initial_Timeout <> DV
4. WRITE Timer_Running = TRUE
5. CHECK (IUT exhibits any changes configured in RUNNING_TO_RUNNING transition)
6. VERIFY Initial_Timeout = DV
7. VERIFY Present_Value ~= DV
8. VERIFY Timer_Running = TRUE
9. VERIFY Last_State_Change = RUNNING_TO_RUNNING

**7.3.2.X57.1.8 Timer accepts all the required datatypes in an Internal Reference**
Reason for Change: No test exists for this functionality.

Purpose: Verify that the IUT with a modifiable List_Of_Object_Property_References, accepts all the required datatypes.

Test Concept: Verify in a Timer object, T1, that supports modification, the IUT allows altering the List_Of_Object_Property_References to refer to an object within the IUT. Repeat for each of the datatypes required for Timers with writable List_Of_Object_Property_References. Also write State_Change_Values or its entries with values of that datatype.

Test Steps:
```
1. REPEAT (for all the required datatypes: values of type NULL, BOOLEAN, Unsigned, INTEGER, REAL, and ENUMERATED {
        WRITE (the State_Change_Values and/or List_Of_Object_Property_References with that datatype, and which references an object within the IUT)
}
2. CHECK (Did the IUT accept the writes, and apply the modified values to properties correctly?)
```

Notes to Tester: It is required that the IUT allows modifying the Timer one property at a time.

**7.3.2.X57.1.9 Timer supports writing an External Device**
Reason for Change: No test exists for this functionality.

Purpose: Verify that IUT allows its List_Of_Object_Property_References to refer to an external device.

Test Concept: Verify in a Timer object T1 that supports modification, the IUT allows altering the List_Of_Object_Property_References to refer to an object in an external device.

Configuration Requirements: If there is no Timer object in IUT which supports reference to an object in an external device, then this test shall be skipped.

Test Steps:
1. WRITE List_Of_Object_Property_References = (a value that is different, and which contains one or more references to objects which are in one or more external devices)
2. VERIFY List_Of_Object_Property_References = (the value written)

**7.3.2.X57.1.10 Forcing Timer Expiration by writing Zero**
Reason for Change: No test exists for this functionality.

Purpose: Interrupt the Timer while it is RUNNING, via a value of zero written to the Present_Value property.

Test Concept: Configure and start the Timer T1 to operate according to its values. Then write a value of zero to the Present_Value property and observe that specified properties take their required values and that the State_Change_Values operations take place.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING.

Test Steps:

```
1. VERIFY Timer_Running = TRUE
2. VERIFY Timer_State = RUNNING
3. WRITE Present_Value = 0
4. CHECK (IUT exhibits any changes configured in FORCED_TO_EXPIRED transition)
5. VERIFY Timer_State = EXPIRED
6. VERIFY Last_State_Change = FORCED_TO_EXPIRED
7. VERIFY Present_Value = 0
8. IF (Update_Time property is present in T1) THEN {
           READ UT = Update_Time
           VERIFY UT ~= (the current date and time)
           IF (Expiration_Time property is present in T1) THEN
                     VERIFY Expiration_Time = UT
   }
   ELSE IF (Expiration_Time property is present in T1) THEN
           VERIFY Expiration_Time ~= (the current date and time)
```

**7.3.2.X57.1.11 Forcing Timer Expiration by writing FALSE**
Reason for Change: No test exists for this functionality.

Purpose: Interrupt the Timer while it is RUNNING, via a value of FALSE written to the Timer_Running property.

Test Concept: Configure and start Timer T1 to operate according to its values. Then write FALSE to Timer_Running and observe that specified properties take their required values and all configured State_Change_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING.

