BACnet® TESTING LABORATORIES

Revision 15.1.final

SPECIFIED TESTS

Revised June 1, 2018

BACnet is a registered trademark of ASHRAE. ASHRAE does not endorse, approve or test products for compliance with ASHRAE standards. Compliance of listed products to the requirements of ASHRAE Standard 135 is the responsibility of BACnet International (BI). BTL is a registered trademark of BI.

Table of Contents

1. PURPOSE 10
2. Interim Data Link Layer Tests 11
2.2 MS/TP Data Link Layer Tests 11
2.2.18 Verify Tno_token w/ Serial Analyzer 11
2.2.X1 Data Not For Us Test 11
2.3 ARCNET (twisted pair bus) Data Link Layer Tests 12
2.3.1 Verify the Failsafe Biasing with an Oscilloscope 13
2.3.2 Verify the Basic Signal Duty Cycle with an Oscilloscope 13
3. DEFINITIONS 15
3.x Common language used in tests 15
5. EPICS CONSISTENCY TESTS 15
7. OBJECT SUPPORT TESTS 16
7.1.1 Read Support Test Procedure 16
7.1.2 Non-documented Property Test 17
7.1.X3 Verifying Property_List against the EPICS 18
7.2 Write Support for Properties in Test Database 18
7.2.1 Functional Range Requirements for Property Values 18
7.2.1.3 Octetstrings and Characterstrings 18
7.2.2 Write Support Test Procedure 18
7.2.3 Read-only Property Test 19
7.2.X1 Date Pattern Properties Test 21
7.2.X2 Time Pattern Properties Test 21
7.2.X3 DateTime Pattern Properties Test 22
7.2.X4 Date Non-Pattern Properties Test 22
7.2.X5 Time Non-Pattern Properties Test 23
7.2.X6 DateTime Non-Pattern Properties Test 23
7.2.X7 BACnetDateRange Non-Pattern Properties Test 24
7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test 24
7.3 Object Functionality Tests 25
7.3.1 Property Tests 25
7.3.1.6 Minimum On/Off Time Tests 25
7.3.1.6.*1* Override of Minimum Time 25
7.3.1.6.2 Minimum Off Time - Writing at priorities numerically greater than 6 26
7.3.1.6.3 Minimum On Time - Writing at priorities numerically greater than 6 27
7.3.1.6.4 Minimum Off Time - Writing at priorities numerically lesser than 6 28
7.3.1.6.5 Minimum On Time - Writing at priorities numerically lesser than 6 29
7.3.1.6.6 Minimum_Off_Time - Clock is not affected by additional write operations 29
7.3.1.6.7 Minimum_On_Time - Clock is not affected by additional write operations 30
7.3.1.6.8 Ensuring Minimum_Off_Time starts at transition to INACTIVE 31
7.3.1.6.9 Ensuring Minimum_On_Time starts at transition to ACTIVE 32
7.3.1.6.10 Ensuring Minimum Times Are Not Effected By Time Changes 34
7.3.1.7 COV Tests 35
7.3.1.7.X1 COV_Resubscription_Interval Test 35
7.3.1.9 Binary Object Elapsed Active Time Tests 36
7.3.1.10 Event_Enable Tests 38
7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL and TO_NORMAL 38
7.3.1.10.2 Event_Enable Tests for TO_NORMAL only Algorithms 40
7.3.1.11 Acked_Transitions Tests 42
7.3.1.13 Limit_Enable Tests 46
7.3.1.13.X1 Limit_Enable Test, LowLimitEnable 50
7.3.1.13.X2 Limit_Enable Test, HighLimitEnable 51
7.3.1.X4 Event_Message_Texts Tests 52
7.3.1.X5 Event_Message_Texts_Config Test 54
7.3.1.X6 Event_Algorithm_Inhibit Tests 55

7.3.1.X6.1 Event_Algorithm_Inhibit Test 55
7.3.1.X6.2 Event_Algorithm_Inhibit Summarization Test 56
7.3.1.X6.3 Event_Algorithm_Inhibit Acknowledgement Test 57
7.3.1.X7 Event_Algorithm_Inhibit_Ref Tests 58
7.3.1.X7.1 Event_Algorithm_Inhibit_Ref Test 58
7.3.1.X7.2 Event_Algorithm_Inhibit Writable Test 58
7.3.1.X8 Reliability_Evaluation_Inhibit Tests 58
7.3.1.X8.1 Reliability_Evaluation_Inhibit Test 58
7.3.1.X8.2 Reliability_Evaluation_Inhibit Summarization Test 60
7.3.1.X9 Event_Detection_Enable Tests 60
7.3.1.X9.1 Event_Detection_Enable Inhibits Event Generation 60
7.3.1.X9.2 Event_Detection_Enable Inhibits FAULT 61
7.3.2 Object Specific Tests 62
7.3.2.4 Averaging Object Tests 62
7.3.2.4.1 Reinitializing the Samples 62
7.3.2.4.2 Managing the Sample Window 63
7.3.2.9 Command Object Tests 64
7.3.2.9.7 Write While In_Process is TRUE Test. 64
7.3.2.10 Device Object Tests 65
7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test 65
7.3.2.10.6 Successful Increment of the Database_Revision Property after Changing the Object_Identifier Property of an Object 66
7.3.2.10.X2 Max_Segments_Accepted at least the minimum 67
7.3.2.13 Global Group 67
7.3.2.13.X1 Global Group Present_Value, Out_Of_Service and Status_Flags Test 67
7.3.2.13.X2 Reliability MEMBER_FAULT Test 68
7.3.2.13.X3 Reliability COMMUNICATION_FAILURE Test 68
7.3.2.13.X4 Present_Value Tracking and Reliability Test 69
7.3.2.13.X5 Present_Value Tracking Test 69
7.3.2.13.X6 COVU_Period and COVU_Recipient Zero Test 70
7.3.2.21 Notification Class Object Tests 70
7.3.2.21.3 Recipient_List Tests 71
7.3.2.21.3.1 ValidDays Test 71
7.3.2.21.3.2 FromTime and ToTime Test 72
7.3.2.21.3.3 IssueConfirmedNotifications Test 73
7.3.2.21.3.4 Transitions Test 74
7.3.2.21.3.5 Recipient_List Property Supports Device Identifier Recipients Test 76
7.3.2.21.3.6 Recipient_List Property Supports Network Address Recipients 77
7.3.2.21.3.X7 Recipient_List non-volatility test 78
7.3.2.21.3.X8 Read-only Recipient_List with internal Notification Forwarder objects 78
7.3.2.21.3.X9 Read-only Recipient_List for external Notification Forwarder Objects 79
7.3.2.22 Program Object Tests 79
7.3.2.22.1 Program_Change property test 79
7.3.2.23 Schedule Object Tests 80
7.3.2.23.6 Weekly_Schedule Restoration Test 80
7.3.2.23.10 Schedule Object Protocol_Revision 4 Tests 81
7.3.2.23.10.3 Revision 4 Exception_Schedule Property Tests 81
7.3.2.23.10.3.8 Revision 4 Event Priority Test 81
7.3.2.24 Log Object Tests 82
7.3.2.24.4 Log_Interval Test 82
7.3.2.24.13 Log-Status Test 82
7.3.2.24.14 Time_Change Test 84
7.3.2.24.15 COV-Sampling Verification Test 85
7.3.2.24.19 Trigger Verification Test 86
7.3.2.24.X8 Clock-Aligned Logging 87
7.3.2.24.X9 Logging Interval_Offset 87

7.3.2.X37 Accumulator Object Tests 87
7.3.2.X37.1.1 Present_Value Remains In-Range Test.....87
7.3.2.X37.1.3 Logging_Record in Accumulator Test.....88
7.3.2.X37.1.5 Logging_Record in Accumulator STARTING Test 89
7.3.2.X37.1.6 Out_Of_Service Accumulator Test.....89
7.3.2.X37.1.7 Value_Set Writing Test.....90
7.3.2.X37.1.8 Value_Before_Change Writing Test.....90
7.3.2.X38 Pulse Converter Object Tests.....91
7.3.2.X38.1.1 Adjust_Value Write Test 91
7.3.2.X38.1.2 Scale_Factor Test.....91
7.3.2.X38.1.3 Out_Of_Service Pulse Converter Test 91
7.3.2.X38.1.5 Update_Time Reflects Change to the Count and is Updated Atomically Test 92
7.3.2.X38.2.1 Adjust_Value Out-of-Range WriteProperty Test.....93
8. APPLICATION SERVICE INITIATION TESTS 94
8.2 ConfirmedCOVNotification Service Initiation Tests.....94
8.2.1 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value Object* Present_Value Property 94
8.2.2 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object Status_Flags Property 95
8.2.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present_Value Property.....96
8.2.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status_Flags Property.....98
8.2.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~or~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Present_Value Property.....99
8.2.6 Change of Value Notification from a Multi-state Input, Multi-state Output Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Status_Flags Property.....100
8.2.7 Change of Value Notification from Loop Object Present_Value Property 101
8.2.8 Change of Value Notification from a Loop Object Status_Flags Property 103
8.2.X9 ConfirmedCOVNotification Pulse Converter changing Present_Value 104
8.2.X10 ConfirmedCOVNotification Pulse Converter changing Status_Flags 105
8.3 UnconfirmedCOVNotification Service Initiation Tests.....106
8.3.1 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object Present_Value Property 106
8.3.2 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object Status_Flags Property 107
8.3.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Present_Value Property.....107
8.3.6 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Status_Flags Property.....107
8.3.X1 COVU_Recipients Notifications 108
8.3.X11 Unsubscribed COV Service Initiation Test.....108
8.3.X12 UnconfirmedCOVNotification Pulse Converter changing Present_Value 109
8.3.X13 UnconfirmedCOVNotification Pulse Converter changing Status_Flags 109
8.4 ConfirmedEventNotification Service Initiation Tests 109
8.4.X1 DOUBLE_OUT_OF_RANGE Tests (ConfirmedEventNotification) 109
8.4.X2 SIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification) 109
8.4.X3 UNSIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification) 110

8.4.X4 CHANGE_OF_CHARACTERSTRING Tests (ConfirmedEventNotification) 110
8.4.X5 Proprietary Algorithm Tests (ConfirmedEventNotifications) 113
8.4.X6 Extended Algorithm Tests (ConfirmedEventNotifications) 113
8.4.X7 UNSIGNED_RANGE ConfirmedEventNotification Test.....113
8.4.X8 CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification).....115
8.5 UnconfirmedEventNotification Service Initiation Tests.....117
8.5.X1 DOUBLE_OUT_OF_RANGE Tests (UnconfirmedEventNotification) 117
8.5.X2 SIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification).....117
8.5.X3 UNSIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification) 117
8.5.X4 CHANGE_OF_CHARACTERSTRING Tests (UnconfirmedEventNotification) 117
8.5.X5 Proprietary Algorithm Tests (UnconfirmedEventNotifications) 118
8.5.X6 Extended Algorithm Tests (UnconfirmedEventNotifications) 118
8.5.X8 CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification).....119
8.5.X9 CHANGE_OF_RELIABILITY Tests 119
8.5.X9.1 CHANGE_OF_RELIABILITY with No Fault Algorithm 119
8.5.X9.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm ..120
8.5.X9.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm.....122
8.5.X9.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm 123
8.5.X9.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm 124
8.5.X9.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm 125
8.5.X9.7 Event Enrollment Fault Condition Precedence Tests 126
8.5.X9.7.1 Internal Faults Take Precedence Over Monitored Object Faults.....126
8.5.X9.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms.....127
8.5.X9.7.3 Internal Faults Take Precedence Over Fault Algorithms.....127
8.5.X9.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault ...128
8.5.X9.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault.....129
8.5.X9.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL 130
8.5.X9.11 CHANGE_OF_RELIABILITY with First Stage Object Fault 132
8.11 SubscribeCOVProperty Service Initiation Tests.....133
8.11.1 Confirmed Notifications Subscription.....133
8.11.2 Unconfirmed Notifications Subscription.....133
8.11.3 Canceling a Subscription.....134
8.11.X1 Change of Value Notification Tests 134
8.11.X1.1 Change of Value Notification 134
8.11.X1.2 Change of Value Notifications with Invalid Process Identifier 135
8.11.X1.3 Change of Value Notification Arrives after Subscription has Expired 136
8.11.X1.4 Change of Value Notifications with Invalid Monitored Object Identifier 137
8.11.X1.5 Change of Value Notifications with Invalid Monitored property.....137
8.11.X4 Requests 8 Hour Lifetimes 138
8.20 ReadPropertyMultiple Service Initiation Tests 139
8.20.5 Cases In Which ReadProperty Shall Be Used After ReadPropertyMultiple Fails.....139
8.20.5.1 The IUT Determines the TD does not Support the ReadPropertyMultiple Service.....139
8.21 ReadRange Service Initiation Tests 139
8.21.1 Reading Values with no Specified Range 139
8.21.3 Reading a Range of Values by Position 140
8.21.9 Presents Log Records ~~Containing a Specific Datatype~~.....140
8.22 WriteProperty Service Initiation Tests.....140
8.22.X4 Writing Array Properties as a Whole Array 140
8.24 DeviceCommunicationControl Service Initiation Tests 141
8.24.1 Indefinite Duration, Disable, No Password 141
8.24.2 Indefinite Duration, Disable, Password 141
8.24.3 Time Duration, Disable, Password 142
8.24.4 Enable, Password.....142
8.24.5 Enable, No Password.....142
8.24.6 Time Duration, Disable, No Password 142
8.24.7 Time Duration, Disable-Initiation, Password 143

8.27 ReinitializeDevice Service Initiation Tests....143
8.27.2 COLDSTART with a Password....143
8.27.4 WARMSTART with a Password....143
8.32 Who-Has Service Initiation Tests....144
8.32.3 Object Identifier Selection with a Device Instance Range....144
8.32.4 Object Name Selection with a Device Instance Range....144
8.34 Who-Is Service Initiation Tests....144
8.34.2 Who-Is Request with a Device Instance Range....144
9. APPLICATION SERVICE EXECUTION TESTS....145
9.1 AcknowledgeAlarm Service Execution Tests....145
9.1.1 Positive AcknowledgeAlarm Service Execution Tests....145
9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter....145
9.1.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications using the Sequence Number Form of the 'Time of Acknowledgment' Parameter....148
9.1.1.3 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter....148
9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter....149
9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter....151
9.1.1.6 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter....151
9.1.1.8 Successful Alarm Acknowledgment of Confirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter....152
9.1.1.9 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter....154
9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications....155
9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications....158
9.1.1.X3 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is either High-Limit or Low-Limit, Revision 5 and higher only....160
9.1.2 Negative AcknowledgeAlarm Service Execution Tests....160
9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old....161
9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event Object Identifier' is Invalid....163
9.1.2.4 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event State Acknowledged' is Invalid....164
9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old....164
9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the Referenced Object Does Not Exist....167
9.1.2.7 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event State Acknowledged' is Invalid....167
9.1.X1 Unsupported Message Text Character Set AcknowledgeAlarm Test....168
9.2 ConfirmedCOVNotification Service Execution Tests....169
9.2.1 Positive ConfirmedCOVNotification Service Execution Tests....169
9.2.1.X4 Change of Value Notification from Proprietary Objects....169
9.2.2 Negative ConfirmedCOVNotification Service Execution Tests....169
9.2.2.1 Change of Value Notification Arrives after Subscription has Expired....169
9.2.2.2 Change of Value Notifications with Invalid Process Identifier....170
9.2.2.4 Change of Value Notifications with Invalid Monitored Object Identifier....171
9.3 UnconfirmedCOVNotification Service Execution Tests....171
9.3.X9 Change of Value Notification from Proprietary Objects....171
9.4 ConfirmedEventNotification Service Execution Tests....171
9.4.5 ConfirmedEventNotification Simple Presentation....171

9.4.6 ConfirmedEventNotification Full Presentation 172
9.4.X1 Unsupported Message Text Character Set ConfirmedEventNotificationTest 173
9.5 UnconfirmedEventNotification Service Execution Tests 173
9.5.X1 Unsupported Message Text Character Set UnconfirmedEventNotificationTest 173
9.7 GetEnrollmentSummary Service Execution Tests 174
9.7.1 Required GetEnrollmentSummary Filters 174
9.7.1.1 Enrollment Summary with Zero Summaries 174
9.7.2 User Selectable GetEnrollmentSummary Filters 174
9.7.2.3 Event Type Filter 174
9.8 GetEventInformation Service Execution Tests 175
9.8.6 Chaining Test 175
9.10 SubscribeCOV Service Execution Tests 176
9.10.1 Positive SubscribeCOV Service Execution Tests 176
9.10.1.7 Finite Lifetime Subscriptions 176
9.10.1.X1 Ensuring 5 Concurrent COV Subscribers 177
9.10.2 Negative SubscribeCOV Service Execution Tests 179
9.10.2.1 The Monitored Object Does Not Support COV Notification 179
Reason For Change: Added configuration requirements. 179
9.10.2.X1 The Monitored Object Does Not Exist 179
9.10.2.X2 There Is No Space For A Subscription 180
9.10.2.X3 The Lifetime Parameter is Out of Range 180
9.10.3 Positive Unsubscribed COVNotification Execution Tests 181
9.10.3.X1 Unsubscribed COVNotification Execution Test 181
9.14 AddListElement Service Execution Tests 182
9.14.2 Negative AddListElement Service Execution Tests 182
9.14.2.2 Adding a List Element With an Invalid Datatype 182
9.14.2.3 An AddListElement Failure Part Way Through a List 182
9.15 RemoveListElement Service Execution Tests 183
9.15.2 Negative RemoveListElement Service Execution Tests 183
9.15.2.2 A RemoveListElement Failure Part Way Through a List 183
9.16 CreateObject Service Execution Tests 184
9.16.1 Positive CreateObject Service Execution Tests 184
9.16.1.2 Creating Objects by Specifying the Object Identifier with No Initial Values 184
9.16.1.4 Creating Objects by Specifying the Object Identifier and Providing Initial Values 184
9.16.2 Negative CreateObject Service Execution Tests 184
9.16.2.1 Attempting to Create an Object That Does Not Have a Unique Object Identifier 184
9.16.2.4 Attempting to Create an Object with an Object Type Specifier and an Error in the Initial Values 185
9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values 186
9.16.2.6 Attempting to Create an Object with an instance of 4194303 187
9.16.2.X1 Attempting to Create a non-Supported Object Type (by Object Type) 187
9.16.2.X2 Attempting to Create a non-Supported Object Type (by Object Identifier) 188
9.17 DeleteObject Service Execution Tests 188
9.17.2 Negative DeleteObject Service Execution Tests 188
9.17.2.1 Attempting to Delete an Object That is Not Deletable 188
9.18 ReadProperty Service Execution Tests 189
9.18.1 Positive ReadProperty Service Execution Tests 189
9.18.1.2 Reading a Single Element of an Array 189
9.18.1.X1 Reading Properties Based on Data Type 189
9.18.1.X3 Respects max-segments-accepted bit pattern 189
9.20 ReadPropertyMultiple Service Execution Tests 190
9.20.1 Positive ReadPropertyMultiple Service Execution Tests 190
9.20.1.1 Reading a Single Property from a Single Object 190
9.20.1.2 Reading Multiple properties from a Single Object 190
9.20.1.3 Reading a Single Property from Multiple Objects 191

9.20.1.4 Reading Multiple Properties from Multiple Objects....191
9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error....192
9.20.1.6 Reading Multiple Properties with Multiple Embedded Access Errors....192
9.20.1.7 Reading ALL Properties....193
9.20.1.8 Reading OPTIONAL Properties....193
9.20.1.9 Reading REQUIRED Properties....194
9.20.1.X1 Reading Properties Based on Data Type....194
9.21 ReadRange Service Execution Tests....194
9.21.1 Positive ReadRange Service Execution Tests....194
9.21.1.X1 ReadRange Support for All List Properties....194
9.21.2 Negative ReadRange Service Execution Tests....195
9.21.2.1 Attempting to Read a Property That Does not Exist....195
9.21.2.2 Attempting to Read a Property That is not a List....196
9.21.2.3 Attempting to Read a non-Array Property with an Array Index....196
9.22 WriteProperty Service Execution Tests....196
9.22.1 Positive WriteProperty Service Execution Tests....196
9.22.1.1 Writing a Single Element of an Array....196
9.22.1.2 Writing a Commandable Property Without a Priority....197
9.22.1.3 Writing a Non-Commandable Property with a Priority....197
9.22.1.X1 Writing an Array Size....198
9.22.1.X2 Writing to Properties Based on Data Type....198
9.22.2 Negative WriteProperty Service Execution Tests....199
9.22.2.1 Writing Non-Array Properties with an Array Index....199
9.22.2.2 Writing Array Properties with an Array Index that is Out of Range....199
9.22.2.3 Writing with a Property Value Having the Wrong Datatype....200
9.22.2.4 Writing with a Property Value that is Out of Range....200
9.22.2.X1 Writing Non-Array Read-only Property with an Array Index....201
9.23 WritePropertyMultiple Service Execution Tests....201
9.23.1 Positive WritePropertyMultiple Service Execution Tests....201
9.23.1.1 Writing a Single Property to a Single Object....201
9.23.1.2 Writing Multiple properties to a Single Object....202
9.23.1.3 Writing a Single Property to Multiple Objects....202
9.23.1.4 Writing Multiple Properties to Multiple Objects....203
9.23.1.X4 Writing an Array Size....204
9.23.2 Negative WritePropertyMultiple Service Execution Tests....204
9.23.2.1 Writing Multiple Properties with a Property Access Error....204
9.23.2.2 Writing Multiple Properties with an Object Access Error....204
9.23.2.3 Writing Multiple Properties with a Write Access Error....205
9.23.2.4 Writing Non-Array Properties with an Array Index....206
9.23.2.5 Writing Array Properties with an Array Index that is Out of Range....206
9.23.2.6 Writing with a Property Value Having the Wrong Datatype....207
9.23.2.7 Writing with a Property Value that is Out of Range....207
9.23.2.X1 WritePropertyMultiple Reject Test....208
9.24 DeviceCommunicationControl Service Execution Test....209
9.24.1 Positive DeviceCommunicationControl Service Execution Tests....209
9.24.1.5 Finite Time Duration Restored by ReinitializeDevice....209
9.24.2 Negative DeviceCommunicationControl Service Execution Tests....209
9.24.2.3 Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'....209
9.27 ReinitializeDevice Service Execution Tests....210
9.27.2 Negative ReinitializeDevice Service Execution Tests....210
9.27.2.3 COLDSTART with Missing *or Invalid* Password....210
9.27.2.4 WARMSTART with Missing *or Invalid* Password....211
9.29 UnconfirmedTextMessage Service Execution Tests....212
9.29.1 UnconfirmedTextMessage With No Message Class....212
9.29.2 UnconfirmedTextMessage With an Unsigned Message Class....212
9.29.3 UnconfirmedTextMessage With a CharacterString Message Class....213