Test Steps:

```
1. VERIFY Timer_Running = TRUE
2. VERIFY Timer_State = RUNNING
3. WRITE Timer_Running = FALSE
4. CHECK (IUT exhibits any changes configured in FORCED_TO_EXPIRED transition)
5. VERIFY Timer_State = EXPIRED
6. VERIFY Last_State_Change = FORCED_TO_EXPIRED
7. IF (Update_Time property is present in T1) THEN {
           READ UT = Update_Time
           VERIFY UT ~= (the current date and time)
           IF (Expiration_Time property is present in T1) THEN
                     VERIFY Expiration_Time = UT
}
   ELSE IF (Expiration_Time property is present in T1) THEN
           VERIFY Expiration_Time ~= (the current date and time)
```

**7.3.2.X57.1.12 Forcing Timer Expiration by writing IDLE**
Reason for Change: No test exists for this functionality.

Purpose: Interrupting the Timer while it is RUNNING, via a value of IDLE written to the Timer_State property.

Test Concept: Configure and start the Timer T1 to operate according to its values. Then write IDLE to Timer_State and observe that specified properties take their required values and all configured State_Change_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING.

Test Steps:

```
1. VERIFY Timer_Running = TRUE
2. VERIFY Timer_State = RUNNING
3. WRITE Timer_State = IDLE
```

```
4. CHECK (IUT exhibits any changes configured in RUNNING_TO_IDLE transition)
5. VERIFY Timer_State = IDLE
6. VERIFY Last_State_Change = RUNNING_TO_IDLE
7. IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time = (the unspecified datetime value)
8. IF (Update_Time property is present in T1) THEN
        VERIFY Update_Time = (the current date and time)
9. VERIFY Present_Value = 0
```

**7.3.2.X57.1.13 Resetting Timer by writing IDLE**
Reason for Change: No test exists for this functionality.

Purpose: Verify the correct behaviors when Timer_State is written from EXPIRED to IDLE value.

Test Concept: Configure and run the Timer as necessary to put it into EXPIRED state. Then write IDLE to Timer_State and observe that specified properties take their required values and all configured State_Change_Values transitions if any, take place.

Configuration Requirements: T1 starts this test with the Timer_State equal to EXPIRED.

Test Steps:

```
1. VERIFY Timer_State = EXPIRED
2. WRITE Timer_State = IDLE
3. CHECK (IUT exhibits any changes configured in EXPIRED_TO_IDLE transition)
4. VERIFY Timer_State = IDLE
4. VERIFY Timer_Running = FALSE
5. VERIFY Last_State_Change = EXPIRED_TO_IDLE
6. IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time = (the unspecified datetime value)
7. IF (Update_Time property is present in T1) THEN
        VERIFY Update_Time = (the current date and time)
8. VERIFY Present_Value = 0
```

**7.3.2.X57.1.14 Timer Object Operation Unaffected by Changes to Local_Time and Local_Date**
Reason for Change: No test exists for this functionality.

Purpose: Verify that Timer expiration is not affected by time changes.

Test Concept: Configure and start the Timer T1 to operate according to its values. Then before the Timer expires, change Local_Date / Local_Time to a NewDate / NewTime in the past or future and observe that the length of time until Timer expiration is not affected, and that expiry still occurs at the time indicated in Present_Value.

Configuration Requirements: T1 starts this test with the Timer_State equal to RUNNING.

Test Steps:

```
1. VERIFY Timer_Running = TRUE
2. VERIFY Timer_State = RUNNING
3. READ PV_beforeTimeChange = Present_Value
4.  MAKE (Local_Date/ Local_Time = NewDate/ NewTime)
4. VERIFY Present_Value ~= PV_beforeTimeChange, continuing its decreasing trend
5. VERIFY Local_Date = NewDate
6. VERIFY Local_Time ~= NewTime
7. WAIT PV_beforeTimeChange
8. VERIFY Present_Value = 0
9. IF (Update_Time property is present in T1) THEN
        VERIFY Update_Time ~= (the current date and time)
10. IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time ~= (the current date and time)
```

Hints to Tester: To ensure that testing would detect an implementation which prematurely expires when the Local_Time becomes a time that the Timer would not be RUNNING, select NewDate / NewTime which when converted to local date/time using UTC_Offset and Daylight_Saving_Status, is earlier or later than the window of time that the Timer would be RUNNING when it started.