9.30 TimeSynchronization Service Execution Tests....213
9.30.1 Positive TimeSynchronization Service Execution Tests....213
9.30.1.1 TimeSynchronization Local Broadcast....213
9.30.1.2 TimeSynchronization Directed to the IUT....214
9.31 UTCTimeSynchronization Service Execution Tests....214
9.31.1 Positive UTCTimeSynchronization Service Execution Tests....215
9.31.1.1 UTCTimeSynchronization Local Broadcast....215
9.31.1.2 UTCTimeSynchronization Directed to the IUT....216
9.32 Who-Has Service Execution Tests....216
9.32.1 Execution of Who-Has Service Requests Originating from the Local Network....216
9.32.1.1 Object ID Version with No Device Range....216
9.32.1.2 Object Name Version with no Device Range....217
9.32.1.3 Object ID Version with IUT Inside of the Device Range....217
9.32.1.4 Object ID Version with IUT Outside of the Device Range....218
9.32.1.5 Object Name Version with IUT Inside of the Device Range....218
9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range....218
9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range....219
9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range....219
9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range....220
9.32.1.11 Object Name Version, Directed to a Specific MAC Address....220
9.32.2 Execution of Who-Has Service Requests Originating from a Remote Network....221
9.32.2.1 Object ID Version, Global Broadcast from a Remote Network....221
9.32.2.2 Object ID Version, Remote Broadcast....221
9.32.2.X3 - Who-Has for Non-existent Object_Name....222
9.32.2.X5 Who-Has for Non-existent Object_Identifier....222
9.33 Who-Is Service Execution Tests....223
9.33.1 Execution of Who-Is Service Requests Originating from the Local Network....223
9.33.1.3 Local Broadcast, Specific Device Inquiry with IUT Outside of the Device Range....223
10. NETWORK LAYER PROTOCOL TESTS....223
10.1 Processing Application Layer Messages Originating from Remote Networks....223
10.2 Router Functionality Tests....224
10.2.2 Processing Network Layer Messages....224
10.2.2.7.2 Unknown Network Layer Message Type....224
10.2.X1 Initiates Network-Number-Is on Startup....224
10.2.X2 Routers Execute What-Is-Network-Number....225
10.6 Non-Router Functionality Tests....225
10.6.3 Ignore Router Commands....225
10.7 Router Functionality....226
10.7.2 Router Binding via Application Layer Services....226
10.8 Virtual Routing Functionality Tests....227
10.8.3 Routing of Unicast APDUs....229
10.8.3.1 Route Request Message from a Local Device to a Virtual Device and Route Response Message from the Virtual Device to the Local Device....229
10.8.3.2 Route Request Message from a Virtual Device to a Local Device....230
10.8.3.5 Unicast Messages That Should Not Be Routed....230
10.8.3.5.1 Unknown Network....230
10.8.4 Routing of Broadcast APDUs to Virtual Devices....231
10.8.4.7 Route Remote Broadcast Message from a Virtual Device to a Local Network....231
10.8.7 Multiple Devices on a Single Virtual Network....231
10.8.7.4 Who-Is Specifying Unknown Device Ids....231
10.8.7.5 Who-Has Specifying Unknown Device Ids....232
12. DATA LINK LAYER PROTOCOLS TESTS....232

12.1 MS/TP State Machine Tests 232
12.1.3 MS/TP Data Link Layer Tests (Alternate) 232
12.1.3.3 Verify T_{frame_gap} 232
13. SPECIAL FUNCTIONALITY TESTS 233
13.1 Segmentation 233
13.1.12.1 IUT Does Not Support Segmented Response 233
13.8 Backup and Restore Procedure Tests 233
13.8.1 Backup and Restore Execution Tests 233
13.8.1.1 Execution of Full Backup and Restore Procedure 233
13.8.1.6 Ending Backup and Restore Procedures via Timeout 236
13.8.1.8 Attempting a Backup Procedure with an Invalid Password 237
13.8.1.9 Attempting a Restore Procedure with an Invalid Password 238
13.8.1.11 Starting and Ending a Restore Procedure when a Password is not Required 239
13.8.2 Backup and Restore Initiation Tests 239
13.8.2.1 Initiate a Full Backup and Restore 239
13.X12.1 Reading with maximum-segments-accepted bit pattern B'000' 241
14. BACnet/IP FUNCTIONALITY TESTS 241
14.1 Non-BBMD B/IP Device 241
14.1.7 Forwarded-NPDU (One-hop Distribution) 241
14.1.8 Original-Broadcast-NPDU 242
14.1.10 Forwarded-NPDU (Two-hop Distribution) 242
14.2 BBMD B/IP Device with a Server Application 243
14.2.1 Execute Forwarded-NPDU 243
14.2.1.1 Execute Forwarded-NPDU (One-hop Distribution) 243
14.2.1.2 Execute Forwarded-NPDU (Two-hop Distribution) 243
14.2.2 Execute Original-Broadcast-NPDU 244
14.2.2.1 Execute Original-Broadcast-NPDU (One-hop Distribution) 244
14.2.2.2 Execute Original-Broadcast-NPDU (Two-hop Distribution) 245
14.3.3 Verify Broadcast Distribution Table Created from the Configuration Saved During the Previous Session 246
14.7 Broadcast management (BBMD, Foreign Devices, Local Application) 247
14.7.1 Broadcast Message from Directly Connected IP Subnet 247
14.7.1.1 Broadcast Message from Directly Connected IP Subnet (One-hop Distribution) 247
14.7.1.2 Broadcast Message from Directly Connected IP Subnet (Two-hop Distribution) 248
14.7.2 Broadcast Message Forwarded by a Peer BBMD 249
14.7.2.1 Broadcast Message Forwarded by a Peer BBMD (One-hop Distribution) 249
14.7.2.2 Broadcast Message Forwarded by a Peer BBMD (Two-hop Distribution) 250
14.7.3 Broadcast Message from a Foreign Device 251
14.7.3.1 Broadcast Message From a Foreign Device (One-hop Distribution) 251
14.7.3.2 Broadcast Message From a Foreign Device (Two-hop Distribution) 252

1. PURPOSE

This document contains tests defined by the BTL that are not included in ANSI/ASHRAE Standard 135.1-2013 or are modified versions of tests in 135.1. These tests are used by the BTL testing process and are referenced by the BTL Test Plan document.

Most of the tests defined in this document will be submitted to SSPC 135. Those that are submitted will be removed from future versions of this document as they are accepted/rejected by the SSPC 135 and 135.1 is updated.

Some of the tests are interim tests defined by the BTL because the test tools are not adequate for testing the particular functionality. These tests will be removed once the tests in SSPC 135.1 can be implemented by the BTL. Examples of such tests are the MS/TP tests.

For those tests that will be submitted to the SSPC 135, the test numbering is based on the numbers that the test would have if they were included in 135.1.

2. Interim Data Link Layer Tests

2.2 MS/TP Data Link Layer Tests

2.2.18 Verify Tno_token w/ Serial Analyzer

Reason for Change: No test exists for this functionality.

Purpose: Verify that the IUT waits at least 500 before declaration of loss of token and start behaving as sole master

Test Concept: A network of two reference masters and IUT is constructed and all are turned on Once the network achieves normal network operation, make one reference master (A) to send a Confirmed Request (Read Property or Read Property Multiple) to the other reference master (B). B is powered off or removed from the network before sending the reply. The network is monitored to verify that the IUT (C) does not take token in hand within 500 milliseconds.

Setup: The test starts with an MS/TP network comprised of two reference master devices and IUT that has achieved normal network operation. Normal network operation should be verified using a serial analyzer. If the IUT does not autobaud, then it shall be configured with the same baud rate of the operating network. The IUT shall be configured with a valid MAC address (0-127) which is not in use by any of the other devices on the network and is less than the Max_Master value in use by the reference masters. The IUT shall be configured with the same Max_Master in use by the reference masters.

Test Steps:

1. VERIFY two reference masters (A & B) and IUT (C) achieved normal network operation
2. MAKE one reference master device (A) to send Confirmed request, either Read Property or Read Property Multiple to other reference master device (B).
3. Power Off or remove the reference Master B from the network before sending the reply.
4. CHECK (verify with the serial analyzer that IUT does not take token in hand and start passing Poll For Master or pass token within 500 millisecond)
5. If the IUT does exhibit the behavior described in step4, fail the IUT.

2.2.X1 Data Not For Us Test

Reason for Change: Addendum 135-2008z.3 Modify MS/TP State Machine to Ignore Data Not For Us.

Purpose: Verify that the IUT properly skips the complete data portion of frames not intended for the IUT.

Test Concept: Send a BACnet Data Not Expecting Reply frame that contains the frame pre-amble octet sequence to an address other that the IUT. Follow it immediately with a ReadProperty request for the IUT's device object to ensure that the IUT will correctly receive and process the ReadProperty request.

Test Steps:

1. TRANSMIT
   Frame Type = BACnet Data Not Expecting Reply
   Destination Address = (any Unicast address other than IUT),
   Length = 7,
   Data = (55 FF 05 FF 00 01 F5)
2. TRANSMIT ReadProperty-Request
   'Object Identifier' = (device, 4194303),
   'Property Identifier' = Object_Name
3. RECEIVE ReadProperty-Response
   'Object Identifier' = (device, IUT),

'Property Identifier' = Object_Name,
'Value' = (any valid value)

2.3 ARCNET (twisted pair bus) Data Link Layer Tests

The ARCNET twisted pair bus is an alternate configuration of the standard ARCNET coax, and therefore, requires a different setup of electronics and chipset configuration. These tests verify that the setup and configuration has been followed in order to provide interoperability.

Since the TD is installed on the non-ARCNET side of a reference router, these tests do not cover strict conformance to the ARCNET data link layer. The methodology is to install the IUT on an ARCNET network containing reference devices that are known to conform to BACnet clause 8 using the twisted pair bus option and verify that the TD can exchange data with the IUT. An oscilloscope will also be employed on the ARCNET network to verify that the IUT meets the duty cycle and biasing requirements of the alternate ARCNET data link layer, as these items are critical to interoperability of ARCNET twisted pair bus. An ARCNET packet sniffer is useful, but not required.

These tests require the use of a reference ARCNET twisted pair bus router and a reference ARCNET twisted pair bus device. These devices will be selected from a pool of qualified devices that are to be submitted by members of the BMA. The tester is free to select any of the qualified references devices to use during the test, and the identity of the reference devices will not be published. The criteria for qualifying the reference devices is virtually identical to the test plans referenced here, with the addition of a few tests for proper formation of the NPCI by the reference router.

General Test Setup:

Install a reference ARCNET twisted pair bus router at ARCNET node address <A>.
Install a reference ARCNET twisted pair bus device at node address <C>.
Install the IUT at node address <B>.

The ARCNET node addresses are not critical, but must be unique and not zero.

Recommended Test Tools:

ARCNET packet sniffer = Any ARCNET packet sniffer that meets the following requirements:
1. Each packet is time stamped with 1msec accuracy.
2. The packet sniffer can support the baud rates being tested.
3. Captured data can be saved and reloaded, including the time stamp information.
4. The packet sniffer is currently available for purchase.

Other desirable traits:
5. A BACnet aware packet sniffer to allow ARCNET packets to be decoded, either online or offline.
6. Export a captured session to a text file, including time stamp information. (This would provide the ability for advanced analysis of the data, such as scanning the data for timing anomalies).

Oscilloscope = Agilent 54620 series or similar. This scope has a 2MB sample memory, which is useful for capturing data for an extended time and then zooming in on the details after the capture is complete. It can also "layer" the samples using 32 levels of display intensity, which makes it easier to spot timing anomalies.

2.3.1 Verify the Failsafe Biasing with an Oscilloscope

Reason for Change: No test exists for this functionality. This test is included in CLB-015.

Test Concept: Verify that failsafe biasing (see ARCNET specification 11.4.2, Fail-safe Bias) is at least 200mV. A maximum value is not specified, but the biasing should be such as to not excessively load the EIA-485 transceivers.
Setup: Run the IUT only on the ARCNET twisted pair bus network with proper termination.
Procedure:
1. Apply power to the IUT, and wait for the ARCNET twisted pair bus device to begin passing tokens.

2. With an oscilloscope probe connected across the bus with the correct polarity, measure the Fail-Safe Bias.

3. Fail the IUT if the Fail-Safe Bias is not at least 200mV.

2.3.2 Verify the Basic Signal Duty Cycle with an Oscilloscope

Reason for Change: No test exists for this functionality. This test is included in CLB-015.

Test Concept: The ARCNET chipset has the option of supporting either coax (normal) or twisted pair (differential driver). The differential driver mode utilizes a 50% duty cycle, while the normal method uses a 25% duty cycle for the bits on the wire. Verify that the duty cycle is 50% for ARCNET twisted pair bus.
Setup: Run the IUT only on the ARCNET twisted pair bus network with proper termination.
Procedure:
1. Apply power to the IUT, and wait for the ARCNET twisted pair bus device to begin passing tokens.
2. With an oscilloscope probe connected across the bus with the correct polarity, measure the bit signal duty cycle (pulse width divided by the interpulse period).

3. Fail the IUT if the duty cycle is not 50% (with allowance for acceptable jitter).

3. DEFINITIONS

3.x Common language used in tests

'any valid value' - Any valid value refers to any value of the correct data type and within the vendor's range specified for the property this is applied to.
'any appropriate password' – Any password that meets the Configuration Requirements specified in the test or test section. Passwords when required by the vendor are required to be no more than 20 characters.

5. EPICS CONSISTENCY TESTS

Reason for Change: Improved the language in this set of tests to clarify the exact requirement of the test.

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 ~~Application_Services_Supported~~*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.

7. OBJECT SUPPORT TESTS

7.1.1 Read Support Test Procedure

Reason for Change: This test does not consider the IUT behavior in cases where a property either can not be read by ReadProperty, and ReadPropertyMultiple services or whose response may be too long to return in the given APDU and segment limitations of the IUT.

Purpose: To verify that all properties of all objects can be read using ReadProperty and ReadPropertyMultiple services.

Test Concept: The test is performed once using ReadProperty and once using ReadPropertyMultiple. When verifying array properties, the whole array shall be read without using an array index, where possible.

Test Steps:

```
1.  REPEAT X = (all objects in the IUT's database) DO {
        REPEAT Y = (all properties in object X) DO {
            IF (Y = property indicated as not accessible by ReadProperty Services) THEN
                TRANSMIT ReadProperty-Request,
                    'Object Identifier' = X,
                    'Property Identifier' = Y
            IF (Protocol_Revision >= 13) THEN
                RECEIVE BACnet-Error PDU,
                    'Error Class' = PROPERTY,
                    'Error Code' = READ_ACCESS_DENIED
            ELSE
                RECEIVE BACnet-Error PDU,
                    'Error Class' = OBJECT | PROPERTY,
                    'Error Code' = (any of the error codes for an OBJECT or PROPERTY class)
        ELSE IF (Y = any property of type ARRAY and is too long to return given the
                APDU and segmentation limitations of the IUT) THEN
            TRANSMIT ReadProperty-Request,
                'Object Identifier' = X,
```

```
            'Property Identifier' = Y
        RECEIVE BACnet-Abort-PDU,
            'Server' = TRUE,
            'Abort Reason' = SEGMENTATION_NOT_SUPPORTED |
                             BUFFER_OVERFLOW
        TRANSMIT ReadProperty-Request,
            'Object Identifier' = X,
            'Property Identifier' = Y,
            'Property Array Index' = 0
        RECEIVE ReadProperty-ACK,
            'Object Identifier' = X,
            'PropertyIdentifier' = Y,
            'Array Index' = 0,
            'Property Value' = (any value specified in the EPICS, P)
        REPEAT Z = (each index 1 through P of the property Y) DO {
            VERIFY (X), Y = (the value for index Z of this property Y in
                                the EPICS), ARRAY INDEX = Z
        }
    ELSE
        VERIFY (X), Y = (the value for this property specified in the EPICS)
        }
    }
```

Notes to Tester: For cases where the EPICS indicates that the value of a property is unspecified using the "?" symbol, any value that is of the correct datatype shall be considered to be a match. *When using the ReadPropertyMultiple service, a received ReadPropertyMultiple-ACK containing the specified Error Class and Error Code shall also be considered a Passing result.*

Passing Result: Trying to read the Log_Buffer property of a Trend Log object by using BACnet ReadProperty and ReadPropertyMultiple services may result in an Error-PDU with an error class of OBJECT or PROPERTY and an error code of OTHER. Note, however, that while neither ASHRAE 135-2001 nor ASHRAE 135-2004 clearly define whether OTHER represents a valid error code in this case, Addendum *u* to ANSI/ASHRAE 135-2008 clearly defined READ_ACCESS_DENIED as the valid error code in this case.

7.1.2 Non-documented Property Test

Reason for Change: Revised test to exclude special property identifiers.

Purpose: To verify that all properties contained in every object are documented in the EPICS.

Test Concept: For each object in the EPICS database, attempt to read each standard property that the EPICS does not document as being part of the object.

Test Steps:

```
1.  REPEAT X = (a tester selected set of objects) DO {
        REPEAT Y = (0 through 511 except 8 (all), 80 (optional) and 105 (required)) DO {
            IF (the property Y is not in the EPICS for object X) THEN
                TRANSMIT ReadProperty-Request,
                    'Object Identifier' =     X,
                    'Property Identifier' =  Y
                RECEIVE BACnet-Error-PDU,
                    Error Class = PROPERTY,
                    Error Code = UNKNOWN_PROPERTY
        }
    }
```

Notes to Tester: The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each variant of that object type should be selected.

7.1.X3 Verifying Property_List against the EPICS

Reason for Change: Addendum 135-2010ao-5.

Purpose: To verify the correct content of the Property_List using the properties in each object as claimed in the EPICS.

Test Concept: Match the properties in each object as claimed in the EPICS, against the content of each object's Property_List.

Test Conditionality: If Protocol_Revision is not present, or Protocol_Revision < 14, then this test shall be skipped.

Test Steps:

1. READ OL = Object_List
2. REPEAT (O1, each object in the content of OL)
3. READ PL = Property_List, in the selected object instance O1
4. CHECK (that the property identifiers in the EPICS for O1 and those in the Property_List property match, except as specified in Notes to Tester)

Notes to Tester: Object_Name (77), Object_Type (79), Object_Identifier (75), and Property_List (371) will appear in the EPICS, but shall not appear in the Property_List value. Any proprietary properties that are supported for the object-type shall be in the Property_List. (see BACnet 15.7.3.1.2). The order in which property identifiers appear in the EPICS, is not required to match the order that they appear in the Property_List value.

7.2 Write Support for Properties in Test Database

7.2.1 Functional Range Requirements for Property Values

7.2.1.3 Octetstrings and Characterstrings

Reason for Change: The description here did not account for the Object_Name property which must be of minimum length of 1 not zero. Not in any SSPC proposal. Addendum 135-2008k-1 Add Support for UTF-8.

Properties with an octetstring or characterstring datatype shall be tested with a string of ~~length zero~~*the minimum supported length*, a string with the maximum supported length, and a string with some length between the two. The vendor shall provide the actual value of the maximum length string in the EPICS. See 4.4.2.

When testing character string properties in a device that supports UTF-8 (Protocol_Revision >= 10), at least one of the data values shall contain multi-byte characters.

7.2.2 Write Support Test Procedure

Reason for Change: 'Notes to Tester' is missing from the version in 135.1-2013.

Purpose: To verify that all writable properties of all objects can be written to using BACnet WriteProperty and WritePropertyMulitiple services. The test is performed once using WriteProperty and once using WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.

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

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {

```
REPEAT Y = (all writable properties in object X) DO {
    REPEAT Z = (all values meeting the functional range requirements of 7.2.1, and any additional
            restrictions placed on the allowable property values by the vendor) DO {
        WRITE (X), Y = Z,
        VERIFY (X), Y = Z
        }
    }
}
```

Notes to Tester: An internal process may set the Present_Value of some properties back to the default value after a successful write, as in the case of a momentary pushbutton, or the Record_Count property. For properties that exhibit this type of behavior, skip the VERIFY step.

Notes to Tester: When a property is currently not writable, the IUT shall return an Error-PDU with 'Error Class' = PROPERTY and 'Error Code' = WRITE_ACCESS_DENIED.

7.2.3 Read-only Property Test

Reason for Change: This test is based on 135.1-2013 and corrects the use of the READ statement. Added 'Configuration Requirements'.

Purpose: To verify that properties marked as read-only in the EPICS are in fact read-only.
Test Concept: To each read-only (not writable and not conditionally writable) property in the EPICS, write the value of the property as read from the device and verify that an error is returned. Write another value that is within the acceptable range for the datatype and verify that an error is returned. If the property is a list and the IUT supports AddListElement, attempt to modify the property with AddListElement and verify that an error is returned.

Configuration Requirements: if the IUT does not support the WriteProperty service, then this test shall be skipped.

Test Steps:

```
1.  REPEAT X = (a tester selected set of objects) DO {
        REPEAT Y = (all read-only properties in object X) DO {
            IF (the property is not an array) THEN
                READ Z = X
                READ Z = (X), property Y
                TRANSMIT WriteProperty-Request,
                    'Object Identifier' =           X,
                    'Property Identifier' =         Y,
                    'Property Value' =              Z
                RECEIVE BACnet-Error-PDU,
                    Error Class =PROPERTY,
                    Error Code =WRITE_ACCESS_DENIED

                TRANSMIT WriteProperty-Request,
                    'Object Identifier' =       X,
                    'Property Identifier' =     Y,
                    'Property Value' =          (any value meeting the range requirements of 7.2.1 except Z)
                RECEIVE BACnet-Error-PDU,
                    Error Class =           PROPERTY,
                    Error Code =            WRITE_ACCESS_DENIED

                IF (the IUT supports AddListElement and the property is a list) THEN
                    TRANSMIT AddListElement-Request,
                        'Object Identifier' =     X,
                        'Property Identifier' =   Y,
```

```
                'List of Elements' =    (any elements value meeting the range requirements of 7.2.1 excluding
                                        those in Z)
            RECEIVE BACnet-Error-PDU,
                Error Class =       PROPERTY,
                Error Code =        WRITE_ACCESS_DENIED
        ELSE
        READ LEN = X, Array_Index = 0
            READ LEN = (X), Y, Array_Index =0
            IF (LEN > 0)
            READ Z = X, Array Index = 1
            READ Z = (X), Y, Array_Index=1
            TRANSMIT WriteProperty-Request,
                'Object Identifier' =               X,
                'Property Identifier' =             Y,
                'Property Value' =      Z,
                'Array Index' =     1
            RECEIVE BACnet-Error-PDU,
                Error Class =PROPERTY,
                Error Code =WRITE_ACCESS_DENIED

            TRANSMIT WriteProperty-Request,
                'Object Identifier' =           X,
                'Property Identifier' =         Y,
                'Property Value' =              (any value meeting the range requirements of 7.2.1 except Z)
                'Array Index' =     1
            RECEIVE BACnet-Error-PDU,
                Error Class =           PROPERTY,
                Error Code =            WRITE_ACCESS_DENIED

            IF (the IUT supports AddListElement and the property is an array of lists) THEN
                TRANSMIT AddListElement-Request,
                    'Object Identifier' =   X,
                    'Property Identifier' = Y,
                    'Array Index' =         1
                    'List of Elements' =    (any elements value meeting the range requirements of 7.2.1
                                            excluding those in Z)
                RECEIVE BACnet-Error-PDU,
                    Error Class =       PROPERTY,
                    Error Code =        WRITE_ACCESS_DENIED
            ELSE
            TRANSMIT WriteProperty-Request,
                'Object Identifier' =           X,
                'Property Identifier' =         Y,
                'Property Value' =              (any value meeting the range requirements of 7.2.1)
            RECEIVE BACnet-Error-PDU,
                Error Class =           PROPERTY,
                Error Code =            WRITE_ACCESS_DENIED
        }
    }
```

Notes to tester: When modifying a property, it is expected that an error class of PROPERTY with an error code of WRITE_ACCESS_DENIED is returned but the IUT may instead return an error_class of PROPERTY with an error_code of VALUE_OUT_OF_RANGE, or an error_class of RESOURCES with an error_code of NO_SPACE_TO_WRITE_PROPERTY. In the case that the property is an array, and it has no elements, then the IUT may return and error class of PROPERTY and an error code of INVALID_ARRAY_INDEX. The objects selected by the tester should include one instance of each supported object type. Where some instances of an object type differ in the set of

supported properties, the allowable value ranges for a property, or the writability of a property, then one instance of each "flavor" of that object type should be selected.

7.2.X1 Date Pattern Properties Test

Reason for Change: Addendum 135-2001a-1 adds odd and even month support, and last-day-of-the-month special value. Addendum 135-2008h-8 adds odd and even day support. Addendum 135-2008ac-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested accepts special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. The value, written to the property is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. The list of Specials comes from the Chapter 21 Application Types section on Date.

Test Steps:

1. IF (Protocol_Revision is not present or Protocol_Revision < 4)
   Specials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified)
   ELSE IF (Protocol_Revision >= 4 and Protocol_Revision < 10)
   Specials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month)
   ELSE
   Specials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days)
2. REPEAT SV = (each value in Specials) DO {
   WRITE P1 = (D1 updated with the value SV)
   VERIFY P1 = (D1 updated with the value SV)
   }

Notes to Tester: if P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

7.2.X2 Time Pattern Properties Test

Reason for Change: Addendum 135-2008h.8 adds odd and even day support. Addendum 135-2008ac-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested accepts special time field values.

Test Concept: The property being test, P1, is written with each of the special time field values to ensure that the property accepts them. A time, T1, is selected which is within the time range that the IUT will accept for the property. The value, written to the property is the time T1 with one of its fields replaced with one of the time special values. If the property is a complex datatype the other fields in the value shall be set within the range accepted by the IUT.

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {
   WRITE P1 = (T1 updated with the value SV)
   VERIFY P1 = (T1 updated with the value SV)
   }

Notes to Tester: if P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

7.2.X3 DateTime Pattern Properties Test

Reason for Change: Addendum 135-2001a-1 adds odd and even month support, and last-day-of-the-month special value. Addendum 135-2008h-8 adds odd and even day support. Addendum 135-2008ac-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested accepts special date and time field values.

Test Concept: The property being tested, P1, is written with each of the special date and time field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. A time, T1, is selected which is within the time range that the IUT will accept for the property. The value, written to the property is the date D1 and time T1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype which contains the BACnetDateTime, the other fields in the value shall be set within the range accepted by the IUT. The list of DateSpecials comes from the Chapter 21 Application Types section on Date and the list of TimeSpecials comes from the Chapter 21 Application Types section on Time.

Test Steps:

1. IF (Protocol_Revision is not present or Protocol_Revision < 4)
   DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified)
   ELSE IF (Protocol_Revision >= 4 and Protocol_Revision < 10)
   DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month)
   ELSE
   DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days)
2. TimeSpecials = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified)
3. REPEAT SV = (each value in DateSpecials +TimeSpecials) DO {
   WRITE P1 = (D1+T1 updated with the value SV)
   VERIFY P1 = (D1+T1 updated with the value SV)
   }

Notes to Tester: if P1 is an array, then an array index shall be provided in the WRITE and VERIFY operations.

7.2.X4 Date Non-Pattern Properties Test

Reason for Change: Addendum 135-2001a-1 adds odd and even month support, and last-day-of-the-month special value. Addendum 135-2008h-8 adds odd and even day support. Addendum 135-2008ac-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property does not accept them. A date is selected which is within the date range that the IUT will accept for the property. The value, V1, written to the property is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Value' = (V1 updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X5 Time Non-Pattern Properties Test

Reason for Change: Addendum 135-2008~~ae~~*ac*-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested does not accept special time field values.

Test Concept: The property being tested, P1, is written with each of the special time field values to ensure that the property does not accept them. A time is selected which is within the time range that the IUT will accept for the property. The value, V1, written to the property is the time T1 with one of its fields replaced with one of the time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified)
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Value' = (V1 updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X6 DateTime Non-Pattern Properties Test

Reason for Change: Addendum 135-2001a-1 adds odd and even month support, and last-day-of-the-month special value. Addendum 135-2008*h*-8 adds odd and even day support. Addendum 135-2008*ac*-1 clarifies when wildcards are allowed in dates and times. Test does not exist in 135.1-2013.

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P_1, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT_1 is selected which is within the range that the IUT will accept for the property. The value, V_1, written to the property is the datetime DT_1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {
   TRANSMIT WriteProperty-Request
   'Object Identifier' = O_1,
   'Property Identifier' = P_1,
   'Property Value' = (DT_1 updated with the special value SV)
   RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY,
   'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X7 BACnetDateRange Non-Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: A BACnetDateRange property, or property that is a complex datatype containing a BACnetDateRange, P_1 is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange DR_1 is selected so it is within the range that the IUT will accept for the property. The value, V_1, written to the property is the dateRange DR_1 with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalenderEntry, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. REPEAT SV = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days) DO {
    TRANSMIT WriteProperty-Request
        'Object Identifier' = O_1,
        'Property Identifier' = P_1,
        'Property Value' = (DR_1 with startDate updated with the special value SV)
    RECEIVE BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE
    TRANSMIT WriteProperty-Request
        'Object Identifier' = O1,
        'Property Identifier' = P1,
        'Property Value' = (DR1 with endDate updated with the special value SV)
    Receive BACnet-Error-PDU
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE

Notes to Tester: if P_1 is an array, then an array index shall be provided in the TRANSMIT portion of step 1.

7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test

Reason for Change: Addendum 135-2008*ac*-1 clarifies in the clause 12 preamble, when wildcards are allowed in BACnetDateRange.

Purpose: To verify that the property being tested accepts a fully unspecified date in either or both halves of the value.

Test Concept: A BACnetDateRange property, or property that has a complex datatype containing a BACnetDateRange, P_1 is written with a fully unspecified date in either or both halves to ensure that the property accepts them. DR_1 is selected which is within the date range that the IUT will accept for the property. The value, written to the property is the dateRange DR_1 replaced with a fully unspecified date in either or both startDate and endDate. If the property is a complex datatype the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

Test Steps:

1. WRITE P_1 = (DR_1 updated with a fully unspecified date in startDate)
2. VERIFY P_1 = (the value written)
3. WRITE P_1 = (DR_1 updated with a fully unspecified date in endDate)
4. VERIFY P_1 = (the value written)
5. WRITE P_1 = (DR_1 updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P_1 = (the value written)

Notes to Tester: if P_1 is an array, then an array index shall be provided in the WRITEs and VERIFYs.

7.3 Object Functionality Tests

7.3.1 Property Tests

7.3.1.6 Minimum On/Off Time Tests

7.3.1.6.*1* Override of Minimum Time

Reason for Change: The test was re-written to remove the dependence on the presence of the Minimum_Off_Time property. This test was renumbered from 7.3.1.6 to 7.3.1.6.1.

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 *commandable* 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 76~~ *and 5, and by using the Minimum_Off_Time in Step 4*.

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¥8	Null	**Inactive**	Inactive	Inactive
PA_Index = P¥9	Null	Null	**Inactive**	Inactive

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 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

Minimum_On_Time

T1 T2

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

Minimum_On_Time

T1 T2

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

7.3.1.7 COV Tests

7.3.1.7.X1 COV_Resubscription_Interval Test

Reason for Change: No existing test in the standard.

Dependencies: Confirmed Notifications Subscription, 8.10.1.

BACnet Reference Clause: 12.25.10 and 12.50.15.

Purpose: To verify that object O1 acquiring data via COV notification reissues its subscription at the interval set by COV_Resubscription_Interval.

Test Concept: O1 is configured to acquire data from the TD by COV notification. The TD verifies the resubscription interval.

Configuration RequirementsO1 is configured to acquire data from TD by COV notification. Non-zero values shall be chosen for COV_Resubscription_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

1. IF (the IUT uses SubscribeCOV) THEN
    RECEIVE SubscribeCOV-Request,
        'Subscriber Process Identifier' = (SPI1, any value),
        'Monitored Object Identifier' = (MOI1, the object to be monitored),
        'Issue Confirmed Notifications' = (ICN1 = TRUE | FALSE),
        'Lifetime' = (L1, any value >= COV_Resubscription_Interval)
   ELSE
    RECEIVE SubscribeCOVProperty-Request,
        'Subscriber Process Identifier' = (SPI1, any value),
        'Monitored Object Identifier' = (MOI1, the object to be monitored),
        'Issue Confirmed Notifications' = (ICN1 = TRUE | FALSE),
        'Lifetime' = (L1, any value >= COV_Resubscription_Interval),
        'Monitored Property Identifier' = (MPI1, the property to be monitored),
        'COV Increment' = (CI1, Client_COV_Increment -- optional)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = SPI1,
    'Initiating Device Identifier' = TD,
    'Monitored Object Identifier' = MOI1,
    'Issue Confirmed Notifications' = ICN1,
    'Time Remaining' = (any value <= L1),
    'List of Values' = (appropriate BACnetPropertyValue(s))
4. RECEIVE BACnet-SimpleACK-PDU
5. BEFORE (the lesser of COV_Resubscription_Interval + **Re-subscription Interval Tolerance** and L1~~LifeTime from step 1~~)
    IF (the IUT uses SubscribeCOV)
        RECEIVE SubscribeCOV-Request,
            'Subscriber Process Identifier' = SPI1,
            'Monitored Object Identifier' = MOI1,
            'Issue Confirmed Notifications' = ICN1,
            'Lifetime' = (L2, any value >= COV_Resubscription_Interval)

```
        ELSE
            RECEIVE SubscribeCOVProperty-Request,
                'Subscriber Process Identifier' =      SPI1,
                'Monitored Object Identifier' =        MOI1,
                'Issue Confirmed Notifications' =      ICN1,
                'Lifetime' =                           (L2, any value >=  COV_Resubscription_Interval)
                'Monitored Property Identifier' =      MPI1,
                'COV Increment' =                      CI1
6.  TRANSMIT BACnet-SimpleACK-PDU
7.  TRANSMIT ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' =      SPI1,
        'Initiating Device Identifier' =       TD,
        'Monitored Object Identifier' =        MOI1,
        'Issue Confirmed Notifications' =      ICN1,
        'Time Remaining' =                     (any value <= L2),
        'List of Values' =                     (appropriate BACnetPropertyValue(s))
8.  RECEIVE BACnet-SimpleACK-PDU
9.  WAIT (COV_Resubscription_Interval - Re-subscription Interval Tolerance)
10. BEFORE (2 *  Re-subscription Interval Tolerance)
        IF (the IUT uses SubscribeCOV)
            RECEIVE SubscribeCOV-Request,
                'Subscriber Process Identifier' =      SPI1,
                'Monitored Object Identifier' =        MOI1,
                'Issue Confirmed Notifications' =      ICN1,
                'Lifetime' =                           L1
        ELSE
            RECEIVE SubscribeCOVProperty-Request,
                'Subscriber Process Identifier' =      SPI1,
                'Monitored Object Identifier' =        MOI1,
                'Issue Confirmed Notifications' =      ICN1,
                'Lifetime' =                           L1,
                'Monitored Property Identifier' =      MPI1,
                'COV Increment' =                      CI1
11. TRANSMIT BACnet-SimpleACK-PDU
```

Passing Result: Where the Lifetime parameter of a SubscribeCOV request is less than COV_Resubscription_Interval + Re-subscription Interval Tolerance, the IUT shall send the subsequent SubscribeCOV request within Lifetime seconds even though this is a smaller time window than defined by the test. If the IUT does not meet this stricter time window, then the IUT shall fail the test.

7.3.1.9 Binary Object Elapsed Active Time Tests

Reason for Change: Errors were pointed out via BTL-CR-0253.

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

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

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

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

object is in an ACTIVE state. The Present_Value is then set to INACTIVE and the Elapsed_Active_Time is reset. The Time_Of_Active_Time_Reset property is checked to verify that it has been updated.

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

Test Steps:

1. IF (Present_Value is writable) THEN
   WRITE Present_Value = INACTIVE
   VERIFY Present_Value = INACTIVE
   ELSE
   MAKE (Present_Value = INACTIVE)
2. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Elapsed_Active_Time
3. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Elapsed_Active_Time,
   'Property Value' = (the elapsed active time, $T_{ELAPSED}$ in seconds)
4. WAIT (1 minute)
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Elapsed_Active_Time
6. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the object being tested),
   'Property Identifier' = Elapsed_Active_Time,
   'Property Value' = (the same $T_{ELAPSED}$ as step 3)
7. IF (Present_Value is writable) THEN
   WRITE Present_Value = ACTIVE
   VERIFY Present_Value = ACTIVE
   ELSE
   MAKE (Present_Value = ACTIVE)
8. WAIT (**Internal Processing Fail Time** + 30 seconds)
9. *IF (Present_Value is writable) THEN*
   WRITE Present_Value = INACTIVE
   VERIFY Present_Value = INACTIVE
   ELSE
   MAKE (Present_Value = INACTIVE)
10. TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the object being tested),
    'Property Identifier' = Elapsed_Active_Time
11. RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the object being tested),
    'Property Identifier' = Elapsed_Active_Time,
    'Property Value' = (T: $(T_{ELAPSED} + 30) \leq T \leq (T_{ELAPSED}$ + *TimeX, where TimeX is the time between the beginning of step 7 and this step*~~30 + **Internal Processing Fail Time**~~))

~~11. IF (Present_Value is writable) THEN~~
~~WRITE Present_Value = INACTIVE~~
~~VERIFY Present_Value = INACTIVE~~
~~ELSE~~
~~MAKE (Present_Value = INACTIVE)~~

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

MAKE (Elapsed_Active_Time = 0)
13. TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Date
14. RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local Date,
    'Property Value' = (the current local date, D)
15. TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Time
16. RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Time,
    'Property Value' = (the current local time, T_{LOC})
17. TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the object being tested),
    'Property Identifier' = Time_Of_Active_Time_Reset
18. RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the object being tested),
    'Property Identifier' = ~~Present_Value~~*Time_Of_Active_Time_Reset*,
    'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})

7.3.1.10 Event_Enable Tests

7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL and TO_NORMAL

Reason For Change: This test was modified to take into account the Feedback behavior that is required by the Output objects.

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.23, 12.2.24, 12.3.20, 12.5.22, 12.6.26, 12.7.24, 12.11.10, 12.14.18, 12.15.18, 12.16.33, 12.17.17, 12.18.18, 12.19.18 and 12.23.23.

Purpose: To verify that notification messages are transmitted only if the bit in Event_Enable corresponding to the event transition has a value of TRUE. This test applies to Event Enrollment objects and objects that support intrinsic reporting.

Test Concept: The IUT is configured such that the Event_Enable property indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

Configuration Requirements: The Event_Enable property shall be configured with a value of TRUE for either the TO-OFFNORMAL transition or the TO-NORMAL transition and the other event transition shall have a value of FALSE. If the Event_Enable property is not configurable, follow the test steps as written and verify correct behavior for the value of the Event_Enable property. For analog objects the Limit_Enable property shall be configured with the value (TRUE, TRUE). The referenced event-triggering property shall be set to a value that results in a NORMAL condition. If a Notification Class object is being used to configure recipient information the value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

In the test description below, "X" is used to designate the event-triggering property.

1. VERIFY Event_State = NORMAL
2. WAIT (Time_Delay + **Notification Fail Time**)
3. IF (X is *the Present_Value property in a Binary Output or Multi-state Output object) THEN*

```
        MAKE (the Feedback_Value property differe from the X property)
    ELSE IF (X is writable) THEN
        WRITE X = (a value that is OFFNORMAL)
    ELSE
        MAKE (X have a value that is OFFNORMAL)
4.  WAIT (Time_Delay)
5.  BEFORE Notification Fail Time
        IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =              (any valid process ID),
                'Initiating Device Identifier' =    IUT,
                'Event Object Identifier' =         (the event-generating object configured for this test),
                'Time Stamp' =                      (the current local time),
                'Notification Class' =              (the class corresponding to the object being tested),
                'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                      (any valid event type),
                'Notify Type' =                     EVENT | ALARM,
                'AckRequired' =                     TRUE | FALSE,
                'From State' =                      NORMAL,
                'To State' =                        OFFNORMAL,
                'Event Values' =                    (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
6.  VERIFY Event_State = OFFNORMAL
7.  IF (X is the Present_Value property in a Binary Output or Multi-state Output object) THEN
        MAKE (the Feedback_Value property differe from the X property)
    ELSE IF (X is writable) THEN
        WRITE X = (a value that is NORMAL)
    ELSE
        MAKE (X have a value that is NORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE Notification Fail Time
        IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =              (any valid process ID),
                'Initiating Device Identifier' =    IUT,
                'Event Object Identifier' =         (the event-generating object configured for this test),
                'Time Stamp' =                      (the current local time),
                'Notification Class' =              (the class corresponding to the object being tested),
                'Priority' =                        (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                      (any valid event type),
                'Notify Type' =                     EVENT | ALARM,
                'AckRequired' =                     TRUE | FALSE,
                'From State' =                      OFFNORMAL,
                'To State' =                        NORMAL,
                'Event Values' =                    (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY Event_State = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
        MAKE (the event-triggering object change to a fault condition)
        BEFORE Notification Fail Time
            IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN
                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
```

```
                'Event Object Identifier' =     (the event-generating object configured for this test),
                'Time Stamp' =                  (the current local time),
                'Notification Class' =          (the class corresponding to the object being tested),
                'Priority' =                    (the value configured to correspond to a TO-FAULT transition),
                'Event Type' =                  (any valid event type),
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  NORMAL,
                'To State' =                    FAULT,
                'Event Values' =                (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
    VERIFY Event_State = FAULT
}
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.10.2 Event_Enable Tests for TO_NORMAL only Algorithms

Reason For Change: There is an error pointed out by BTL-CR-0196, of not returning the TO_NORMAL bit of the Event_Enable to TRUE in step 7.

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

Purpose: To verify that notification messages are transmitted only if the bit in Event_Enable corresponding to the event transition has a value of TRUE. This test applies to objects that only support generation of TO_NORMAL transitions.

Test Concept: The IUT is configured such that the Event_Enable property indicates that some event transitions are to trigger an event notification~~ and some are not~~. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

Configuration Requirements: In the Notification Class object providing recipient information, the value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

1. VERIFY Event_State = NORMAL
2. MAKE (the TO_NORMAL bit of the Event_Enable property equal to TRUE)
3. MAKE (a condition exist that would cause the object to generate a TO_NORMAL transition)
4. BEFORE **Notification Fail Time**