**7.3.2.X57.1.15 Changes made by State_Change_Values are at Correct Priority**
Reason for Change: No test exists for this functionality.

Purpose: Verify by changing the Priority_for_Writing property that subsequent State_Change_Values operations use the right Priority.

Test Concept: Write Timer_Running and observe it enter RUNNING state, with its current configuration. After time passes, observe it enters the EXPIRED state. Observe that specified properties take their required values.

Configuration Requirements: Start this test in the IDLE state. O1 P1 is any commandable property amongst the elements of List_Of_Object_Property_References. O1 should contain in its Priority_Array at the index which will change, a value which is different from the values in State_Change_Values, for the operations which will take place, for ease of ensuring that the Timer commanded the change.

Test Steps:
```
1. WRITE Priority_for_Writing = (any valid value, different from what it had)
2. READ ValueItR = State_Change_Values, ARRAY INDEX = IDLE_TO_RUNNING
3. WRITE Timer_Running = TRUE
4. VERIFY Timer_State = RUNNING
5. READ RT = Present_Value
6. IF (ValueItR is a value other than no-value) THEN
       VERIFY (O1), Priority_Array = ValueItR, ARRAY INDEX = Priority_for_Writing
7. WAIT RT
8. READ ValueRtE = State_Change_Values, ARRAY INDEX = RUNNING_TO_EXPIRED
9. VERIFY Timer_State = EXPIRED
10. IF (ValueRtE is a value other than no-value) THEN
       VERIFY (O1), Priority_Array = ValueRtE, ARRAY INDEX = Priority_for_Writing
```

**7.3.2.X57.1.16 Changing Default_Timeout Test**
Reason for Change: No test exists for this functionality.

Purpose: Reconfigure the Default_Timeout and see that governs the length the timer runs.

Test Concept: Start this test in the IDLE state. Configure the Timer with an updated Default_Time and observe it operates according to the new value written.

Configuration Requirements: T1 starts this test with the Timer_State equal to IDLE.

Test Steps:
```
1. READ IT = Initial_Timeout
2. WRITE Default_Timeout = (any valid value, different from IT and different from what it had)
3. WRITE Timer_Running = TRUE
4. VERIFY Present_Value ~= Default_Timeout
```

**7.3.2.X57.2.1 Writing Timer with an Unsupported External Reference**
Reason for Change: No test exists for this functionality.

Purpose: Verify the correct Result(-) when List_Of_Object_Property_References does not support objects in an external device.

Test Concept: Attempt writing List_Of_Object_Property_References of a Timer object T1 which does not support referring to an object in an external device. Verify the IUT returns the correct Result(-).

Configuration Requirements: If the IUT supports referring to an object in an external device in all of its Timer objects, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = T1
   'Property Identifier' = List_Of_Object_Property_References
   'Property Value' = (a value that is different, and which references an object in an external device)
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY
   'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED

**7.3.2.X57.2.2 Writing an Unsupported Datatype to State_Change_Values**
Reason for Change: No test exists for this functionality.

Purpose: Verify the correct Result(-) when State_Change_Values is written with a datatype that instance does not support.

Test Concept: Attempt writing State_Change_Values of a Timer object T1 with a datatype that instance does not support. Verify the IUT returns the correct Result(-).

Configuration Requirements: The State_Change_Values property initially holds a valid configuration.

Test Steps:
1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = T1
   'Property Identifier' = State_Change_Values
   'Property Value' = (a value which contains a datatype that T1 does not support)
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY
   'Error Code' = DATATYPE_NOT_SUPPORTED

**7.3.2.X57.2.3 Invalid Property Writing in a Timer**
Reason for Change: No test exists for this functionality.

Purpose: Verify the correct Result(-) when Timer_State or Present_Value is written with an invalid value.