```
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' =          (any valid process ID),
        'Initiating Device Identifier' =   IUT,
        'Event Object Identifier' =  (the event-generating object configured for this test),
        'Time Stamp' =          (the current local time),
        'Notification Class' =          (the class corresponding to the object being tested),
        'Priority' =            (the value configured to correspond to a TO_NORMAL transition),
        'Event Type' =          (any valid event type),
        'Notify Type' =         EVENT | ALARM,
        'AckRequired' =         TRUE | FALSE,
        'From State' =          NORMAL,
        'To State' =            NORMAL,
        'Event Values' =        (values appropriate to the event type)
```

5. TRANSMIT SimpleAck-PDU
6. VERIFY Event_State = NORMAL
7. IF (Event_Enable can be changed such that the TO_NORMAL transition is FALSE) THEN
    MAKE (the TO_NORMAL bit of the Event_Enable property equal to FALSE)
    MAKE (a condition exist that would cause the object to generate a TO_NORMAL transition)
    CHECK (verify that the IUT did not transmit an event notification message)
    MAKE (the TO_NORMAL bit of the Event_Enable property equal to TRUE)
8. IF (the event-generating object can be placed into a fault condition) THEN
    IF (Event_Enable can be modified) THEN
        MAKE(Event_Enable TO_FAULT transition equal TRUE)
    IF (Event_Enable TO_FAULT transition = TRUE) THEN
        MAKE (the event-triggering object change to a fault condition)
        BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' = (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' = (the event-generating object configured for this test),
                'Time Stamp' = (the current local time),
                'Notification Class' = (the class corresponding to the object being tested),
                'Priority' = (the value configured to correspond to a TO_FAULT transition),
                'Event Type' = (any valid event type),
                'Notify Type' = EVENT | ALARM,
                'AckRequired' = TRUE | FALSE,
                'From State' = NORMAL,
                'To State' = FAULT,
                'Event Values' = (values appropriate to the event type)
            TRANSMIT SimpleAck-PDU
        VERIFY Event_State = FAULT
        MAKE (the event-triggering object change to a normal condition)
        BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' = (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' = (the event-generating object configured for this test),
                'Time Stamp' = (the current local time),
                'Notification Class' = (the class corresponding to the object being tested),
                'Priority' = (the value configured to correspond to a TO_NORMAL transition),
                'Event Type' = (any valid event type),
                'Notify Type' = EVENT | ALARM,
                'AckRequired' = TRUE | FALSE,
                'From State' = FAULT,
                'To State' = NORMAL,
                'Event Values' = (values appropriate to the event type)
            TRANSMIT SimpleAck-PDU
9. IF (Event_Enable can be modified) THEN
    MAKE (Event_Enable TO_FAULT transition equal FALSE)
10. IF (Event_Enable TO_FAULT transition = FALSE) THEN
    MAKE (the event-triggering object change to a fault condition)
    VERIFY Event_State = FAULT
    CHECK (verify that the IUT did not transmit an event notification message)
    MAKE (the event-triggering object change to a normal condition)

Notes to Tester: For objects that do not have a Time_Delay property, the Time_Delay value used in the test shall be 0. The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.11 Acked_Transitions Tests

Reason For Change: Corrected language of parameter descriptions.

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; AcknowledgeAlarm Service Execution Tests, 9.1; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.28, 12.2.24, 12.3.25, 12.4.21, 12.6.23, 12.7.27, 12.8.25, 12.12.11, 12.15.20, 12.16.20,12.17.35, 12.18.18, 12.19.19, 12.20.19, 12.23.27 and 12.25.23.

Purpose: To verify that the Acked_Transitions property tracks whether or not an acknowledgment has been received for a previously issued event notification. It also verifies the interrelationship between Status_Flags and Event_State. This test applies to Event Enrollment objects and Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Loop, Multi-state Input, Multi-state Output, Multi-state Value, Pulse Converter and Trend Log objects that support intrinsic reporting.

Test Concept: The IUT is configured such that the Event_Enable property indicates that all event transitions are to trigger an event notification. The Acked_Transitions property shall have the value (TRUE, TRUE, TRUE) indicating that all previous transitions have been acknowledged. Each event transition is triggered and the Acked_Transitions property is monitored to verify that the appropriate bit is cleared when a notification message is transmitted and reset if an acknowledgment is received.

Configuration Requirements: The Event_Enable and Acked_Transitions properties shall be configured with a value of (TRUE, TRUE, TRUE). For analog objects the Limit_Enable property shall be configured with the value (TRUE, TRUE). The referenced event-triggering property shall be set to a value that results in a NORMAL condition. The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

1. WAIT (Time_Delay + **Notification Fail Time**)
2. VERIFY Event_State = NORMAL
3. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
4. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
5. IF (X is writable) THEN
   WRITE X = (a value that is OFFNORMAL)
   ELSE
   MAKE (X have a value that is OFFNORMAL)
6. WAIT (Time_Delay)
7. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object configured for this test),
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	(any valid event type),
'Notify Type' =	(the notify type configured for this event),
'AckRequired' =	TRUE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	(values appropriate to the event type)

8. TRANSMIT BACnet-SimpleACK-PDU
9. VERIFY Event_State = OFFNORMAL
10. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
11. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
12. IF (X is writable) THEN
    WRITE X = (a value that is NORMAL)
    ELSE
    MAKE (X have a value that is NORMAL)
13. WAIT (Time_Delay)
14. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = (any valid event type),
    'Notify Type' = (the notify type configured for this event),
    'AckRequired' = TRUE,
    'From State' = OFNORMAL,
    'To State' = NORMAL,
    'Event Values' = (values appropriate to the event type)
15. TRANSMIT BACnet-SimpleACK-PDU
16. VERIFY Event_State = NORMAL
17. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
18. VERIFY Status_Flags = (FALSE, FALSE, ?,?)
19. IF (the event-triggering object can be placed into a fault condition) THEN
20. MAKE (the event-triggering object change to a fault condition)
21. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-FAULT transition),
    'Event Type' = (any valid event type),
    'Notify Type' = (the notify type configured for this event),
    'AckRequired' = TRUE,
    'From State' = NORMAL,
    'To State' = FAULT,
    'Event Values' = (values appropriate to the event type)
22. TRANSMIT BACnet-SimpleACK-PDU
23. VERIFY Event_State = FAULT
24. VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
25. VERIFY Status_Flags = (TRUE, TRUE, ?, ?)
26. MAKE (the event-triggering object change to a normal condition)
27. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-generating object configured for this test),
    'Time Stamp' = (any valid time stamp),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),

'Event Type' = (any valid event type),
'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)

28. TRANSMIT BACnet-SimpleACK-PDU
29. VERIFY Event_State = NORMAL
30. VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
31. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
32. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' in step 21),
    'Event Object Identifier' = (the 'Event Object Identifier' in step 21),
    'Event State Acknowledged' = FAULT,
    'Time Stamp' = (the 'Time Stamp' in step 21),
    'Time of Acknowledgment' = (the TD's current time)
33. RECEIVE BACnet-SimpleACK-PDU
34. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the value of the 'Process Identifier' in step 21),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the 'Event Object Identifier' in step 21),
    'Time Stamp' = (the *current time or sequence number*~~'Time Stamp' in step 21~~),
    'Notification Class' = (the 'Notification Class' in step 21),
    'Priority' = (the 'Priority' in step 21),
    'Event Type' = (the 'Event Type' in step 21),
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = FAULT
    ELSE
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the value of the 'Process Identifier' in step 21),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the 'Event Object Identifier' in step 21),
    'Time Stamp' = (the *current time or sequence number*~~'Time Stamp' in step 21~~),
    'Notification Class' = (the 'Notification Class' in step 21),
    'Priority' = (the 'Priority' in step 21),
    'Event Type' = (the 'Event Type' in step 21),
    'Notify Type' = ACK_NOTIFICATION
35. TRANSMIT BACnet-SimpleACK-PDU
36. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
37. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' in step 27),
    'Event Object Identifier' = (the 'Event Object Identifier' in step 27),
    'Event State Acknowledged' = NORMAL,
    'Time Stamp' = (the 'Time Stamp' in step 27),
    'Time of Acknowledgment' = (the TD's current time)
38. RECEIVE BACnet-SimpleACK-PDU
39. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the value of the 'Process Identifier' in step 27),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the 'Event Object Identifier' in step 27),
    'Time Stamp' = (the current time or sequence number),

'Notification Class' = (the 'Notification Class' in step 27),
'Priority' = (the 'Priority' in step 27),
'Event Type' = (the 'Event Type' in step 27),
'Notify Type' = ACK_NOTIFICATION,
'To State' = NORMAL

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (the value of the 'Process Identifier' in step 27),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the 'Event Object Identifier' in step 27),
'Time Stamp' = (the current time or sequence number),
'Notification Class' = (the 'Notification Class' in step 27),
'Priority' = (the 'Priority' in step 27),
'Event Type' = (the 'Event Type' in step 27),
'Notify Type' = ACK_NOTIFICATION

40. TRANSMIT BACnet-SimpleACK-PDU
41. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
42. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' in step 7),
'Event Object Identifier' = (the 'Event Object Identifier' in step 7),
'Event State Acknowledged' = OFFNORMAL,
'Time Stamp' = (the 'Time Stamp′ in step 7),
'Time of Acknowledgment' = (the TD's current time)
43. RECEIVE BACnet-SimpleACK-PDU
44. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (the value of the 'Process Identifier' in step 7),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the 'Event Object Identifier' in step 7),
'Time Stamp' = (the current time or sequence number),
'Notification Class' = (the 'Notification Class' in step 7),
'Priority' = (the 'Priority' in step 7),
'Event Type' = (the 'Event Type' in step 7),
'Notify Type' = ACK_NOTIFICATION,
'To State' = OFFNORMAL

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (the value of the 'Process Identifier' in step 7),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the *'Event Object Identifier' in step 7*~~current time or sequence number~~),
'Time Stamp' = (the *current time or sequence number*~~'Time Stamp' in step 7~~),
'Notification Class' = (the 'Notification Class' in step 7),
'Priority' = (the 'Priority' in step 7),
'Event Type' = (the 'Event Type' in step 7),
'Notify Type' = ACK_NOTIFICATION

45. TRANSMIT BACnet-SimpleACK-PDU
46. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.13 Limit_Enable Tests

~~Reason for Change: Added a missing step to check that a notification is not sent.~~

~~Dependencies: ConfirmedEventNotification-Service-Initiation-Tests, 8.4; UnconfirmedEventNotification-Service-Initiation Tests, 8.5; ReadProperty-Service-Execution-Tests, 9.18; WriteProperty-Service-Execution-Tests, 9.22.~~

~~BACnet Reference Clauses: 12.1.22, 12.2.23, and 12.3.19.~~

~~Purpose: To verify that the Limit_Enable property correctly enables or disables reporting of out-of-range events. This test applies to objects with a Limit_Enable property.~~

~~Test Concept: The event-triggering property is manipulated to cause both the high limit and the low limit to be exceeded for each possible combination of values for Limit_Enable. The resulting event-notification messages are monitored to verify that they are transmitted only for circumstances where the associated event limit is enabled.~~

~~Configuration Requirements: Configure the object with High_Limit, Low_Limit and Deadband values such that High_Limit - Deadband > Low_Limit and both the Low_Limit and High_Limit values are within the valid range of values for Present_Value. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property should be set to (TRUE, ?, TRUE) for this test. If the Event_Enable cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test may be skipped.~~

~~In the test description below "X" is used to designate the event-triggering property.~~

~~Test Steps:~~

~~1. IF Limit_Enable can be made to be equal (TRUE, TRUE)~~
~~2. If Limit_Enable is writable~~
~~WRITE Limit_Enable = (TRUE, TRUE)~~
~~ELSE~~
~~MAKE Limit_Enable = (TRUE, TRUE)~~
~~3. WAIT (Time_Delay + **Notification Fail Time**)~~
~~4. VERIFY Event_State = NORMAL~~
~~5. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~6. WAIT (Time_Delay)~~
~~7. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = HIGH_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~8. TRANSMIT SimpleAck-PDU~~
~~9. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~

~~MAKE (X a value that is lower than Low_Limit)~~
~~10 . WAIT (Time_Delay)~~
~~11. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' =(the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = HIGH_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~12. TRANSMIT SimpleAck-PDU~~
~~13. WAIT (Time_Delay)~~
~~14. BEFORE **Notification Fail Time**~~
~~RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = LOW_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~15. TRANSMIT SimpleAck-PDU~~
~~16. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit + deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between than Low_Limit + deadband and High_Limit)~~
~~17. WAIT (Time_Delay)~~
~~18. BEFORE **Notification Fail Time** RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = LOW_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~19. TRANSMIT SimpleAck-PDU~~
~~20. IF Limit_Enable can be made to equal (FALSE, TRUE)~~

~~21. IF Limit_Enable is writable~~
~~WRITE Limit_Enable = (FALSE, TRUE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (FALSE,TRUE))~~
~~22. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~23. WAIT (Time_Delay)~~
~~24. BEFORE **Notification Fail Time** RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = HIGH_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~25. TRANSMIT SimpleAck-PDU~~
~~26. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit and High_Limit-Deadband)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit and High_Limit-Deadband)~~
~~27. WAIT (Time_Delay)~~
~~28. BEFORE **Notification Fail Time** RECEIVE ConfirmdEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = HIGH_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~29. TRANSMIT SimpleAck-PDU~~
~~30. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~31. WAIT (Time_Delay + **Notification Fail Time**)~~
~~32. CHECK (verify that no notification message was transmitted)~~
~~*33.* IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit+Deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit+Deadband and High_Limit)~~
~~34. WAIT (Time_Delay + **Notification Fail Time**)~~
~~*35. CHECK (verify that no notification message was transmitted)*~~

~~36. IF Limit_Enable can be made to equal (TRUE, FALSE)~~
~~37. IF Limit_Enable is writable~~
~~WRITE Limit_Enable = (TRUE, FALSE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (TRUE, FALSE))~~
~~38. IF (X is writable) THEN~~
~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~39. WAIT (Time_Delay + **Notification Fail Time**)~~
~~40. CHECK (verify that no notification message was transmitted)~~
~~41. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~42. WAIT (Time_Delay)~~
~~43. BEFORE **Notification Fail Time** RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = NORMAL,~~
~~'To State' = LOW_LIMIT,~~
~~'Event Values' = (values appropriate to the event type)~~
~~44. TRANSMIT SimpleAck-PDU~~
~~45. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit + Deadband and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit + Deadband and High_Limit)~~
~~46. WAIT (Time_Delay)~~
~~47. BEFORE **Notification Fail Time** RECEIVE ConfirmedEventNotification-Request,~~
~~'Process Identifier' = (any valid process ID),~~
~~'Initiating Device Identifier' = IUT,~~
~~'Event Object Identifier' = (the object configured for this test),~~
~~'Time Stamp' = (the current local time),~~
~~'Notification Class' = (the class corresponding to the object being tested),~~
~~'Priority' = (the value configured to correspond to a TO-NORMAL transition),~~
~~'Event Type' = OUT_OF_RANGE,~~
~~'Notify Type' = ALARM | EVENT,~~
~~'AckRequired' = TRUE | FALSE,~~
~~'From State' = LOW_LIMIT,~~
~~'To State' = NORMAL,~~
~~'Event Values' = (values appropriate to the event type)~~
~~48. IF Limit_Enable can be made to equal (FALSE, FALSE)~~
~~49. IF Limit_Enable is writable~~
~~WRITE Limit_Enable = (FALSE, FALSE)~~
~~ELSE~~
~~MAKE (Limit_Enable = (FALSE, FALSE))~~
~~50. IF (X is writable) THEN~~

~~WRITE X = (a value that exceeds High_Limit)~~
~~ELSE~~
~~MAKE (X a value that exceeds High_Limit)~~
~~51. WAIT (Time_Delay + **Notification Fail Time**)~~
~~52. CHECK (verify that no notification message was transmitted)~~
~~53. IF (X is writable) THEN~~
~~WRITE X = (a value that is lower than Low_Limit)~~
~~ELSE~~
~~MAKE (X a value that is lower than Low_Limit)~~
~~54. WAIT (Time_Delay + **Notification Fail Time**)~~
~~55. CHECK (verify that no notification message was transmitted)~~
~~56. IF (X is writable) THEN~~
~~WRITE X = (a value that is between Low_Limit and High_Limit)~~
~~ELSE~~
~~MAKE (X a value that is between Low_Limit and High_Limit)~~

~~57. WAIT (Time_Delay + **Notification Fail Time**)~~
~~58. CHECK (verify that no notification message was transmitted)~~

~~Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.~~

7.3.1.13.X1 Limit_Enable Test, LowLimitEnable

Reason for Change: New test to reduce scope of original test and simplify.

Purpose: To verify that the LowLimitEnable flag in the Limit_Enable property correctly enables or disables reporting of out of range events. This test applies to objects with a Limit_Enable property.

Test Concept: The LowLimitEnable flag is set to true in the Limit_Enable property and the event-triggering property is manipulated to cause the low limit to be exceeded. This should generate an event notification and make Event_State = Low_Limit. After the event-triggering property is returned to a normal value, the LowLimitEnable flag is the set to false and the event-triggering property is again manipulated to exceed the low limit. No event notification should be observed and the Event_State must have a value of normal.

Configuration Requirements: Configure the object with pHighLimit, pLowLimit and pDeadband values such that pLowLimit + pDeadband < pHighLimit and both the pLowLimit and pHighLimit values are within the valid range of values for the event-triggering property. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event_Enable property cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test shall be skipped.

Test Steps:

1. MAKE pLimitEnable = (TRUE, ?)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value less than pLowLimit)
4. WAIT (pTimeDelay)
5. BEFORE Notification Fail Time
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object configured for this test),
   'Time Stamp' = (the current local time),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),

'Event Type' =	OUT_OF_RANGE,
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	LOW_LIMIT,
'Event Values' =	(values appropriate to the event type)

6. TRANSMIT SimpleAck-PDU
7. VERIFY pCurrentState = LOW_LIMIT
8. MAKE (pMonitoredValue a value that is between pLowLimit + pDeadband and pHighLimit)
9. WAIT (pTimeDelayNormal)
10. BEFORE Notification Fail Time

    RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object configured for this test),
'Time Stamp' =	(the current local time),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	OUT_OF_RANGE,
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	LOW_LIMIT,
'To State' =	NORMAL,
'Event Values' =	(values appropriate to the event type)

11. TRANSMIT SimpleAck-PDU
12. MAKE pLimitEnable = (FALSE, ?)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value less than pLowLimit)
15. WAIT (pTimeDelay + Notification Fail Time)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY pCurrentState = NORMAL

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.13.X2 Limit_Enable Test, HighLimitEnable

Reason for Change: New test to reduce scope of original test and simplify.

Purpose: To verify that the HighLimitEnable flag in the Limit_Enable property correctly enables or disables reporting of out of range events. This test applies to objects with a Limit_Enable property.

Test Concept: The HighLimitEnable flag is set to true in the Limit_Enable property and the event-triggering property is manipulated to cause the high limit to be exceeded. This should generate an event notification and make Event_State = High_Limit. After the event-triggering property is returned to a normal value, the HighLimitEnable flag is the set to false and the event-triggering property is again manipulated to exceed the high limit. No event notification should be observed and the Event_State must have a value of normal.

Configuration Requirements: Configure the object with pHighLimit, pLowLimit and pDeadband values such that pHighLimit - pDeadband > pLowLimit and both the pLowLimit and pHighLimit values are within the valid range of values for the event triggering property. If the device cannot be configured with limit values that meet these conditions, then this test shall be skipped. The Event_Enable property shall be set to (TRUE, ?, TRUE) for this test. If the Event_Enable property cannot be configured such that the TO-NORMAL and the TO-OFFNORMAL transitions are TRUE, this test shall be skipped.

Test Steps:

1. MAKE pLimitEnable = (?, TRUE)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value that exceeds pHighLimit)
4. WAIT (pTimeDelay)
5. BEFORE Notification Fail Time
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object configured for this test),
   'Time Stamp' = (the current local time),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
   'Event Type' = OUT_OF_RANGE,
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = HIGH_LIMIT,
   'Event Values' = (values appropriate to the event type)
6. TRANSMIT SimpleAck-PDU
7. VERIFY pCurrentState = HIGH_LIMIT
8. MAKE (pMonitoredValue a value that is between pLowLimit and pHighLimit - pDeadband)
9. WAIT (pTimeDelayNormal)
10. BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object configured for this test),
    'Time Stamp' = (the current local time),
    'Notification Class' = (the class corresponding to the object being tested),
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    'Event Type' = OUT_OF_RANGE,
    'Notify Type' = ALARM | EVENT,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = (values appropriate to the event type)
11. TRANSMIT SimpleAck-PDU
12. MAKE pLimitEnable = (?, FALSE)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value that exceeds pHighLimit)
15. WAIT (pTimeDelay + Notification Fail Time)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY pCurrentState = NORMAL

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.X4 Event_Message_Texts Tests

Reason For Change: 135-2008z-1. This test does not exist in 135.1-2013.

Purpose: To verify that the value of the Event_Message_Texts property is updated when an object generates an event notification.

Test Concept: Read the Event_Message_Texts from the object. Transition the object through each event state which is enabled in the object saving the Message Text parameter from the received notification. Verify that the Event_Message_Texts updates with the Event_Message_Texts value received from the notification.

Configuration Requirements: The IUT shall be configured with an event-generation object, O_1 which shall be in a NORMAL Event_State at the beginning of the test. If the algorithm of the object does not support NORMAL to NORMAL transitions, then the TO-OFFNORMAL bit of the Event_Enable shall be TRUE. If the IUT does not contain any objects which can transition to any offnormal state, then this test shall be skipped.

In the test description below X_1 is used to designate the event-triggering property linked to O_1.

Test Steps:

1. READ EMT = Event_Message_Texts
2. IF (Event_Enable is (TRUE, ?, ?)) THEN
3. IF (X_1 is writable) THEN
   WRITE X_1 = (a value that is offnormal)
   ELSE
   MAKE (X_1 a value that is offnormal)
4. WAIT (Time_Delay)
5. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (O_1),
   'Time Stamp' = (the IUT's local time),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the configured TO_OFFNORMAL priority),
   'Event Type' = (any valid event type),
   'Notify Type' = Notify_Type,
   'AckRequired' = (the configured value for the TO_OFFNORMAL transition),
   'From State' = NORMAL,
   'To State' = (any valid offnormal state),
   'Message Text' = (M: any valid value placed into EMT[1]),
   'Event Values' = (values appropriate to the event type)
6. VERIFY Event_Message_Texts = EMT
7. IF (Event_Enable is (?, ?, TRUE)) THEN
8. IF (X_1 is writable) THEN
   WRITE X_1 = (a value that will result in a TO_NORMAL transition)
   ELSE
   MAKE (X_1 a value that will result in a TO_NORMAL transition)
9. WAIT (Time_Delay)
10. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (O_1),
   'Time Stamp' = (the IUT's local time),
   'Notification Class' = (the class corresponding to the object being tested),
   'Priority' = (the configured TO_NORMAL priority),
   'Event Type' = (any valid event type),
   'Notify Type' = Notify_Type,
   'AckRequired' = (the configured value for the TO_NORMAL transition),
   'From State' = (any valid value),
   'To State' = NORMAL,
   'Message Text' = (M: any valid value placed into EMT[3]),

'Event Values' = (values appropriate to the event type)
11. VERIFY Event_Message_Texts = EMT
12. IF (Event_Enable is (?, TRUE, ?)) THEN
13. MAKE (O_1 transition to a FAULT state)
14. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (O_1),
'Time Stamp' = (the IUT's local time),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the configured TO_FAULT priority),
'Event Type' = (any valid event type),
'Notify Type' = Notify_Type,
'AckRequired' = (the configured value for the TO_FAULT transition),
'From State' = (any valid value),
'To State' = FAULT,
'Message Text' = (M: any valid value placed into EMT[2]),
'Event Values' = (values appropriate to the event type)
15. VERIFY Event_Message_Texts = EMT

7.3.1.X5 Event_Message_Texts_Config Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that the Message Text parameter of generated event notifications is controlled via the Event_Message_Texts_Config property.

Test Concept: Select an object, O1, in the IUT that supports the Event_Message_Texts_Config property. Make O1 perform each supported event transition (i.e. to-offnormal, to-normal and to-fault). Verify that the 'Message Text' parameter matches the associated Event_Message_Texts_Config value. Note that due to the use of text substitution codes, the resulting text might not be an exact match.

Configuration Requirements: Configure each entry in the Event_Message_Texts_Config property of Object O1 to be distinct, if possible. ES1 shall be the state to which O1 transitions. DELAY shall represent the time delay appropriate to the transition being tested (i.e. Time_Delay for TO_OFFNORMAL, 0 for TO_FAULT and FAULT to NORMAL transitions, and either Time_Delay or Time_Delay_Normal for TO_NORMAL). ESINDEX shall be the array index associated with ES1 (1 for offnormal states, 2 for FAULT, and 3 for NORMAL). The notification class for O1 is configured for UnconfirmedEventNotification.

Test Steps:
1. MAKE(a condition exist which will cause O1 to transition to ES1)
2. WAIT DELAY
3. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid timestamp),
'Notification Class' = (the notification class configured for O1),
'Priority' = (any valid priority),
'Event Type' = (any standard event type),
'Message Text' = T1,
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = (any valid event state),
'To State' = ES1,

'Event Values' = (any values appropriate to the event type)

4. CHECK(T1 is equivalent to Event_Message_Texts_Config[ESINDEX] with any text substitutions as defined by the vendor)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X6 Event_Algorithm_Inhibit Tests

7.3.1.X6.1 Event_Algorithm_Inhibit Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that Event_Algorithm_Inhibit property in objects with intrinsic or algorithmic reporting controls whether or not the event state detection algorithm is executed.

Test Concept: Select an event generating object, O1, which supports the Event_Algorithm_Inhibit property and does not support the Event_Algorithm_Inhibit_Ref property. With Event_Algorithm_Inhibit set to FALSE, make a condition exist that should result in an event transition to a normal or offnormal state. Verify that a transition occurs and that a notification is generated. Set Event_Algorithm_Inhibit to TRUE. Verify that the object transitions to NORMAL, if not already in that state. Make a condition exist that should result in an event transition if the object Event_Algorithm_Inhibit were FALSE. If O1 supports fault detection, make a fault condition exist and verify that object detects it and transitions to FAULT.

Configuration Requirements: O1 is configured to detect and report unconfirmed events, is in the NORMAL state and, if supported, is configured to detect fault conditions.

Test Steps:

1. VERIFY Event_State = NORMAL
2. VERIFY Event_Algorithm_Inhibit = FALSE
3. MAKE (a condition exist which results in a transition of O1. If possible, 'To State' shall be an offnormal event state)
4. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
        'Process Identifier' = (PID1: any valid process identifier),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1,
        'Time Stamp' = (the current local time or sequence number),
        'Notification Class' = (the notification class configured for O1),
        'Priority' = (the value configured for the transition),
        'Event Type' = (ET1: any valid event type),
        'Notify Type' = (value from the Notify_Type property configured for O1),
        'Message Text' = (any valid message text),
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = (ES1: any event state appropriate to the event type),
        'Event Values' = (any values appropriate to the event type)
5. WRITE Event_Algorithm_Inhibit = TRUE
6. IF (ES1 <> NORMAL) THEN
    BEFORE **Notification Fail Time**
        RECEIVE UnconfirmedEventNotification-Request
            'Process Identifier' = PID1,
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = O1,
            'Time Stamp' = (the current local time or sequence number),
            'Notification Class' = (the notification class configured for O1),
            'Priority' = (the value configured for the transition),
            'Event Type' ET1,

'Notify Type' = (value from the Notify_Type property configured for O1),
'Message Text' = (any valid message text),
'AckRequired' = TRUE | FALSE,
'From State' = ES1,
'To State' = NORMAL,
'Event Values' = (any values appropriate to the event type)

7. VERIFY Event_State = NORMAL
8. MAKE (a condition exist which would result in a transition of O1 other than to FAULT, if Event_Algorithm_Inhibit were FALSE.)
9. WAIT **Notification Fail Time**
10. CHECK (that the IUT did not send any event notifications other than to FAULT for O1)
11. VERIFY Event_State = NORMAL
12. IF (O1 supports fault detection) THEN
    MAKE (a fault condition exist for O1)
    BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = PID1,
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = (the current local time or sequence number),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_RELIABILITY,
    'Notify Type' = (value from the Notify_Type property configured for O1),
    'Message Text' = (any valid message text),
    'AckRequired' = TRUE | FALSE,
    'From State' = NORMAL,
    'To State' = FAULT,
    'Event Values' = (any values appropriate for CHANGE_OF_RELIABILITY)
    MAKE (remove the fault condition)
    BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
    'Process Identifier' = PID1,
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = O1,
    'Time Stamp' = (the current local time or sequence number),
    'Notification Class' = (the notification class configured for O1),
    'Priority' = (the value configured for the transition),
    'Event Type' = CHANGE_OF_RELIABILITY,
    'Notify Type' = (value from the Notify_Type property configured for O1),
    'Message Text' = (any valid message text),
    'AckRequired' = TRUE | FALSE,
    'From State' = FAULT,
    'To State' = NORMAL,
    'Event Values' = (any values appropriate for CHANGE_OF_RELIABILITY)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X6.2 Event_Algorithm_Inhibit Summarization Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that event generating objects are reported by summarization routines as needed even when the algorithm has been inhibited.

Test Concept: Select an event generating object O1 which is configured for event reporting, which is configured to need acknowledgement for either the TO_NORMAL or TO_OFFNORMAL transition. The TO_FAULT bit being FALSE in Acked_Transitions is not suitable as the testable point in this test because Event_Algorithm_Inhibit does not influence detection and reporting of FAULT. Similarly, a transition from FAULT is not suitable for this test. Verify that the event is reported when the device responds to a GetEventInformation request.

Configuration Requirements: O1 is configured such that it requires at least one of its transitions to require operator acknowledgement. O1's algorithm is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T)~~,~~ is such that they can all be reported in a single GetEventInformation-ACK response.

Test Steps:
1. AT = READ Acked_Transitions
2. CHECK (AT <> (T, T, T))
3. VERIFY Acked_Transitions = (?, T, ?)
4. VERIFY Event_Algorthm_Inhibit = TRUE
5. TRANSMIT GetEventInformation
6. RECEIVE GetEventInformation-Ack,
   'List of Event Summaries' = (list of object identifiers which includes O1)
   'More Events' = FALSE

7.3.1.X6.3 Event_Algorithm_Inhibit Acknowledgement Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that event generating objects can be acknowledged when the algorithm has been inhibited.

Test_Concept: Select an event generating object O1 which is configured for event reporting, which is configured to need acknowledgement for at least one of its transitions, and its Acked_Transitions property is not (T, T, T). Verify that the IUT accepts an acknowledgment for the transition that requires it. The TO_FAULT bit in Acked_Transitions is not suitable as the testable point in this test because Event_Algorithm_Inhibit does not influence detection and reporting of FAULT. Similarly, a transition from FAULT is not suitable for this test.

Configuration Requirements: O1 is configured such that it requires at least one of its transitions to require operator acknowledgement. O1's algorithm is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T) is such that they can all be reported in a single GetEventInformation-ACK response. For this test, ES_TO_ACK is the Event_State that is to be acknowledged, TS_TO_ACK is the timestamp associated with that transition. The IUT is configured such that TD will receive a confirmed notification when O1 transitions.

Test Steps:
1. AT = READ Acked_Transitions
2. CHECK(AT <> (T, T, T))
3. VERIFY Event_Algorthm_Inhibit = TRUE
4. TRANSMIT AcknowledgeAlarm
   'Acknowledging Process Identifier' = (any valid value),
   'Event Object Identifier' = O1,
   'Event State Acknowledged' = ES_TO_ACK,
   'Time Stamp' = TS_TO_ACK,
   'Time of Acknowledgment' = (the current timestamp)
5. RECEIVE BACnet-SimpleACK-PDU
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the configured process ID),

'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(O1),
'Time Stamp' =	(the current local time or sequence number),
'Notification Class' =	(the class configured for O1),
'Priority' =	(the value configured for the transition),
'Event Type' =	(any valid event type),
'Notify Type' =	ACK_NOTIFICATION,
'To State' =	ES_TO_ACK

7. TRANSMIT BACnet-SimpleACK-PDU
8. AT2 = READ Acked_Transitions
9. CHECK(AT2 is equal to AT, except the bit associated with ES_TO_ACK is TRUE)

7.3.1.X7 Event_Algorithm_Inhibit_Ref Tests

7.3.1.X7.1 Event_Algorithm_Inhibit_Ref Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that the object referenced by Event_Algorithm_Inhibit_Ref controls Event_Algorithm_Inhibit and thus whether or not the event state detection algorithm is executed.

Test Concept: Execute test 7.3.1.X2.1 against an object O2 which supports both Event_Algorithm_Inhibit_Ref and Event_Algorithm_Inhibit and instead of writing Event_Algorithm_Inhibit, write the property referenced by Event_Algorithm_Inhibit_Ref to change the value in the Event_Algorithm_Inhibit property.

Configuration Requirements: If the IUT has no object in which the Event_Algorithm_Inhibit_Ref property is absent or can be made uninitialized, or has no object in which Event_Detection_Enable can be made TRUE, this test shall be skipped.

7.3.1.X7.2 Event_Algorithm_Inhibit Writable Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that if the Event_Algorithm_Inhibit_Ref property is absent or is uninitialized then the Event_Algorithm_Inhibit property shall be writable.

Configuration Requirements: Select an event-initiating object, O1 in which Event_Algorithm_Inhibit_Ref property is absent or is uninitialized. If the IUT has no such object, this test shall be skipped.

Test Steps:
1. WRITE Event_Algorithm_Inhibit = TRUE
2. WRITE Event_Algorithm_Inhibit = FALSE

7.3.1.X8 Reliability_Evaluation_Inhibit Tests

7.3.1.X8.1 Reliability_Evaluation_Inhibit Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that Reliability_Evaluation_Inhibit controls whether or not fault conditions are detected.

Test Concept: Select an event generating object, O1, which supports the Reliability_Evaluation_Inhibit property. With Reliability_Evaluation_Inhibit FALSE, make a fault condition exist. Verify that Reliability changes and that a notification is generated. Set Reliability_Evaluation_Inhibit to TRUE. Verify that the Reliability changes to NO_FAULT_DETECTED

and that a TO_NORMAL notification is generated. Remove the fault condition and ensure that no notification is generated. Make a fault condition exist and verify that Reliability remains NO_FAULT_DETECTED, and that no notification is generated.

Test Configuration: O1 is configured to detect and report unconfirmed events, is in the NORMAL state, and Reliability_Evaluation_Inhibit equals FALSE, so that reliability evaluation for that object is configured to detect fault conditions. If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.

Test Steps:
1. VERIFY Event_State = NORMAL
2. VERIFY Reliability = NO_FAULT_DETECTED
3. MAKE(a fault condition exist for O1)
4. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
        'Process Identifier' = (the value configured for the transition),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1,
        'Time Stamp' = (any valid timestamp),
        'Priority' = (any valid priority),
        'Event Type' = CHANGE_OF_RELIABILITY,
        'Notify Type' = ALARM | EVENT,
        'Message Text' = (any valid message text),
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = FAULT,
        'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)
5. WRITE Reliability_Evaluation_Inhibit = TRUE
6. BEFORE **Internal Processing Fail Time** + **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
        'Process Identifier' = (the value configured for the transition),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1,
        'Time Stamp' = (any valid timestamp),
        'Priority' = (any valid priority),
        'Event Type' = CHANGE_OF_RELIABILITY,
        'Notify Type' = ALARM | EVENT,
        'Message Text' = (any valid message text),
        'AckRequired' = TRUE | FALSE,
        'From State' = FAULT,
        'To State' = NORMAL,
        'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)
7. VERIFY Reliability = NO_FAULT_DETECTED
8. VERIFY Event_State = NORMAL
9. MAKE(remove the fault condition)
10. WAIT **Notification Fail Time**
11. CHECK (that the IUT did not send any event notifications for O1)
12. MAKE(a fault condition exist for O1)
13. WAIT **Notification Fail Time**
14. VERIFY Reliability = NO_FAULT_DETECTED
15. VERIFY Event_State = NORMAL
16. CHECK (that the IUT did not send any event notifications for O1)

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X8.2 Reliability_Evaluation_Inhibit Summarization Test

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that event generating objects are reported by summarization routines as needed even when the reliability evaluation has been inhibited.

Test_Concept: Select an event generating object O1 which is configured for event reporting, which is configured to require acknowledgement for TO_FAULT transition, and its Acked_Transitions property is (T, F, T). Verify that the event is reported when the device responds to a GetEventInformation request.

Configuration Requirements: O1 is configured such that it requires acknowledgement of the TO_FAULT transition, and the Acked_Transitions is (T, F, T). O1's Reliability_Evaluation_Inhibit equals TRUE, so that reliability evaluation for that object is inhibited. The number of event generating objects in the IUT that have an Event_State other than NORMAL, or which have an Acked_Transitions other than (T, T, T) is such that they can all be reported in a single GetEventInformation-ACK response.

Test Steps:
1. VERIFY Acked_Transitions = (T, F, T)
2. VERIFY Event_Algorithm_Inhibit = TRUE
3. TRANSMIT GetEventInformation
4. RECEIVE GetEventInformation-Ack,
   'List of Event Summaries' = (list of object identifiers which includes O1)
   'More Events' = FALSE

7.3.1.X9 Event_Detection_Enable Tests

7.3.1.X9.1 Event_Detection_Enable Inhibits Event Generation

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that Event_Detection_Enable enables and disables event detection in objects which are configured for event reporting.

Test Concept: Select an event generating object, O1 that is configured for event reporting. Make the object generate an event, to an offnormal state if possible, so that if the object can have a non-normal state, it enters that state early in the test. This will help detect incorrect implementations that initiate a TO_NORMAL transition when the algorithm is disabled. Set the Event_Detection_Enable property to FALSE. Verify the Event_State is NORMAL and the Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard. Repeat the process that made the object generate an event and observe that no notification messages are transmitted.

Configuration Requirements: O1 is configured to detect and report unconfirmed events and requires acknowledgments for all transitions. Event_Detection_Enable is equal to TRUE. DELAY shall represent the time delay appropriate to the transition being tested (i.e. Time_Delay for TO_OFFNORMAL, 0 for TO_FAULT and FAULT to NORMAL transitions, and either Time_Delay or Time_Delay_Normal for TO_NORMAL). For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

Test Steps:
1. VERIFY Event_Detection_Enable = TRUE
2. MAKE (a condition exist which will cause O1 to transition, to an offnormal state if possible)
3. WAIT DELAY
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = (any valid event type),
'Notify Type' = (value from the Notify_Type property configured for O1),
'Message Text' = (any valid message text),
'AckRequired' = TRUE,
'From State' = (any valid event state),
'To State' = (any event state appropriate to the event type),
'Event Values' = (any values appropriate to the event type)

5. IF Event_Detection_Enable is writable THEN
   WRITE Event_Detection_Enable = FALSE
   ELSE
   MAKE (Event_Detection_Enable to FALSE. This property is expected to be set during system configuration and is not expected to change dynamically.)
6. WAIT DELAY + **Notification Fail Time + Internal Processing Fail Time**
7. CHECK (that the IUT did not send any further event notifications for O1)
8. VERIFY Event_State = NORMAL
9. VERIFY Acked_Transitions = (T,T,T)
10. VERIFY Event_Time_Stamps = [NO_TS , NO_TS , NO_TS]
11. IF the Event_Message_Texts property exists THEN
    VERIFY Event_Message_Texts = [", ", "]
12. MAKE (a condition exist which would cause O1 to transition, if Event_Detection_Enable were TRUE)
13. WAIT DELAY + **Notification Fail Time**
14. CHECK (that the IUT did not send any event notifications for O1)
15. VERIFY Event_State = NORMAL
16. VERIFY Acked_Transitions = (T,T,T)
17. VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
18. IF the Event_Message_Texts property exists THEN
    VERIFY Event_Message_Texts = [", ", "]

Notes to Tester: This behavior can alternately be tested using the ConfirmedEventNotification service, but it is not necessary to test both.

7.3.1.X9.2 Event_Detection_Enable Inhibits FAULT

Reason for Change: New functionality added with Addendum 135-2010af. This test does not exist in 135.1-2013.

Purpose: To verify that Event_Detection_Enable disables fault reporting.

Test Concept: When the event-state-detection process is disabled via the Event_Detection_Enable, both the event algorithm and the Reliability value are ignored, and Event_State remains NORMAL. Select an event generating object, O1 that is configured for event reporting and which can be made to go into FAULT. Set the Event_Detection_Enable property to FALSE. Create a condition which would cause O1 to transition to FAULT, if Event_Detection_Enable were TRUE. Verify the Event_State is NORMAL and the Acked_Transitions, Event_Time_Stamps, and Event_Message_Texts are equal to their respective initial conditions, as mandated in the standard, and no notification messages are transmitted.

Configuration Requirements: O1 is an object capable of detecting and reporting an event for a FAULT condition, and the Event_Detection_Enable can be set to FALSE. Reliability_Evaluation_Inhibit is equal to TRUE. For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

Test Steps:
1. VERIFY Event_Detection_Enable = FALSE
2. IF Reliability is writable THEN

WRITE Reliability = (any value other than NO_FAULT_DETECTED)
ELSE
MAKE (a condition exist which would cause O1 to transition to FAULT, if Event_Detection_Enable were TRUE)

3. WAIT **Notification Fail Time**
4. CHECK (that the IUT did not send any event notifications due to this condition)
5. VERIFY Event_State = NORMAL
6. VERIFY Acked_Transitions = (T,T,T)
7. VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
8. IF Event_Message_Texts property exists THEN
   VERIFY Event_Message_Texts = [", ", "]

7.3.2 Object Specific Tests

7.3.2.4 Averaging Object Tests

An Averaging object provides a way to monitor the average, minimum, and maximum values attained by a sampled property. The datatype of the sampled property can be BOOLEAN, INTEGER, Unsigned, Enumerated, or Real. The tests in this clause shall be repeated once for each of these datatypes.

7.3.2.4.1 Reinitializing the Samples

Reason For Change: Per clarification BTL-CR-0309

Purpose: To verify that an Averaging object correctly resets the Attempted_Samples, Valid_Samples, Minimum_Value, Average_Value, and Maximum_Value when Attempted_Samples, Object_Property_Reference, Window_Interval, or Window_Samples are changed.

Test Concept: The IUT is configured with an Averaging object that is actively monitoring some property value. The sampling is reinitialized by writing to the Attempted_Samples, ~~Object_Property_Reference,~~ Window_Interval, *Window_Samples*, and ~~Window_Samples~~*Object_Property_Reference* in turn. After each reinitialization~~, the TD pauses and~~ verify~~ies~~ that new sampling has begun.

Configuration Requirements: The IUT shall be configured with an Averaging object that is actively monitoring some property value. The sampling interval shall be long enough to permit the TD to verify that the sample is properly reinitialized.

Test Steps:

[Renumber remaining steps to close the gaps for those which are now omitted.]

~~1. VERIFY Minimum_Value = (a value x: -INF < x < INF),~~
~~2. VERIFY Average_Value = (a value ≠ NaN),~~
~~3. VERIFY Maximum_Value = (a value x: Minimum_Value ≤ x < INF),~~
~~4. VERIFY Attempted_Samples = (a value x > 0),~~
~~5. VERIFY Valid_Samples = (a value x > 0),~~
6. WRITE Attempted_Samples = 0,
~~7. VERIFY Attempted_Samples = 0,~~
~~8. VERIFY Minimum_Value = INF,~~
~~9. VERIFY Maximum_Value = INF,~~
~~10. VERIFY Average_Value = NaN,~~
~~11. VERIFY Valid_Samples = 0,~~
12. WAIT (at least two sample times),
13. VERIFY Minimum_Value = (a value x: -INF < x < INF),
14. VERIFY Average_Value = (a value ≠ NaN),
15. VERIFY Maximum_Value = (a value x: Minimum_Value ≤ x < INF),
16. VERIFY Attempted_Samples = (a value x ≥ 2),
17. VERIFY Valid_Samples = (a value x ≥ 2),

18. WRITE Window_Interval = (any new value that will result in an appropriate sample time),
~~19. VERIFY Attempted_Samples = 0,~~
~~20. VERIFY Minimum_Value = INF,~~
~~21. VERIFY Maximum_Value = -INF,~~
~~22. VERIFY Average_Value = NaN,~~
~~23. VERIFY Valid_Samples = 0,~~
24. WAIT (at least two sample times),
25. VERIFY Minimum_Value = (a value x: -INF < x < INF),
26. VERIFY Average_Value = (a value ≠ NaN),
27. VERIFY Maximum_Value = (a value x: Minimum_Value ≤ x < INF),
28. VERIFY Attempted_Samples = (a value x ≥ 2),
29. VERIFY Valid_Samples = (a value x ≥ 2),
30. WRITE Window_Samples = (any new value that will result in an appropriate sample time),
~~31. VERIFY Attempted_Samples = 0,~~
~~32. VERIFY Minimum_Value = INF,~~
~~33. VERIFY Maximum_Value = -INF,~~
~~34. VERIFY Average_Value = NaN,~~
~~35. VERIFY Valid_Samples = 0,~~
36. IF (Object_Property_Reference is writable) THEN {
    WAIT (at least two sample times),
    VERIFY Minimum_Value = (a value x: -INF < x < INF),
    VERIFY Average_Value = (a value ≠ NaN),
    VERIFY Maximum_Value = (a value x: Minimum_Value ≤ x < INF),
    VERIFY Attempted_Samples = (a value x ≥ 2),
    VERIFY Valid_Samples = (a value x ≥ 2),
    WRITE Object_Property_Reference = (any new value),
    IF (Samples_are_taken_immediately) THEN {
        VERIFY Attempted_Samples = 1,
        VERIFY Minimum_Value = Average_Value,,
        VERIFY Maximum_Value = Average_Value,
        VERIFY Valid_Samples = 1
    ELSE
        VERIFY Attempted_Samples = 0,
        VERIFY Minimum_Value = INF,
        VERIFY Maximum_Value = -INF,
        VERIFY Average_Value = NaN,
        VERIFY Valid_Samples = 0

7.3.2.4.2 Managing the Sample Window

Reason For Change: Per clarification BTL-CR-0309

Purpose: To verify that an Averaging object correctly tracks the average, minimum, and maximum values attained in a sample. This includes monitoring before and after the sampling window is full.

Test Concept: An Averaging object is configured to monitor a property that can be controlled manually by the testing agent or by the TD. The TD initializes the sample and then monitors the Minimum_Value, Average_Value, Maximum_Value, Attempted_Samples, and Valid_Samples properties after each sampling interval to verify that their values are properly tracking the monitored value. This requires the ability to manipulate the values of the monitored property value and a slow enough sampling interval to permit the analysis. This continues until after the sample window is full. ~~If the IUT does not support Averaging object configuration for this Test Concept, then this test shall be omitted.~~

Configuration Requirements: The IUT shall be configured with an Averaging object used to monitor a property that can be controlled by the testing agent or by the TD. The sampling interval shall be configured to allow time to change the

monitored property value and to determine if each of the properties Minimum_Value, Average_Value, Maximum_Value, Attempted_Samples, and Valid_Samples correctly changes after each sample interval.

Test Steps:

1. WRITE Attempted_Samples = 0~~,~~
~~2. VERIFY Attempted_Samples = 0,~~
~~3. VERIFY Minimum_Value = INF,~~
~~4. VERIFY Maximum_Value = INF,~~
~~5. VERIFY Average_Value = NaN,~~
~~6. VERIFY Valid_Samples = 0,~~
2. READ StartingSample = Valid_Samples +1
~~7~~*3.* REPEAT X = (~~1~~*StartingSample* to Window_Samples + 5) DO {
    WAIT (Window_Interval / Window_Samples)
    IF (X ≤ Window_Samples) THEN
        VERIFY Attempted_Samples = X
    ELSE
        VERIFY Attempted_Samples = Window_Samples,
    VERIFY Minimum_Value = (the minimum of the monitored values so far),
    VERIFY Maximum_Value = (the maximum of the monitored values so far),
    VERIFY Average_Value = (the average of the monitored values so far),
    IF (X ≤ Window_Samples) THEN
        VERIFY Valid_Samples = X
    ELSE
        VERIFY Valid_Samples = Window_Samples

7.3.2.9 Command Object Tests

7.3.2.9.7 Write While In_Process is TRUE Test.

Reason for Change: Updated with new error codes for Protocol_Revision >= 10.

Purpose: To verify that an action list continues to completion if a second action list is commanded while In_Process is TRUE and that the second action list is not executed.

Test Concept: The IUT is configured with two action lists that include a sequence of externally visible outputs with post delays for each action. The TD triggers the first action list. The external outputs are observed in order to trigger the second action list during the post delay of the first list. The TD triggers the second action list. The external outputs are observed to verify that the second action list is not executed. If the IUT does not support Post Delay, then this test shall be omitted. If the IUT does not support action list configuration, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object having two distinct action lists, X and Y, that include writing to a sequence of externally visible outputs. There shall be a post delay between writes to the externally visible outputs that is long enough for the tester to observe the delay (This ensures In_Process remains TRUE long enough to command the second action list).

Test Steps:

1. WRITE Present_Value = X
~~2. RECEIVE Simple-ACK-PDU~~
2. WRITE Present_Value = Y
3. *IF (Protocol_Revision exists and Protocol_Revision >= 10) THEN*
    RECEIVE BACnet-Error-PDU
        Error Class = OBJECT,

Error Code = *BUSY*
ELSE
RECEIVE *(BACnet-Error PDU*
Error Class = *OBJECT,*
Error Code = *BUSY)*
|
*(*BACnet- Error-PDU
Error Class = SERVICES,
Error Code = SERVICE_REQUEST_DENIED | OTHER*)*

4. CHECK (that the externally visible actions of X took place)
5. CHECK (that the externally visible actions of Y did not take place)
6. VERIFY In_Process = FALSE,
7. VERIFY All_Writes_Successful = TRUE

7.3.2.10 Device Object Tests

These are the tests for the Device object. Other tests for functionality of the Device object are covered by tests for the application service or special functionality to which they correspond.

7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test

Reason for Change: Reprinted here to show all the steps. Steps 15-21 were left out of the 135.1-2013 publication.

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. *VERIFY Active_COV_Subscriptions = {{{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)}}
16. *TRANSMIT SubscribeCOV-Request,*
    'Subscriber Process Identifier' = P 3 ,
    'Monitored Object Identifier' = O 3
17. *RECEIVE BACnet-SimpleACK-PDU*
18. *VERIFY Active_COV_Subscriptions = {{{TD, P 2 }, {O 2 , Present_Value}, FALSE, (a value less than L 2), (a valid Increment if the property is REAL) }}*
19. *TRANSMIT SubscribeCOV-Request,*
    'Subscriber Process Identifier' = P 2 ,
    'Monitored Object Identifier' = O 2
20. *RECEIVE BACnet-SimpleACK-PDU*
21. *VERIFY Active_COV_Subscriptions = { }*

7.3.2.10.6 Successful Increment of the Database_Revision Property after Changing the Object_Identifier Property of an Object

Reason for change: To correct a cut&paste&forgot-to-revise typo in the Test Concept.

Purpose: To verify that the Database_Revision property of the Device object increments after changing the Object_Identifier property of an object. If the Object_Identifier property of an object cannot be changed, this test shall be omitted.

Test Concept: The Database_Revision property of the Device object is read. An object's ~~name~~*Object_Identifier property* is changed. The Database_Revision property of the Device object is read again to verify that it incremented.

Configuration Requirements: none.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the Device object),
   'Property Identifier' = Database_Revision
2. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the Device object),
   'Property Identifier' = Database_Revision,
   'Property Value' = (any value = initial value)
3. MAKE (the Object_Identifier property of an object change)
4. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the Device object),
   'Property Identifier' = Database_Revision
5. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the Device object),
   'Property Identifier' = Database_Revision,
   'Property Value' = (greater than initial value)

7.3.2.10.X2 Max_Segments_Accepted at least the minimum

Reason for Change: There is no SSPC proposal for this change.

Purpose: To verify that the IUT implements the Max_Segments_Accepted property value when it does support segmentation.

Configuration Requirements: If the IUT cannot be configured to support segmentation, then this test shall be skipped.

Test Steps:

1. VERIFY (Max_Segments_Accepted > 1)

7.3.2.13 Global Group

7.3.2.13.X1 Global Group Present_Value, Out_Of_Service and Status_Flags Test

Reason for Change: New Tests for Global Group object type.

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

Purpose: This test verifies the interrelationship between the Present_Value, Out_Of_Service and Status_Flags properties of a Global Group object.

Test Concept: Verify the Present_Value stops updating when Out_Of_Service is TRUE.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group_Members property containing a member M1 at index N1 that has a value that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Test Steps:

1. MAKE (Out_Of_Service = TRUE)
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = {?, ?, FALSE, TRUE}
4. X1 = READ Present_Value, ARRAY_INDEX = N1
5. MAKE (M1 value change)
6. WAIT (W1)
7. X2 = READ Present_Value, ARRAY_INDEX = N1
8. VERIFY X1 = X2

7.3.2.13.X2 Reliability MEMBER_FAULT Test

Reason for Change: New Tests for Global Group object type.

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

Purpose: This test case verifies the FAULT flag of the Member_Status_Flags is TRUE and the Reliability property is equal to MEMBER_FAULT when a member of the Group_Members property goes into FAULT.

Test Concept: Force a member of the Group_Members property to enter a Fault condition and verify the Member_Status_Flags FAULT flag equals TRUE and Reliability equals MEMBER_FAULT.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group_Members property containing a member M1 at index N1 that has a value that can be made to indicate a fault condition (see Notes To Tester). The Out_Of_Service property of the Global Group object must remain FALSE throughout the test. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Test Steps:

1. MAKE (M1 Status_Flags = {?, TRUE, ?, ?})
2. WAIT (W1)
3. VERIFY Member_Status_Flags = {?, TRUE, ?, ?}
4. VERIFY Reliability = MEMBER_FAULT

Notes to Tester: Member_Status_Flags FAULT flag will the TRUE and the Reliability property will change to MEMBER_FAULT when a member of the Group_Members property goes into fault.

7.3.2.13.X3 Reliability COMMUNICATION_FAILURE Test

Reason for Change: New Tests for Global Group object type.

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

Purpose: This test case verifies that the Member_Status_Flags FAULT flag will remain FALSE while the Reliability property is COMMUNICATION_FAILURE.

Test Concept: Force a member of the Group_Members property to stop communicating and verify the Reliability property equals COMMUNICATION_FAILURE and the Member_Status_Flags FAULT flag remains FALSE.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group_Members containing a member M1 at index N1that can be made to discontinue communications and also respond with an error such as OBJECT/UNKNOWN_OBJECT. (See Notes To Tester). The Out_Of_Service property of the Global Group object must remain FALSE throughout the test. W1 is the maximum time it takes for the Global Group to receive an update from M1.

Test Steps:

1. MAKE (M1 fail (communications or error))
2. WAIT (W1)
3. VERIFY Reliability = COMMUNICATION_FAILURE
4. VERIFY Member_Status_Flags = {?, FALSE, ?, ?}

Notes to Tester: Reliability will change to COMMUNICATION_FAILURE when a member is no longer able to communicate its Status_Flags property. This can occur when the device goes offline or the object is deleted within the device.

7.3.2.13.X4 Present_Value Tracking and Reliability Test

Reason for Change: New Tests for Global Group object type.

Dependencies: ReadProperty Service Execution Tests, 9.18

Purpose: This test verifies that the Global Group object continues to update its Present_Value independent of the state of the Reliability property.

Test Concept: While the Reliability property is not NO_FAULT_DETECTED verify the Present_Value continues to update.

Configuration Requirements: The IUT shall be configured with a Global Group object with its Reliability not equal to NO_FAULT_DETECTED and a Group_Members member M1 at index N1 that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

1. VERIFY Reliability <> NO_FAULT_DETECTED
2. MAKE (M1 = X1)
3. WAIT (W1)
4. X2 = READ Present_Value, ARRAY_INDEX = N1
5. VERIFY X1 = X2

Note to Tester: Reliability will change to COMMUNICATION_FAILURE when a member is no longer able to communicate its Status_Flags property. This can occur when the device goes offline or the object is deleted within the device. Also, the Reliability property will change to MEMBER_FAULT when a member of the Group_Members property goes into fault.

7.3.2.13.X5 Present_Value Tracking Test

Reason for Change: New Tests for Global Group object type.

Dependencies: ReadProperty Service Execution Tests, 9.18

Purpose: This test verifies that the Global Group object tracks the value of the monitored properties value and data type.

Test Concept: Make a member of the Group_Members property change value and verify the Present_Value updates to match that value.

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group_Members containing a member M1 at index N1 of the specified data type that can be changed. W1 is the maximum time it takes for the Global Group to receive an update from M1.

1. MAKE (M1 = X1)
2. WAIT (W1)
3. X2 = READ Present_Value, ARRAY_INDEX = N1
4. VERIFY X1 = X2

7.3.2.13.X6 COVU_Period and COVU_Recipient Zero Test

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

Purpose: To verify that object O1 does not initiate UnconfirmedCOVNotification service requests when COVU_Period is zero or COVU_Recipient contains an empty list.

Test Concept: Configure O1 to produce unsubscribed UnconfirmedCOVNotifications, set COVU_Period to zero and and attempt to produce unsubscribed UnconfirmedCOVNotifications. Repeat with COVU_Recipients containing an empty list.

Test Steps:
1. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)
2. BEFORE Notification Fail Time
   RECEIVE UnconfirmedCOVNotification-Request,
   DESTINATION = (any valid address),
   'Subscriber Process Identifier' = 0,
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = 0,
   'List of Values' = (any valid set of values)
3. MAKE (COVU_Period = 0)
4. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)
5. WAIT Notification Fail Time times 2
6. CHECK (Verify that O1 has not transmitted an UnconfirmedCOVNotification-Request.)
7. MAKE (COVU_Period <> 0)
8. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)
9. BEFORE Notification Fail Time
   RECEIVE UnconfirmedCOVNotification-Request,
   DESTINATION = (any valid address),
   'Subscriber Process Identifier' = 0,
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = 0,
   'List of Values' = (any valid set of values)
10. MAKE (COVU_Recipient an empty list)
11. MAKE (O1 issue an unsubscribed UnconfirmedCOVNotification)
12. WAIT Notification Fail Time times 2
13. CHECK (Verify that O1 has not transmitted an UnconfirmedCOVNotification-Request.)

7.3.2.21 Notification Class Object Tests

This section was renumbered in 135.1-2007 to 7.3.2.21. This was section 7.3.2.20 in 135.1-2003.

7.3.2.21.3 Recipient_List Tests

7.3.2.21.3.1 ValidDays Test

Reason for Change: Updated Test Concept to include changes from 135-2010af.

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22; TimeSynchronization Service Execution Tests, 9.30; *UTCTimeSynchronization Service Execution Tests, 9.31*.

BACnet Reference Clause: 12.21.8.

Purpose: To verify the operation of the Valid Days parameter of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to it. The Recipient_List of the Notification Class object shall contain a single recipient with the Valid Days parameter configured so that at least one day is TRUE and at least one day is FALSE. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL. The tester verifies that if the local date is one of the valid days a notification message is transmitted and the if local date is not a valid day then no notification message is transmitted. *For devices of protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all days (Valid Days set to all Days), this test shall be omitted. For devices of protocol revision 13 or higher that implement a writeable Recipient_List property for all instances of Notification Class objects, and exclusively accept all days as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The event-generating object shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE and at least one day of the week has a value of FALSE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions.

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days, converted to UTC)) |
   MAKE (the local date and time = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days))
2. WAIT (Time_Delay + **Notification Fail Time**)
3. VERIFY Event_State = NORMAL
4. IF (X is writable) THEN
   WRITE X = (a value that is OFFNORMAL)
   ELSE
   MAKE (X have a value that is OFFNORMAL)
5. WAIT (Time_Delay)
6. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (the current local time),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = (any valid event type),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (values appropriate to the event type)

7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY Event_State = OFFNORMAL
9. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To time in the BACnet Destination that corresponds to one of the invalid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the invalid days, converted to UTC)) |
   MAKE (the local date and time = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the invalid days))
10. IF (X is writable) THEN
    WRITE X = (a value that is NORMAL)
    ELSE
    MAKE (X have a value that is NORMAL)
11. WAIT (Time_Delay + **Notification Fail Time**)
12. CHECK (verify that no notification message was transmitted)

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3.2 FromTime and ToTime Test

Reason for Change: Incorporated changes from Addendum 135-2010af.

Dependencies: ValidDays Test, 7.3.2.21.3.1; ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30; *UTCTimeSynchronization Service Execution Tests, 9.31*.

BACnet Reference Clause: 12.21.8.

Purpose: To verify the operation of the From Time and To Time parameters of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The case where the local date and time fall within the window defined by the From Time and To Time parameters is covered by the ValidDays test in 7.3.2.21.3.1. This test uses the same IUT configuration and sets the local time to a value that is one of the ValidDays but outside of the window defined by the From Time and To Time parameters. The objective is to verify that an event notification message is not transmitted when the event is triggered. *For devices of protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all times (From Time set to 00:00:00.0, To_Time set to 23:59:59.90), this test shall be omitted. For devices of protocol revision 13 or higher that implement a writeable Notification Class Recipient_List property for all instances of Notification Class objects, and exclusively accept all times as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The configuration requirements are identical to the requirements in 7.3.2.21.3.1.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
   'Time' = (any time outside the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days)) |
   (TRANSMIT UTCTimeSynchronization-Request,
   'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that corresponds to one of the valid days, converted to UTC)) |
   MAKE (the local date and time = (any time outside the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days))
2. WAIT (Time_Delay + **Notification Fail Time**)
3. VERIFY Event_State = NORMAL
4. IF (X is writable) THEN
   WRITE X = (a value that is OFFNORMAL)
   ELSE
   MAKE (X have a value that is OFFNORMAL)
5. WAIT (Time_Delay + **Notification Fail Time**)
6. CHECK (verify that no notification message was transmitted)

7.3.2.21.3.3 IssueConfirmedNotifications Test

Reason for Change: Updates per Addendum 135-2010af.

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.8.

Purpose: To verify that ConfirmedEventNotification messages are used if the Issue Confirmed Notifications parameter has the value TRUE and UnconfirmedEventNotification messages are used if the value is FALSE. If the IUT does not support both confirmed and unconfirmed event notifications this test may be omitted. *For devices of protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is a value of FALSE for the IssueConfirmedNotifications component in all instances, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with two or more instances of the Notification Class object and event-generating objects that are linked to the Notification Class objects. The event-generating objects may be objects that support intrinsic reporting or they may be Event Enrollment objects. The event-generating objects shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating objects shall be configured to be in a NORMAL event state at the start of the test. One Notification Class object, N_1, shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, N_2, shall be configured with Issue Confirmed Notifications equal to FALSE. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions. The local date and time shall be configured to be within the window defined by From Time and To Time on one of the ValidDays.

In the test description below "X_1" and "X_2" are used to designate the event-triggering property linked to Notification objects "N_1" and "N_2" respectively.

Test Steps:

1. VERIFY (the event-generating object linked to N_1), Event_State = NORMAL
2. VERIFY (the event-generating object linked to N_2), Event_State = NORMAL

3. WAIT (Time_Delay + **Notification Fail Time**)
4. IF (X_1 is writable) THEN
   WRITE X_1 = (a value that is OFFNORMAL)
   ELSE
   MAKE (X_1 a value that is OFFNORMAL)
5. WAIT (Time_Delay)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object linked to N_1),
'Time Stamp' =	(the current local time),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	(any valid event type),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	(values appropriate to the event type)

7. IF (X_2 is writable) THEN
   WRITE X_2 = (a value that is OFFNORMAL)
   ELSE
   MAKE (X_2 a value that is OFFNORMAL)
8. WAIT (Time_Delay)
9. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the event-generating object linked to N_2),
'Time Stamp' =	(the current local time),
'Notification Class' =	(the class corresponding to the object being tested),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	(any valid event type),
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	(values appropriate to the event type)

Notes to Tester: If the Recipient_List is writable and the Issue Confirmed Notifications can be changed then this test can be performed using only one Notification Class object by writing to the Recipient_List in order to change between confirmed and unconfirmed notifications. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3.4 Transitions Test

Reason for change: Incorporated changes for addendum 135-2010af.
Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.8.

Purpose: To verify that notification messages are transmitted only if the bit in the Transitions parameter corresponding to the event transition is set.

Test Concept: The IUT is configured such that the Transitions parameter indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Transitions parameter has a value of TRUE. *For devices of protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all transitions (all bits in Transitions set to TRUE), this test shall be omitted. For devices of protocol revision 13 or higher that implement a writeable Notification Class Recipient_List property for all instances of Notification Class objects, and exclusively accept all transitions as the only permitted configuration, this test shall be omitted.*

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The event-generating object shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Transitions parameter shall be configured with a value of TRUE for either the TO-OFFNORMAL transition or the TO-NORMAL transition and the other event transition shall have a value of FALSE. The local time shall be configured such that it represents one of the valid days in the window specified by From Time and To Time.

In the test description below, “X” is used to designate the event-triggering property.

1. VERIFY Event_State = NORMAL
2. WAIT (Time_Delay + **Notification Fail Time**)
3. IF (X is writable) THEN
    WRITE X = (a value that is OFFNORMAL)
   ELSE
    MAKE (X have a value that is OFFNORMAL)
4. WAIT (Time_Delay)
5. BEFORE **Notification Fail Time**
    IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' = (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the event-generating object configured for this test),
            'Time Stamp' = (the current local time),
            'Notification Class' = (the class corresponding to the object being tested),
            'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' = (any valid event type),
            'Notify Type' = EVENT | ALARM,
            'AckRequired' = TRUE | FALSE,
            'From State' = NORMAL,
            'To State' = OFFNORMAL,
            'Event Values' = (values appropriate to the event type)
    ELSE
        CHECK (verify that the IUT did not transmit an event notification message)
6. VERIFY Event_State = OFFNORMAL
7. IF (X is writable) THEN
    WRITE X = (a value that is NORMAL)
   ELSE
    MAKE (X have a value that is NORMAL)
8. WAIT (Time_Delay)
9. BEFORE **Notification Fail Time**
    IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN
        RECEIVE ConfirmedEventNotification-Request,