Test Concept: Attempt writing of a Timer object T1 with a value outside the supported range and not zero being written to the Present_Value property, or a value of other than IDLE written to the Timer_State property. Verify the IUT returns the correct Result(-).

Configuration Requirements: The State_Change_Values property, if present, holds a valid configuration.

Test Steps:
1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = T1
   'Property Identifier' = Present_Value
   'Property Value' = (a value that is outside the supported range and not zero)
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY
   'Error Code' = VALUE_OUT_OF_RANGE
3. TRANSMIT WriteProperty-Request,
   'Object Identifier' = T1
   'Property Identifier' = Timer_State
   'Property Value' = (a value other than IDLE)
4. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY
   'Error Code' = VALUE_OUT_OF_RANGE

**7.3.2.X57.2.4 Expired Timer Ignores Writing Zero**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Present_Value of a Timer with an expiration value while in the EXPIRED state.

Test Concept: With a Timer object in the EXPIRED state, write 0 to Present_Value and verify that the object remains in the EXPIRED state.

Configuration Requirements: The Timer object starts the test with Timer_State equal to EXPIRED.

Test Steps:
1. VERIFY Timer_State = EXPIRED
2. WRITE Present_Value = 0
3. VERIFY Timer_State = EXPIRED

**7.3.2.X57.2.5 Expired Timer Ignores Writing FALSE**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Timer_Running property of a Timer with an expiration value while in the EXPIRED state.

Test Concept: With a Timer object in the EXPIRED state, write FALSE to Timer_Running and verify that the object remains in the EXPIRED state.

Configuration Requirements: The Timer starts this test with Timer_State equal to EXPIRED.

Test Steps:
1. VERIFY Timer_State = EXPIRED
2. WRITE Timer_Running = FALSE
3. VERIFY Timer_State = EXPIRED

**7.3.2.X57.2.6 Idle Timer Ignores Writing Zero**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Present_Value of a Timer with an expiration value while in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write 0 to Present_Value and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer_State equal to IDLE.

Test Steps:
1. VERIFY Timer_State = IDLE
2. WRITE Present_Value = 0
3. VERIFY Timer_State = IDLE

**7.3.2.X57.2.7 Idle Timer Ignores Writing FALSE**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Timer_Running property of a Timer with an expiration value while already in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write FALSE to Timer_Running and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer_State equal to IDLE.

Test Steps:
1. VERIFY Timer_State = IDLE
2. WRITE Timer_Running = FALSE
3. VERIFY Timer_State = IDLE

**7.3.2.X57.2.8 Idle Timer Ignores Writing IDLE**
Reason for Change: No test exists for this functionality.

Purpose: Verify the success of writes to Timer_State of a Timer with the reset value while already in the IDLE state.

Test Concept: With a Timer object in the IDLE state, write IDLE to Timer_State and verify that the object remains in the IDLE state.

Configuration Requirements: The Timer starts this test with Timer_State equal to IDLE.

Test Steps:
1. VERIFY Timer_State = IDLE
2. WRITE Timer_State = IDLE
3. VERIFY Timer_State = IDLE

**7.3.2.X57.2.9 Default_Timeout Written Outside Supported Range**
Reason for Change: No test exists for this functionality.

Purpose: Verify the correct Result(-) when Default_Timeout is written with an invalid value.

Test Concept: Attempt writing Timer object T1 with a value outside the supported range to the Default_Timeout property. Verify the IUT returns the correct Result(-).

Configuration Requirements: If Default_Timeout is not present or is not writable, this test shall be skipped.