```
            'Process Identifier' =            (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       (the event-generating object configured for this test),
            'Time Stamp' =                    (the current local time),
            'Notification Class' =            (the class corresponding to the object being tested),
            'Priority' =                      (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =                    (any valid event type),
            'Notify Type' =                   EVENT | ALARM,
            'AckRequired' =                   TRUE | FALSE,
            'From State' =                    OFFNORMAL,
            'To State' =                      NORMAL,
            'Event Values' =                  (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY Event_State = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
        MAKE (the event-triggering object change to a fault condition)
        BEFORE Notification Fail Time
            IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN
                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =            (any valid process ID),
                    'Initiating Device Identifier' =  IUT,
                    'Event Object Identifier' =       (the event-generating object configured for this test),
                    'Time Stamp' =                    (the current local time),
                    'Notification Class' =            (the class corresponding to the object being tested),
                    'Priority' =                      (the value configured to correspond to a TO-FAULT transition),
                    'Event Type' =                    (any valid event type),
                    'Notify Type' =                   EVENT | ALARM,
                    'AckRequired' =                   TRUE | FALSE,
                    'From State' =                    NORMAL,
                    'To State' =                      FAULT,
                    'Event Values' =                  (values appropriate to the event type)
            ELSE
                CHECK (verify that the IUT did not transmit an event notification message)
        VERIFY Event_State = FAULT
        }
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3.5 Recipient_List Property Supports Device Identifier Recipients Test

Reason for Change: Changes per Addendum 135-2010af.

Purpose: To verify that the Recipient_List property of the Notification Class object supports the device form of the Recipient component and that the IUT is able to associate a MAC address with the Device Identifier. The intent is to ensure that the IUT is able to locate the specified alarm recipient and send notification to the specified recipient. This test shall be run if the IUT's Notification Class object's Recipient_List property supports the BACnet object identifier form of BACnetRecipient.

Test Concept: The tester shall select a single event-generating object E in the IUT that references Notification Class object N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a Device Identifier, D, for a device that the IUT is not already aware of. The TD, acting as device D, shall be located on a different network than the IUT to ensure that the IUT is capable of binding to recipients located on any network. *For devices of*

protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects and there is an address form of the Recipient component in all instances, this test shall be omitted.

Configuration Requirements:The TD shall be configured so that it does not execute WhoHas.

Test Steps:

1. WRITE N.RecipientList = ({all days, all times, D, any process ID, FALSE, all transitions})
2. MAKE (the event generating object, E, transition)
3. BEFORE Notification Fail Time plus the amount of time the IUT takes to perform device discovery
    RECEIVE UnconfirmedEventNotification-Request,
        'Process Identifier' = (the valid process ID from step 1),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = E,
        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (N's instance),
        'Priority' = (any valid priority),
        'Event Type' = (any valid event type),
        'Notify Type' = ALARM | EVENT,
        'AckRequired' = TRUE | FALSE,
        'From State' = (any valid event state),
        'To State' = (any valid event state),
        'Event Values' = (values appropriate to the event type)

Notes to Tester: The IUT is expected to initiate one or more range-restricted WhoIs requests after the modification of the Recipient_List but before the sending of the notification. The IUT might also need to perform other network discovery operations. Given that there are multiple approaches to the use of WhoIs for device discovery, the test only focuses on the IUT's ability to find device D and not on the specifics or timing of the WhoIs requests.

7.3.2.21.3.6 Recipient_List Property Supports Network Address Recipients

Reason for Change: Changes per Addendum 135-2010af.

Purpose: To verify that the Recipient_List property of the Notification Class object supports the address form of the Recipient component. The intent is to ensure that the IUT is able to send notifications to the specified recipient.

Test Concept: The tester shall select a single event-generating object E in the IUT that references Notification Class object N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a BACnetAddress A, where A is a unicast or is a local, remote, or global broadcast address. *For devices of protocol revision 13 or higher that implement a read-only Recipient_List property for all instances of Notification Class objects and there is a Device Identifier form of the Recipient component in all instances, this test shall be skipped.*

Test Steps:

1. WRITE N.RecipientList = ({all days, all times, A, any process ID, FALSE, all transitions})
2. MAKE (the event generating object, E, transition)
3. BEFORE Notification Fail Time
    RECEIVE
        DESTINATION = A,
        UnconfirmedEventNotification-Request,
        'Process Identifier' = (the valid process ID from step 1),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = E,
        'Time Stamp' = (the current local time),
        'Notification Class' = (N's instance),
        'Priority' = (any valid priority),

'Event Type' =	(any valid event type),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	(any valid event state),
'To State' =	(any valid event state),
'Event Values' =	(values appropriate to the event type)

7.3.2.21.3.X7 Recipient_List non-volatility test

Reason for Change: New test per Addendum 135-2010af.

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

BACnet Reference Clauses: 12.21.8

Purpose: This test case verifies that a Notification Class object Recipient_List is maintained through a power failure and device restart.

Test Concept: Write the Recipient_List of a Notification Class object and restart the IUT device by issuing a ReinitializeDevice – WARMSTART service request and by temporarily removing power. When the device has resumed operation after each restart, verify that the Recipient_List contains the values that were written. This test is only applied to IUT devices that have writable Notification Class object Recipient_List properties. If the device only accepts Recipient_List values that include Valid Days = (1, 1, 1, 1, 1, 1, 1), From Time = 00:00:00.00, To Time = 23:59:59.99, and Transitions = (True, True, True), then those values shall be used in this test. If the IUT accepts Recipient_List sizes greater than one, then at least two different BACnetDestination values shall be written in the list. If the device does not support ReinitializeDevice WARMSTART, then only the removal of power will be tested.

Configuration Requirements: If the Recipient_List of a Notification Class object is read-only in all instances, this test shall be skipped.

Test Steps:

```
1.  MAKE (Recipient_List consist of entries at least one of which is different from what it has)
2.  IF  (ReinitializeDevice is supported) THEN
    {   TRANSMIT ReinitializeDevice-Request
            Reinitialized State of Device = WARMSTART
            Password = (any valid password)
        RECEIVE BACnet-Simple-ACK-PDU
        CHECK (Did the IUT perform a WARMSTART reboot?)
        VERIFY RecipientList = (the entries with which it was configured)
    }
3.  MAKE (the IUT power cycle to reinitialize)
4.  VERIFY RecipientList = (the entries with which it was configured)
```

7.3.2.21.3.X8 Read-only Recipient_List with internal Notification Forwarder objects

Reason for Change: New test per Addendum 135-2010af.