Test Steps:
```
1. TRANSMIT WriteProperty-Request,
        'Object Identifier' = T1
        'Property Identifier' = Default_Timeout
        'Property Value' = (a value lower than Min_Pres_Value)
2. RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY
        'Error Code' = VALUE_OUT_OF_RANGE
3. TRANSMIT WriteProperty-Request,
        'Object Identifier' = T1
        'Property Identifier' = Default_Timeout
        'Property Value' = (a value higher than Max_Pres_Value)
4. RECEIVE BACnet-Error-PDU,
        'Error Class' = PROPERTY
        'Error Code' = VALUE_OUT_OF_RANGE
```

**8.4.10.X1 CHANGE_OF_TIMER ConfirmedEventNotification Test**
Reason for Change: New algorithm for Protocol_Revision 17.

Purpose: To verify the correct operation of the CHANGE_OF_TIMER event algorithm. This test applies to objects that support an Event_Type of CHANGE_OF_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed, typically by putting the Timer into operation where it conducts its usual operations, such that pMonitoredValue becomes equal to any of the values contained in pAlarmValues. After pTimeDelay time later in that same value, the object shall enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed such it is no longer equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time later in that same value, the object shall enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The O1 shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of TRUE. The Recipient_List of the configured Notification Class shall contain recipients. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:
1. VERIFY Event_State = NORMAL

```
2. MAKE (pMonitoredValue equal to any of the values contained in pAlarmValues)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =            (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       O1,
            'Time Stamp' =                    (the current local datetime or time or sequence number),
            'Notification Class' =            (the notification class configured for O1),
            'Priority' =                      (the value configured in transition),
            'Event Type' =                    CHANGE_OF_TIMER,
            'Message Text' =                  (optional, any valid message text),
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    NORMAL,
            'To State' =                      OFFNORMAL,
            'Event Values' =                  pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)

5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Status_Flags = {TRUE, FALSE, ?, ?}
7. VERIFY Event_State = OFFNORMAL
8. MAKE (pMonitoredValue <> any of the values contained in pAlarmValues)
9. WAIT (pTimeDelayNormal)
10. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =            (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       O1
            'Time Stamp' =                    (the current local datetime or time or sequence number),
            'Notification Class' =            (the notification class configured for O1),
            'Priority' =                      (the value configured in transition),
            'Event Type' =                    CHANGE_OF_TIMER,
            'Message Text' =                  (optional, any valid message text),
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    OFFNORMAL,
            'To State' =                      NORMAL,
            'Event Values' =                  pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available)
11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY Status_Flags = {FALSE, FALSE, ?, ?}
13. VERIFY Event_State = NORMAL
```

### 8.4.10.X2 CHANGE_OF_TIMER Offnormal-to-Offnormal ConfirmedEventNotification

Reason for Change: New algorithm for Protocol_Revision 17, which specifies there is a transition when the object is already OFFNORMAL, and the new state is also one of the pAlarmValues.

Purpose: To verify the correct operation of the CHANGE_OF_TIMER event algorithm when pUpdateTime changes while the object is already OFFNORMAL, and the new state is also one of the pAlarmValues. This test applies to objects that support an Event_Type of CHANGE_OF_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed multiple times, typically by putting the Timer into operation where it conducts its usual operations, such that at multiple points the pMonitoredValue equal to any of the values contained in pAlarmValues. After the object enters the OFFNORMAL state and transmits a first event notification message, when pMonitoredValue changes but is again equal to any of the values contained in pAlarmValues, it is observed to transmit another event notification message. Finally the pMonitoredValue becomes a value that is not equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time holding at that value, the object shall enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The O1 shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of TRUE. The Recipient_List of the configured Notification Class shall contain recipients. The event-generating object shall be in a NORMAL state at the start of the test. If the pAlarmValues cannot be configured with two different values to which pMonitored will become equal, then this test shall be skipped.