Purpose: This test case verifies that a read-only Notification Class object Recipient_List is configured with the only content designed for internal Notification Forwarder objects.

Test Concept: This test is only applied to IUT devices that have read-only Notification Class object Recipient_List properties and are capable of containing a Notification Forwarder object. The Notification Class Recipient_List is read and checked to insure all entries in the Recipient_List refer to the local device.

Test Steps:

1. READ RL = Recipient_List
2. CHECK (All Recipients in RL are equal to IUT)

7.3.2.21.3.X9 Read-only Recipient_List for external Notification Forwarder Objects

Reason for Change: New test per Addendum 135-2010af.

Purpose: This test case verifies that a read-only Notification Class object Recipient_List is configured with the content designed for external Notification Forwarder objects.

Test Concept: Read the Recipient_List of the Notification Class objects and check that the length is 1, the Recipient is local broadcast, Valid Days are all days, From Time and To Time are the entire day, Process Identifier is 0, Issue Confirmed Notifications parameter is False and Transitions is set to all transitions. This test is only applied to IUT devices that have read-only Notification Class object Recipient_List properties, and which do not contain internal Notification Forwarder objects.

Test Steps:

1. VERIFY Recipient_List = {
```
(1, 1, 1, 1, 1, 1, 1)--Valid Days
    00:00:00.0          --From Time
    23:59:59.99         --To Time
    (BACnetAddress: network-number = 0, zero length mac-address)
    0                   --Process Identifier
    False               --Issue Confirmed Notifications
    (True, True, True)  --Transitions
}
```

7.3.2.22 Program Object Tests

~~The Program object makes parameters of a custom program network visible. Since BACnet does not define the functionality of the program there are no standard tests to verify this functionality.~~ *The Program object utilizes parameter control through its writable Program_Change property.*

7.3.2.22.1 Program_Change property test

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

Purpose: To verify writability of Program_Change property.

Test Concept: The Program_Change property is set to a value other than READY and then it and the Program_State property are verified to update correctly.

Configuration Requirements: The Program_Change property of the program object being tested shows a value of READY.

Test Steps:
1. VERIFY Program_Change = READY
2. WRITE Program_Change = (a value other than READY)
3. WAIT (for the processing to consume that value written to Program_Change)
4. VERIFY Program_Change = READY
5. VERIFY Program_State = the new state reflected, based upon value written to Program_Change in step 2.

Notes to Tester: In step 2, depending on the current Program_State, and the implementation, certain requested values for Program_Change may be invalid and would return a Result(-) if an attempt were made to write them.

7.3.2.23 Schedule Object Tests

This section was renumbered in 135.1-2007 to 7.3.2.23. The old reference was 7.3.2.22

7.3.2.23.6 Weekly_Schedule Restoration Test

Reason for Change: Corrected the Configuration Requirements to allow the test to be executed on devices greater than or equal to Protocol_Revision 4.

Dependencies: ReadProperty Service Execution Tests, 9.18; ReinitializeDevice Service Execution Tests, 9.27; TimeSynchronization Service Execution Tests, 9.30; UTCTimeSynchronization Service Execution Tests, 9.31.

BACnet Reference Clause: 12.24.4, 12.24.7, 12.24.9.

Purpose: To verify the restoration behavior in a Weekly_Schedule.

Test Concept: The IUT is configured with a Schedule object containing a Weekly_Schedule with a BACnetDailySchedule that has multiple BACnetTimeValue entries that do not include the time 00:00. There shall be no Exception_Schedule that overrides this Weekly_Schedule during the date and time used for this test. The local date and time are changed to a value between 00:00 and the first entry in the BACnetDailySchedule. Present_Value is read to verify that it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4. The IUT is reset and the Present_Value is checked again to verify that it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4.

Configuration Requirements: The IUT shall be configured with a Schedule object that contains a Weekly_Schedule that has more than one scheduled write operation for a particular day. None of the write operations shall be scheduled for time 00:00 and there shall be no higher priority coincident BACnetSpecialEvents. In the test description D_1 represents a time between 00:00 and the time of the first scheduled write operation in the BACnetDailySchedule. V_{last} represents the value that is scheduled to be written in the last BACnetTimeValue pair for the day. ~~This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.~~

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D_1) |
   (TRANSMIT UTCTimeSynchronization-Request 'Time' = D_1) |
   MAKE (the local date and time = D_1)
2. WAIT **Schedule Evaluation Fail Time**
3. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   VERIFY Present_Value = Schedule_Default
   ELSE
   VERIFY Present_Value = V_{last}
4. IF (ReinitializeDevice execution is supported) THEN
   TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = WARMSTART,
   'Password' = (any valid password)
   RECEIVE BACnet-Simple-ACK-PDU
   ELSE
   MAKE (the IUT reinitialize)
5. CHECK (Did the IUT perform a WARMSTART reboot?)
6. WAIT **Schedule Evaluation Fail Time**
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   VERIFY Present_Value = Schedule_Default
   ELSE

VERIFY Present_Value = V_{last}

7.3.2.23.10 Schedule Object Protocol_Revision 4 Tests

7.3.2.23.10.3 Revision 4 Exception_Schedule Property Tests

Reason for Change: No tests existed for revision 4 functionality. The change is in SED-006.

7.3.2.23.10.3.8 Revision 4 Event Priority Test

Reason for Change: Added 'Notes to Tester' for clarity.

Dependencies: ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30.

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a BACnetSpecialEvent of a higher priority takes precedence over one of lower priority when both are active at the same time, and that it relinquishes to the lower priority.

Configuration Requirements: The IUT shall be configured with a Schedule object containing two or more BACnetSpecialEvents, all active on the same date, with different eventPriority values (if possible, all 16 priority levels should be represented), and with overlapping BACnetTimeValue entries distributed thus: the entry with the lowest priority shall have the earliest time-value pair (D_1) with a non-NULL value, and the last time-value pair (D_N) with a NULL value; the next higher priority shall have a time-value pair D_2 occurring after D_1 with a different non-NULL value, and a time-value pair D_{N-1} with a NULL value and occurring before D_N; and so on. The result is that the time-value pairs shall be ordered chronologically thus: D_1, D_2, D_3, ..., D_{N-1}, D_N. An example of such a configuration testing five priority levels is shown in Table 7-11.

Table 7-11. Example of event and value prioritization

Event	Time:								
Priority:	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
1	-	-	-	-	V_5	NULL	-	-	-
2	-	-	-	V_4	-	-	NULL	-	-
3	-	-	V_3	-	-	-	-	NULL	-
4	-	V_2	-	-	-	-	-	-	NULL
5	V_1	-	-	-	-	-	-	-	-
Present_Value:	V_1	V_2	V_3	V_4	V_5	V_4	V_3	V_2	V_1

Note: Each event priority in the table above represents 1 BACnetSpecialEvent. The BACnetSpecialEvent should contain the time value pairs listed in the table (Dx, Vx). There should be only 1 BACnetSpecialEvent per priority for this test.

Test Steps:

1. REPEAT D = (the times in the configured time-value pairs with non-NULL values) DO {
   (TRANSMIT TimeSynchronization-Request, 'Time' = D) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D converted to UTC) |
   MAKE (the local date and time = D)
   WAIT Schedule Evaluation Fail Time
   VERIFY Present_Value = (the value corresponding to the time D)
   }
2. REPEAT D = (the times in the configured time-value pairs with NULL values,
   except the final DN) DO {
   (TRANSMIT TimeSynchronization-Request, 'Time' = D) |
   (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D converted to UTC) |
   MAKE (the local date and time = D)

WAIT Schedule Evaluation Fail Time
VERIFY Present_Value = (the non-NULL value corresponding to the priority lower than that associated with D)
}

Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6. The Priority parameter for WriteProperty-Request may be left out if the target property is a standard property of a standard object for which commandability is not an option.

7.3.2.24 Log Object Tests

This section was renumbered in 135.1-2007 to 7.3.2.24. The old section number was 7.3.2.23.

7.3.2.24.4 Log_Interval Test

Reason for Change: The Configuration Requirements are enhanced, and a Notes to Tester is added.

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

Purpose: To verify that the logging period is controlled by Log_Interval.

Test Concept: The logging object is configured to acquire data by polling. Polling is done at two different intervals, defined by Log_Interval, with about 10 records acquired at each rate. The timestamps of the records are inspected to verify the polling rate.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE. Enable shall be set to TRUE. *Logging_Type is not equal to TRIGGERED.* Non-zero values shall be chosen for Log_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

1. WRITE Log_Interval = (some non-zero value)
2. WRITE Record_Count = 0
3. WAIT (**Internal Processing Fail Time** + 10* Log_Interval hundredths-seconds)
4. VERIFY (Log_Buffer record timestamp intervals, on average, are as written in step 1)
5. WRITE Log_Interval = (a non-zero value different from the one written in step 1)
6. WRITE Record_Count = 0
7. WAIT (**Internal Processing Fail Time** + 10* Log_Interval hundredths-seconds)
8. VERIFY (Log_Buffer record timestamp intervals, on average, are as written in step 5)

Notes to tester: The step 1 write of Logging_Interval to a non-zero value will make a change in Logging_Type from COV to POLLED, if Logging_Type was initially COV.

7.3.2.24.13 Log-Status Test

Reason for Change: The test here supercedes the version in 135.1-2013, with a completely different, less prescriptive approach.

Dependencies: ReadRange Service Execution Tests, 9.21; WriteProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.25.14, *12.27.13, 12.30.19*

Purpose: To verify proper logging of log-disabled and buffer-purged events.

Test Concept: The buffer is cleared. Then the Enable property is changed and it is verified that the Record_Count property is changed and it is verified that the status entry is made correctly in the Log_Buffer. The Record_Count is also set to zero while the Enable property is FALSE and it is verified that the buffer-purged event is recorded into the Log_Buffer.

Test Configuration: The logging object is configured to acquire data by whatever means available. Configure the logging such that the entire test may be run without the trend buffer overflowing.

Test Steps:

~~1. WRITE Enable = FALSE~~
~~2. WRITE Record_Count = 0~~
~~3. VERIFY Record_Count = 1~~
~~4. TRANSMIT ReadRange~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Index' = 1,~~
~~'Count' = 1~~
~~5. RECEIVE ReadRange-Ack~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Result Flags' = (True, True, False),~~
~~'Item Count' = 1~~
~~'Item Data' = ((a buffer purged record))~~
~~6. WRITE Enable = TRUE~~
~~7. WRITE Enable = FALSE~~
~~8. TRANSMIT ReadRange~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Index' = 1,~~
~~'Count' = 2~~
~~9. RECEIVE ReadRangeAck~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Result Flags' = (True, False, False),~~
~~'Item Count' = 2~~
~~'Item Data' = ((a buffer purged record), (a log-enable record))~~
~~10. TRANSMIT ReadRange~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Time' = (2154-12-31, 23:59:59.99),~~
~~'Count' = 1~~
~~11. RECEIVE ReadRangeAck~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Result Flags' = (False, True, False),~~
~~'Item Count' = 1~~
~~'Item Data' = ((a log-disable record))~~

1. WRITE Enable = FALSE
2. WRITE Record_Count = 0
3. VERIFY (Log_Buffer contains 1 entries, and it is the buffer-purged event)
4. WRITE Enable = TRUE
5. WRITE Enable = FALSE
6. VERIFY (Record_Count => 3 and the first entry is the buffer-purged event, the second entry is the log-enable TRUE event and the last entry is the log-enable FALSE event)

Notes to Tester: When the IUT's Protocol_Revision < 7, the length of BACnetLogStatus shall be 2; otherwise, it shall be 3.

7.3.2.24.14 Time_Change Test

Reason for Change: The test here supercedes the version in 135.1-2013, with a completely different, less prescriptive approach. Addendum 135-2008x-2 Clarify Trend Log Time Stamp.

Purpose: To verify proper logging of time-change events in the log buffer

Test Concept: Change the clock in the device and verify that a record is logged indicating the number of seconds that the clock changed ~~by or indicating zero if unknown~~. This test shall be skipped if the device does not support the Local_Time property in the device object or there is no way to change the time in the device.

Configuration Requirements: The log object is configured to acquire data by whatever means available. ~~The Log_Buffer should be cleared such that the Record_Count is 0.~~ Configure the logging such that the entire test may be run without the trend buffer overflowing.

Test Steps:

~~1. WRITE Enable = FALSE~~
~~2. WRITE Record_Count = 0~~
~~3. READ currentTime = (Device Object of device that contains the log object), Local_Time~~
~~4. WRITE Enable = TRUE~~
~~5. MAKE(the time change on the device by deltaTime where deltaTime >= 1 hour)~~
~~6. WRITE Enable = FALSE~~
~~7. READ N = Record_Count~~
~~8. REPEAT X = (N down through 1) DO {~~
~~TRANSMIT ReadRange~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Index' = X,~~
~~'Count' = 1~~
~~RECEIVE ReadRangeAck~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Result Flags' = (False, True, False),~~
~~'Item Count' = 1,~~
~~'Item Data' = ((a record. If the record is a time-change record, save the timestamp into TS and the time-change value into TC))~~
~~}~~
~~9. CHECK (TC ~= deltaTime)~~
~~10. CHECK (TS ~= currentTime + deltaTime)~~

1. *WRITE Enable = FALSE*
2. *WRITE Record_Count = 0*
3. *VERIFY (Log_Buffer contains 1 entry, and it is the buffer-purged event)*
4. *TRANSMIT ReadProperty-Request,*
   'Object Identifier' = (device that contains log object)
   'Property Identifier' = Local_Time
5. *RECEIVE ReadProperty-Ack,*
   'Object Identifier' = (device that contains log object)
   'Property Identifier' = Local_Time
   *'Property Value' = (**currentTime**)*
6. *WRITE Enable = TRUE*
7. *MAKE (the time change on the device by a reasonable amount (**deltaTime**); change by one hour or more)*
8. *WRITE Enable = FALSE*
9. *VERIFY Record_Count => 4*
10. *CHECK (Log_Buffer contains a log-status entry of time-change)*

11. *VERIFY (time-change value ~= **deltaTime**)*
12. *VERIFY TimeStamp on the time-change entry ~= (**currentTime** + **deltaTime**)*

7.3.2.24.15 COV-Sampling Verification Test

Reason for Change: The test here supercedes the version in 135.1-2013, with a completely different, less prescriptive approach. The Test Concept is simplified. The Configuration Requirements are enhanced.

Purpose: To verify logged samples are based on COV rather than by interval.

Test Concept: The trend log is configured to log based on COV increment. Logging is enabled. After a period of time the buffer is checked to verify the data in the buffer is based on the COV values and not on the set interval.

~~Configuration Requirements: The IUT shall be configured such that the monitored object has a COV_Increment property that is set to a value other than 0.0, the Client_COV_Increment is set to a value other than 0.0 or NULL, or the monitored property is not of datatype REAL.~~

~~Test Steps:~~

~~1. WRITE Enable = FALSE~~
~~2. WRITE Record_Count = 0~~
~~3. WRITE Interval = 0~~
~~4. WRITE Enable = TRUE~~
~~5. WAIT (10 seconds)~~
~~6. MAKE (monitored property change its value)~~
~~7. WAIT (60 seconds)~~
~~8. MAKE (monitored property change its value)~~
~~9. WAIT (90 seconds)~~
~~10. MAKE (monitored property change its value)~~
~~11. WAIT (40 seconds)~~
~~12. MAKE (monitored property change its value)~~
~~13. WAIT **Notification Fail Time**~~
~~14. WRITE Enable = FALSE~~
~~15. READ N = RecordCount~~
~~16. REPEAT X = (1 through 4) {~~
~~TRANSMIT ReadRange~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Reference Index' = N-5+X,~~
~~'Count' = 1~~
~~RECEIVE ReadRangeAck~~
~~'Object Identifier' = O1,~~
~~'Property Identifier' = Log_Buffer,~~
~~'Result Flags' = (False, False, False),~~
~~'Item Count' = 1~~
~~'Item Data' = ((one data record storing the timestamp in TS[X]))~~
~~}~~
~~17. CHECK(TS[2] - TS[1] ~= 60 seconds)~~
~~18. CHECK(TS[3] - TS[2] ~= 90 seconds)~~
~~19. CHECK(TS[4] - TS[3] ~= 40 seconds)~~

Configuration Requirements: The IUT shall be configured such that the monitored object has COV configured or the Client_COV_Increment shall be configured or it is not monitoring a REAL property. The Logging_Type shall not have a value of TRIGGERED.

Test Steps:

1. *WRITE Enable = FALSE*

2. *WRITE Record_Count = 0*
3. *WRITE Log_Interval = 0*
4. *WRITE Enable = TRUE*
5. *MAKE (monitored property change its value)*
6. *WAIT (60 seconds)*
7. *MAKE (monitored property change its value)*
8. *WAIT (90 seconds)*
9. *MAKE (monitored property change its value)*
10. *WAIT (40 seconds)*
11. *CHECK (Log_Buffer contains 3 or 4 data entries, and time between each sample is not equal)*

7.3.2.24.19 Trigger Verification Test

Reason for Change: This test has been included in 135.1-2013, but is here with a correction to the typo in Record_Count, with the steps renumbered to be consecutive, , with the distinct 'Result Flags' in the final record as noted in CR-0259, the REPEAT loop should be one fewer, and the appropriate fields present in ReadRange-Request and ReadRange-ACK are based upon Record_Count, not Total_Record_Count, since this is a request byPosition as noted by CR-0282.

Purpose: To verify logged samples are based on the triggered Logging_Type.

Test Concept: The log, O_1 is configured to log based on TRIGGERED. Logging is enabled. After a period of time the buffer is checked to verify the data in the buffer is based on triggered values.
Configuration Requirements: The IUT shall be configured such that the monitored object's Logging_Type is set to TRIGGERED.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Record_Count = 0 ~~results in a buffer purged record~~
3. WRITE Enable = TRUE ~~results in a logging enable record~~
4. WAIT (10 seconds)
5. WRITE Trigger = TRUE
6. WAIT (20 seconds)
7. WRITE Trigger = TRUE
8. WAIT (40 seconds)
9. WRITE Trigger = TRUE
10. WAIT (30 seconds)
11. WRITE Enable = FALSE ~~results in a logging disabled record~~

~~12. VERIFY RecordCount = 6~~

12. *READ N = Record_Count*
13. REPEAT X = (1 through *3*~~4~~)
    TRANSMIT ReadRange-*Request*
    'Object Identifier' = O_1,
    'Property Identifier' = Log_Buffer,
    'Reference Index' = *N-4*+X,
    'Count' = 1
    RECEIVE ~~ReadRangeAck~~*ReadRange-ACK*
    'Object Identifier' = O_1,
    'Property Identifier' = Log_Buffer,
    'Result Flags' = (~~?~~*False*, ~~?~~*False*, False),
    'Item Count' = 1,
    'Item Data' = ~~(~~(one data record storing the timestamp in TS[X])~~)~~
14. *TRANSMIT ReadRange-Request*
    'Object Identifier' = O_1,
    'Property Identifier' = Log_Buffer,

'Reference Index' = N,
'Count' = 1
*RECEIVE ReadRange-*ACK~~ck~~
'Object Identifier' = O1,
'Property Identifier' = Log_Buffer,
'Result Flags' = (False, True, False),
'Item Count' = 1,
'Item Data' = (one data record storing the timestamp in TS[4])
~~14. CHECK(TS[3] - TS[2] ~= 10 seconds)~~
~~15. CHECK(TS[4] - TS[3] ~= 20 seconds)~~
~~16. CHECK(TS[5] - TS[4] ~= 40 seconds)~~
~~17. CHECK(TS[6] - TS[5] ~= 30 seconds)~~
15. CHECK(TS[2] - TS[1] ~= 20 seconds)
16. CHECK(TS[3] - TS[2] ~= 40 seconds)
17. CHECK(TS[4] - TS[3] ~= 30 seconds)

7.3.2.24.X8 Clock-Aligned Logging

Test yet to be defined.

7.3.2.24.X9 Logging Interval_Offset

Test yet to be defined.

7.3.2.X37 Accumulator Object Tests

7.3.2.X37.1.1 Present_Value Remains In-Range Test

Reason for Change: New test for Accumulator object.
Purpose: To verify the correct wrapping operation of the Accumulator Present_Value.

Test Concept: The IUT shall be configured with a Max_Pres_Value which is attainable, within reasonable testing time, after Present_Value is preset to a value slightly less than that, then incremented. The Present_Value shall remain in range from one to Max_Pres_Value, by wrapping back to 1 when it would exceed Max_Pres_Value.

Test Steps:

1. IF (Value_Set is writable) THEN
   WRITE Value_Set = (a value slightly less than Max_Pres_Value)
   ELSE
   MAKE (Present_Value equal a value slightly less than Max_Pres_Value)
2. MAKE (the Accumulator increase its Present_Value until it rolls over Max_Pres_Value)
3. CHECK (Present_Value < Max_Pres_Value)

7.3.2.X37.1.2 Prescale in Accumulator Test

Reason for Change: New test for Accumulator object.

Purpose: To verify the correct effect of Prescale on the increment of the Present_Value in Accumulator.

Test Concept: The IUT shall be configured with a Prescale whose effect when incrementing Present_Value is testable. Three readings of the Present_Value are observed, then the math is checked to ensure that it increments at the rate expected given Prescale.

Configuration Requirements: If there is no Prescale property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. IF (Value_Set is writable) THEN
   WRITE Value_Set = (any valid value V_1)
   ELSE
   MAKE (Present_Value equal any valid value V_1)
2. MAKE (the Accumulator increase its Present_Value)
3. READ V_2 = Present_Value)
4. READ V_3 = Present_Value)
5. IF (the Accumulator is stopped) THEN
   CHECK ($V_3 = V_2$ = Prescale-multiplier * pulse-count of signals generated by the measuring instrument) / Prescale-moduloDivide + V_1)
   ELSE
   CHECK ($V_1 < V_2 < V_3$)

7.3.2.X37.1.3 Logging_Record in Accumulator Test

Reason for Change: New test for Accumulator object.
Purpose: To verify the correct values represented in Logging_Record of Accumulator.

Test Concept: Two readings of the Logging_Object acquiring the Logging_Record are performed, Pv_{prior} being the value from the first, and Present_Value matching what is observed in the second Logging_Record. Then all fields are checked to ensure these match the values expected.

Configuration Requirements: The IUT shall be configured so that Logging_Record capture is testable. If there is no Logging_Record property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. MAKE (the Logging_Object acquire the Logging_Record)
2. Pv_{prior} = present-value parameter in the Logging_Record
3. MAKE (the Logging_Object acquire another Logging_Record)
4. CHECK (Logging_Record list of values are:
   timestamp: the local date and time,
   present-value: Present_Value,
   accumulated-value: Present_Value - Pv_{prior},
   accumulated-status: NORMAL)

7.3.2.X37.1.4 Logging_Record in Accumulator RECOVERED Test

Reason for Change: New test for Accumulator object.

Purpose: To verify the correct values represented in Logging_Record of Accumulator after one or more writes to Value_Before_Change or Value_Set.