Test Steps:
```
1. VERIFY Event_State = NORMAL
2. MAKE (pMonitoredValue equal to any of the values contained in pAlarmValues)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         O1,
            'Time Stamp' =                      (the current local datetime or time or sequence number),
            'Notification Class' =              (the notification class configured for O1),
            'Priority' =                        (the value configured in transition),
            'Event Type' =                      CHANGE_OF_TIMER,
            'Message Text' =                    (optional, any valid message text),
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      NORMAL,
            'To State' =                        OFFNORMAL,
            'Event Values' =                    pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential
optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime,
if one is available)

5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Status_Flags = {TRUE, FALSE, ?, ?}
7. VERIFY Event_State = OFFNORMAL
8. MAKE (pUpdateTime change, usually by pMonitoredValue becoming a different value, but ensure it is again equal to any
of the values contained in pAlarmValues)
9. BEFORE **Notification Fail Time** -- Note: no pTimeDelay here
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         O1,
            'Time Stamp' =                      (the current local datetime or time or sequence number),
            'Notification Class' =              (the notification class configured for O1),
            'Priority' =                        (the value configured in transition),
            'Event Type' =                      CHANGE_OF_TIMER,
            'Message Text' =                    (optional, any valid message text),
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      OFFNORMAL,
            'To State' =                        OFFNORMAL,
            'Event Values' =                    pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential
optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime,
if one is available)

10. TRANSMIT BACnet-SimpleACK-PDU
11. VERIFY Status_Flags = {TRUE, FALSE, ?, ?}
12. VERIFY Event_State = OFFNORMAL
13. MAKE (pMonitoredValue <> any of the values contained in pAlarmValues)
14. WAIT (pTimeDelayNormal)
15. BEFORE **Notification Fail Time**
```

RECEIVE ConfirmedEventNotification-Request,

| | |
|---|---|
| 'Process Identifier' = | (any valid process ID), |
| 'Initiating Device Identifier' = | IUT, |
| 'Event Object Identifier' = | O1 |
| 'Time Stamp' = | (the current local datetime or time or sequence number), |
| 'Notification Class' = | (the notification class configured for O1), |
| 'Priority' = | (the value configured in transition), |
| 'Event Type' = | CHANGE_OF_TIMER, |
| 'Message Text' = | (optional, any valid message text), |
| 'Notify Type' = | EVENT \| ALARM, |
| 'AckRequired' = | TRUE \| FALSE, |
| 'From State' = | OFFNORMAL, |
| 'To State' = | NORMAL, |
| 'Event Values' = | pMonitoredValue, pStatusFlags, pUpdateTime, (there are also three potential optional notification parameters: pLastStateChange, if one is available, pInitialTimeout, if one is available, pExpirationTime, if one is available) |

16. TRANSMIT BACnet-SimpleACK-PDU
17. VERIFY Status_Flags = {FALSE, FALSE, ?, ?}
18. VERIFY Event_State = NORMAL

**8.5.10.X1 CHANGE_OF_TIMER UnconfirmedEventNotification Test**
Reason for Change: New algorithm for Protocol_Revision 17.

Purpose: To verify the correct operation of the CHANGE_OF_TIMER event algorithm. This test applies to objects that support an Event_Type of CHANGE_OF_TIMER.

Configuration Requirements: The O1 shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of FALSE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.10.X11 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.10.X11 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

**8.5.10.X2 - CHANGE_OF_TIMER Offnormal-to-Offnormal UnconfirmedEventNotification Test**
Reason for Change: New algorithm for Protocol_Revision 17.

Purpose: To verify the correct operation of the CHANGE_OF_TIMER event algorithm. This test applies to objects that support an Event_Type of CHANGE_OF_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed multiple times, typically by putting the Timer into operation where it conducts its usual operations, such that at multiple points the pMonitoredValue is equal to any of the values contained in pAlarmValues. After the object enters the OFFNORMAL state and transmits a first event notification message, when pMonitoredValue changes but is again equal to any of the values contained in pAlarmValues, it is observed to transmit another event notification message. Finally the pMonitoredValue becomes a value that is not equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time holding at that value, the object shall enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The O1 shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter in the Recipient_List of the configured Notification Class shall have a value of FALSE. The Recipient_List of the configured Notification Class shall contain recipients. The event-generating object shall be in a NORMAL state at the start of the test. If the pAlarmValues cannot be configured with two different values to which pMonitored will become equal, then this test shall be skipped.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.10.X2 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.10.X2 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.