Test Concept: The effect of the Logging_Object acquiring the Logging_Record is checked to ensure that after one or more writes to Value_Before_Change or Value_Set, it matches the values expected.

Configuration Requirements: The IUT shall be configured so that Logging_Record capture is testable. If there is no Logging_Record property present in any Accumulator object, or if neither Value_Before_Change nor Value_Set is writable in an object which does have a Logging_Record property, then this test shall be skipped.

Test Steps:

1. MAKE (the Logging_Object acquire the Logging_Record)
2. Pv_{prior} = present-value parameter in the Logging_Record
3. WRITE (either Value_Before_Change or Value_Set in the object that contains Logging_Record)
4. MAKE (the Logging_Object acquire another Logging_Record)
5. CHECK (Logging_Record list of values are:
   timestamp: the local date and time,
   present-value: Present_Value,
   accumulated-value: (Present_Value - Value_Set) + (Value_Before_Change - Pv_{prior}),
   accumulated-status: RECOVERED)

7.3.2.X37.1.5 Logging_Record in Accumulator STARTING Test

Reason for Change: New test for Accumulator object.

Purpose: To verify the correct values represented in Logging_Record of Accumulator when no data has been acquired since startup by the object identified by Logging_Object.

Test Concept: The Logging_Record is observed when no data has been acquired by the object identified by Logging_Object, to ensure that it matches the values expected.

Configuration Requirements: The IUT shall be in a state when no data has been acquired since startup by the object identified by Logging_Object. If there is no Logging_Record property present in any Accumulator object, then this test shall be skipped.

Test Steps:

1. CHECK (Logging_Record list of values are:
   timestamp: unspecified,
   present-value: Present_Value,
   accumulated-value: 0,
   accumulated-status: STARTING)
2. MAKE (the Logging_Object acquire the Logging_Record)
3. CHECK (Logging_Record list of values are:
   timestamp: the local date and time,
   present-value: Present_Value,
   accumulated-value: same as present-value,
   accumulated-status: STARTING)

7.3.2.X37.1.6 Out_Of_Service Accumulator Test

Reason for Change: New test for Accumulator object.

Purpose: This test case verifies that Present_Value, Pulse_Rate, and the Reliability property are writable when Out_Of_Service is TRUE.

Test Concept: Select one instance of each appropriate object type and test it as described. Verify the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties. If the Out_Of_Service property of the object under test is not writable, and the value of the property cannot be changed by other means, then this test shall be omitted. If the Reliability property is not supported then step 5 shall be omitted.

Test Steps:

1. IF (Out_Of_Service is writable) THEN
   WRITE Out_Of_Service = TRUE
   ELSE

```
        MAKE (Out_Of_Service TRUE)
2.  VERIFY Out_Of_Service = TRUE
3.  VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4.  REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
        WRITE Present_Value = X
        VERIFY Present_Value = X
    }
5.  IF (Reliability is present and writable) THEN
        REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
                        NO_FAULT_DETECTED) DO {
            WRITE Reliability = X
            VERIFY Reliability = X
            VERIFY Status_Flags = (TRUE, TRUE, ?, TRUE)
            WRITE Reliability = NO_FAULT_DETECTED
            VERIFY Reliability = NO_FAULT_DETECTED
            VERIFY Status_Flags = (?, FALSE, ?, TRUE)
        }
6.  REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
        WRITE Pulse_Rate = X
        VERIFY Pulse_Rate = X
    }
7.  IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service = FALSE
    ELSE
        MAKE (Out_Of_Service FALSE)
8.  VERIFY Out_Of_Service = FALSE
9.  VERIFY Status_Flags = (?, ?, ?, FALSE)
```

7.3.2.X37.1.7 Value_Set Writing Test

Reason for Change: New test for Accumulator object.

Purpose: Verifying that writes to the Value_Set are reflected atomically into the object's properties.

Test Concept: Writing the Value_Set shall be reflected atomically in the Value_Set and Present_Value properties, while the old Present_Value is stored into the Value_Before_Change property, and the Value_Change_Time shall update.

Test Steps:

```
1.  READ OldV =  Present_Value
2.  WRITE Value_Set = (NewV, any valid value)
3.  VERIFY Value_Set = NewV
4.  VERIFY Present_Value = NewV
5.  VERIFY Value_Before_Change = OldV
6.  VERIFY Value_Change_Time = (approximately the current local time)
```

7.3.2.X37.1.8 Value_Before_Change Writing Test

Reason for Change: New test for Accumulator object.

Purpose: To verify the correct atomic operations of writing the Accumulator Value_Before_Change.

Test Concept: Write the Value_Before_Change and verify that it is reflected atomically in the Value_Before_Change property, while the old Present_Value is stored into the Value_Set property, and the Value_Change_Time shall update.

Test Steps:

1. READ OldV = Present_Value
2. WRITE Value_Before_Change = (NewV, any valid value)
3. VERIFY Value_Before_Change = NewV
4. VERIFY Value_Set = OldV
5. VERIFY Value_Change_Time = (approximately the current local time)

7.3.2.X38 Pulse Converter Object Tests

7.3.2.X38.1.1 Adjust_Value Write Test

Purpose: To verify the correct write operation of a Pulse Converter's several properties, when writing the Adjust_Value. Count_Before_Change reflects the prior Count before a write to the Adjust_Value property.

Test Steps:

1. READ OldV = Present_Value
2. READ OldC = Count
3. READ OldU = Update_Time
4. READ OldT = Count_Change_Time
5. READ OldA = Adjust_Value
6. READ OldS = Scale_Factor
7. READ OldB = Count_Before_Change
8. WRITE Adjust_Value = (NewA, any valid value, different from OldA so that it can be distinguished)
9. CHECK (Count is decremented by the value calculated by performing the integer division (NewA/OldS) and discarding the remainder)
10. VERIFY Present_Value is decremented by the value NewA
11. VERIFY Count_Change_Time = (approximately the current local time, and different from OldT)
12. VERIFY Count_Before_Change = OldC and != OldB

7.3.2.X38.1.2 Scale_Factor Test

Purpose: To verify the correct effect of Scale_Factor on the Present_Value in Pulse Converter.

Test Concept: The IUT shall be configured with a Scale_Factor whose influence on the behavior of Present_Value is observable. After Present_Value is read, then the value derived from Count and Scale_Factor is compared to the expected Present_Value.

Test Steps:

1. IF (Scale_Factor is writable) THEN
   WRITE Scale_Factor = (any valid value V_1)
   ELSE
   MAKE (Scale_Factor equal any valid value V_1)
2. VERIFY (Present_Value = conversion specified by Scale_Factor V_1 coefficient times the Count property)

7.3.2.X38.1.3 Out_Of_Service Pulse Converter Test

Purpose: This test case verifies that Present_Value and the Reliability property are writable when Out_Of_Service is TRUE. It also verifies the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties. If the PICS indicates that the Out_Of_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.

Test Concept: The IUT will select one instance of each appropriate object type and test it as described. If the Reliability property is not supported then step 5 shall be omitted.

Test Steps:

```
1.  IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service = TRUE
    ELSE
        MAKE (Out_Of_Service TRUE)
2.  VERIFY Out_Of_Service = TRUE
3.  VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4.   REPEAT X = (any values meeting the functional range requirements of 7.2.1) DO {
        WRITE Present_Value = X
        VERIFY Present_Value = X
    }
5.  IF (Reliability is present and writable) THEN
        REPEAT X = (any values of the Reliability enumeration appropriate to the object type except
                        NO_FAULT_DETECTED) DO {
            WRITE Reliability = X
            VERIFY Reliability = X
            VERIFY Status_Flags = (?, TRUE, ?, TRUE)
            WRITE Reliability = NO_FAULT_DETECTED
            VERIFY Reliability = NO_FAULT_DETECTED
            VERIFY Status_Flags = (?, FALSE, ?, TRUE)
        }
6.   IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service = FALSE
    ELSE
        MAKE (Out_Of_Service FALSE)
7.  VERIFY Out_Of_Service = FALSE
8.  VERIFY Status_Flags = (?, ?, ?, FALSE)
```

7.3.2.X38.1.5 Update_Time Reflects Change to the Count and is Updated Atomically Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter's several properties, for an inherent change in Count.

Test Steps:

```
1.  READ OldV =  Present_Value
2.  READ OldC =  Count
3.  READ OldU =  Update_Time
4.  READ OldT =  Count_Change_Time
5.  READ OldA =  Adjust_Value
6.  READ OldS =  Scale_Factor
7.  READ OldB =  Count_Before_Change
8.  WAIT (for a change in Count to any valid value, different from OldC so that it can be distinguished)
9.  CHECK Present_Value is recalculated, increasing in proportion to the change in Count multiplied by OldS (or such
that Present_Value minus OldA is still the same fixed difference)
10. VERIFY Update_Time = (approximately the current local time, and different from OldU)
11. VERIFY Count_Change_Time = OldT
```

7.3.2.X38.2.1 Adjust_Value Out-of-Range WriteProperty Test

Purpose: To verify the correct atomic operations of change to the Pulse Converter Count property, when an attempt is made to write Adjust_Value with a value that would cause an overflow or underflow condition in Count. The test is performed once using WriteProperty and once using WritePropertyMultiple, if IUT supports both services.

Test Steps:

1. READ OldV = Present_Value
2. READ OldC = Count
3. READ OldU = Update_Time
4. READ OldT = Count_Change_Time
5. READ OldA = Adjust_Value
6. READ OldS = Scale_Factor
7. READ OldB = Count_Before_Change
8. TRANSMIT WriteProperty-Request
   'Property Identifier' = Adjust_Value
   'Property Value' = (NewA, a valid value that would cause an overflow or underflow condition in Count)
9. RECEIVE BACnet-Error-PDU
   'Error Class' = PROPERTY
   'Error Code' = VALUE_OUT_OF_RANGE
10. VERIFY Update_Time = OldU
11. VERIFY Adjust_Value = OldA
12. VERIFY Count_Before_Change = OldB

8. APPLICATION SERVICE INITIATION TESTS

8.2 ConfirmedCOVNotification Service Initiation Tests

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

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

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present_Value is then changed by an amount greater than the COV increment and a notification shall be received. The Present_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by an Analog Input object. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

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

Test Steps:

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

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

7. IF (Out_Of_Service is writable) THEN
   WRITE X, Out_Of_Service = TRUE

   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (*ReportedPV* =~~any value appropriate for~~ *the current* Present_Value, ~~and~~ new Status_Flags)
   TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present_Value is now writable) THEN
   WRITE X, Present_Value = (any value that differs ~~from "initial Present_Value"~~ *ReportedPV* by less than "increment")
   ELSE
   MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by less than "increment")
9. WAIT **Notification Fail Time**
10. CHECK (verify that no COV notification was transmitted)
11. IF (Present_Value is now writable) THEN
   WRITE X, Present_Value = (any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by an amount greater than "increment")
   ~~RECEIVE BACnet-SimpleACK-PDU~~
   ELSE
   MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by an amount greater than "increment")
12. BEFORE **NotificationFailTime**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the new Present_Value and new Status_Flags)
13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
   WRITE X, Out_Of_Service = FALSE
   ~~RECEIVE BACnet-SimpleACK-PDU~~

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

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

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or

by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped

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

Test Steps:

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

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

5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from initial Status_Flags) | MAKE (Status_Flags = any value that differs from initial Status_Flags)

~~6.IF (WriteProperty is used in step 5) THEN~~
~~RECEIVE BACnet-SimpleACK-PDU~~

~~7~~6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (~~the initial~~*the current* Present_Value and new Status_Flags)

~~8~~7. TRANSMIT BACnet-SimpleACK-PDU

~~9~~8. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X

~~10~~9. RECEIVE BACnet-SimpleACK-PDU

~~11~~10. IF (Out_Of_Service was changed in step 5) THEN
   WRITE X, Out_Of_Service = FALSE
   ~~RECEIVE BACnet-SimpleACK-PDU~~

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

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

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

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

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

Test Steps:

REPEAT X = (one supported object of each type from the set Binary Input, Binary Output, and Binary Value) DO {

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

'List of Values' = (the new Present_Value and new Status_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out_Of_Service is writable) THEN
WRITE X, Out_Of_Service = FALSE
~~RECEIVE BACnet-SimpleACK-PDU~~

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

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

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. Removed extraneous SimpleACKs after WRITE statements. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-_Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped.
Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. *Select an object where Present_Value is not expected to change outside the tester's control or which has a writable Out_Of_Service.*

Test Steps:

REPEAT X = (one supported object of each type from the set Binary Input, Binary Output, and Binary Value) DO {

1. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
'Monitored Object Identifier' = X,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from initial Status_Flags) | MAKE (Status_Flags = any value that differs from initial Status_Flags)
~~1. IF (WriteProperty is used in step 5) THEN~~
~~RECEIVE BACnet-SimpleACK-PDU~~
~~7~~*6*. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (~~the initial~~*the current* Present_Value, and new Status_Flags)

~~8~~*7*. TRANSMIT BACnet-SimpleACK-PDU

~~9~~*8*. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = X

~~10~~*9*. RECEIVE BACnet-SimpleACK-PDU

~~11~~*10* IF (Out_Of_Service was changed in step 5) THEN
WRITE X, Out_Of_Service = FALSE
~~RECEIVE BACnet-SimpleACK-PDU~~

8.2.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~or~~ Life Safety Zone, *CharacterString Value*, *OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Present_Value Property

Reason for Change: Added more primitive value objects. Updated text for 'List of Values'. Updated 'Configuration Requirements'. Removed extraneous SimpleACKs after WRITE statements.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* objects.

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

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

Test Steps:

REPEAT X = (one supported object of each type from the set Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value*) DO {

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

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

8.2.6 Change of Value Notification from a Multi-state Input, Multi-state Output Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Status_Flags Property

Reason for Change: Added more primitive value objects. Updated Configuration Requirements. Modified text for 'List of Values' in step 7. Removed extraneous SimpleACKs after WRITE statements.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status_Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped.
Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. *Select an object where Present_Value is not expected to change outside the tester's control or which has a writable Out_Of_Service.*

Test Steps:

REPEAT X = (one supported object of each type from the set Multi-state input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value*) DO {

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

~~2. IF (WriteProperty is used in step 5) THEN~~
~~RECEIVE BACnet-SimpleACK-PDU~~

6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (~~the initial~~*the current* Present_Value, and new Status_Flags)

~~8~~7. TRANSMIT BACnet-SimpleACK-PDU
~~9~~8. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = X

~~10~~9. RECEIVE BACnet-SimpleACK-PDU
~~11~~10. IF (Out_Of_Service was changed in step 5) THEN
   WRITE X, Out_Of_Service = FALSE
   ~~RECEIVE BACnet-SimpleACK-PDU~~

8.2.7 Change of Value Notification from Loop Object Present_Value Property

Reason for Change: Added 'Configuration Requirements'. Corrected object reference in step 11. Updated wording for 'List of Values' properties. Removed extraneous SimpleACKs after WRITE statements.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of a loop object.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present_Value is then changed by an amount greater than the COV increment and a notification shall be received.

The Present_Value may be changed by placing the Loop Out_Of_Service and writing directly to the Present_Value. For implementations where this option is not possible an alternative trigger mechanism shall be provided to accomplish this task, such as changing the Setpoint or the Setpoint_Reference. All of these methods are equally acceptable.

The object identifier of the Loop object being tested is designated as O1 in the test steps below.

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

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = O1,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value, initial Status_Flags, initial Setpoint, and initial Controlled_Variable_Value)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment
6. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment,
   'Property Value' = (a value "increment" that will be used below)
7. IF (Out_Of_Service is writable) THEN
   WRITE O1, Out_Of_Service = TRUE
   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (~~the initial~~*ReportedPV = the current* Present_Value, new Status_Flags~~,~~ ~~initial~~*current* Setpoint, and ~~initial~~*current* Controlled_Variable_Value)
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Present_Value is now writable) THEN
   WRITE O1, Present_Value = (any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by less than "increment")
   ELSE
   MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~ *ReportedPV* by less than "increment")
10. WAIT **Notification Fail Time**
11. CHECK (verify that no COV notification was transmitted)
12. IF (Present_Value is now writable) THEN

WRITE O1, Present_Value = (any value that differs from ~~"initial Present_Value"~~*ReportedPV* by an amount greater than "increment")
~~RECEIVE BACnet-SimpleACK-PDU~~
ELSE
MAKE (Present_Value = any value that differs from ~~"initial Present_Value"~~*ReportedPV* by an amount greater than "increment")
13. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the new Present_Value, new Status_Flags, ~~initial~~*current* Setpoint, and ~~initial~~ *current*Controlled_Variable_Value)
14. TRANSMIT BACnet-SimpleACK-PDU
15. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = O1
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Out_Of_Service is writable) THEN
WRITE ~~L~~O1, Out_Of_Service = FALSE
~~RECEIVE BACnet-SimpleACK-PDU~~

8.2.8 Change of Value Notification from a Loop Object Status_Flags Property

Reason for Change: Updated the 'Configuration Requirements' to clarify the restrictions on the object selected. Updated descriptions in 'List of Values' property. Fixed object reference in step 11. Removed extraneous SimpleACKs after WRITE statements.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags property of a Loop object.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status_-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped.
The object identifier of the Loop object being tested is designated as O1 in the test steps below.

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

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
'Monitored Object Identifier' = O1,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the initial Present_Value, initial Status_Flags, initial Setpoint, and initial Controlled_Variable_Value)

4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE O1, Out_Of_Service = TRUE | WRITE O1, Status_Flags = (a value that differs from initial Status_Flags) | MAKE (Status_Flags = any value that differs from initial Status_Flags)

~~3. IF (WriteProperty is used in step5) THEN~~
~~RECEIVE BACnet-SimpleACK-PDU~~

6. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (~~the initial~~*the current* Present_Value, new Status_Flags, ~~initial~~*current* Setpoint, and ~~initial~~ *current* Controlled_Variable_Value)

~~8~~7. TRANSMIT BACnet-SimpleACK-PDU
~~9~~8. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (the same value used in step 1),
'Monitored Object Identifier' = O1

~~10~~9. RECEIVE BACnet-SimpleACK-PDU
~~11~~10. IF (Out_Of_Service was changed in step 5) THEN
WRITE ~~L~~*O1*, Out_Of_Service = FALSE
~~RECEIVE BACnet-SimpleACK-PDU~~

8.2.X9 ConfirmedCOVNotification Pulse Converter changing Present_Value

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present_Value property.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed by an amount less than the COV increment and it is verified that no COV notification is received. The Present_Value property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will enable the Present_Value property to be changed by the WriteProperty service. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

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

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
'Monitored Object Identifier' = O1,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the same value used in step 1),

'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (the initial Present_Value, initial Status_Flags, and Update_Time)

4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment
6. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = O1,
   'Property Identifier' = COV_Increment,
   'Property Value' = (a value "increment" that will be used below)
7. IF (Out_Of_Service is writable) THEN
   WRITE O1, Out_Of_Service = TRUE
   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (ReportedPV = the current Present_Value, new Status_Flags, and current Update_Time)
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Present_Value is now writable) THEN
   WRITE O1, Present_Value = (any value that differs from ReportedPV by less than "increment")
   ELSE
   MAKE (Present_Value = any value that differs from ReportedPV by less than "increment")
10. WAIT **Notification Fail Time**
11. CHECK (verify that no COV notification was transmitted)
12. IF (Present_Value is now writable) THEN
    WRITE O1, Present_Value = (any value that differs from ReportedPV by an amount greater than "increment")
    ELSE
    MAKE (Present_Value = any value that differs from ReportedPV by an amount greater than "increment")

13. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = O1,
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (the new Present_Value, new Status_Flags, and current Update_Time)
14. TRANSMIT BACnet-SimpleACK-PDU
15. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (the same value used in step 1),
    'Monitored Object Identifier' = O1
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Out_Of_Service was changed in step 7) THEN
    WRITE O1, Out_Of_Service = FALSE

8.2.X10 ConfirmedCOVNotification Pulse Converter changing Status_Flags

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status_Flags property.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status_-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

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

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
   'Monitored Object Identifier' = O1,
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the initial Present_Value, initial Status_Flags, and Update_Time)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE O1, Out_Of_Service = TRUE | WRITE O1, Status_Flags = (a value that differs from initial Status_Flags) |
   MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = O1,
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (the current Present_Value, new Status_Flags, and Update_Time)
7. TRANSMIT BACnet-SimpleACK-PDU
8. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (the same value used in step 1),
   'Monitored Object Identifier' = O1
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Out_Of_Service was changed in step 5) THEN
    WRITE O1, Out_Of_Service = FALSE

8.3 UnconfirmedCOVNotification Service Initiation Tests

8.3.1 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object Present_Value Property

Reason for Change: Addendum 135-2008w-1 Add more primitive value objects.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the

Present_Value property of Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* objects.

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

8.3.2 Change of Value Notification from an Analog Input, Analog Output, ~~and~~ Analog Value, *Large Analog Value, Integer Value, and Positive Integer Value* Object Status_Flags Property

Reason for Change: Addendum 135-2008w-1 Add more primitive value objects.

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

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

8.3.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Present_Value Property

Reason for Change: Addendum 135-2008w-1 Add more primitive value objects.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.5 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

8.3.6 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* Object Status_Flags Property

Reason for Change: Addendum 135-2008w-1 Add more primitive value objects.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status_Flags property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, ~~and~~ Life Safety Zone, *CharacterString Value, OctetString Value, Date Value, Date Pattern Value, DateTime Value, DateTime Pattern Value, Time Value, or Time Pattern Value* objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.6 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

8.3.X1 COVU_Recipients Notifications

Reason for Change: No existing test in the standard.

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests to each entry in its COVU_Recipients property based on COVU_Period.

Test Concept: The IUT contains a Global Group object, O1, that is configured to periodically send UnconfirmedCOVNotification using COVU_Period and COVU_Recipients. The TD checks for these notifications.

Configuration Requirements: COVU_Recipients property shall be non-empty and contain at least one device and one address based recipient. The COVU_Period shall be non-zero.

Test Steps:

```
1.  REPEAT X = (each entry in the COVU_Recipients) DO {
        BEFORE COVU_Period + Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                DESTINATION =                   X,
                'Subscriber Process Identifier' =  0,
                'Initiating Device Identifier' =   IUT,
                'Monitored Object Identifier' =    O1,
                'Time Remaining' =              0,
                'List of Values' =              (Member_Status_Flags,
                                                Elements of Present_Value)
    }
2.  READ T1 = Local_Time
3.  REPEAT X = (each entry in the COVU_Recipients) DO {
        BEFORE COVU_Period + Notification Fail Time
            RECEIVE UnconfirmedCOVNotification-Request,
                DESTINATION =                   X,
                'Subscriber Process Identifier' =  0,
                'Initiating Device Identifier' =   IUT,
                'Monitored Object Identifier' =    O1,
                'Time Remaining' =              0,
                'List of Values' =              (Member_Status_Flags,
                                                Elements of Present_Value)
    }
4.  READ T2 = Local_Time
5.  CHECK (T2 - T1 ~= COVU_Period)
```

Note to tester: The test shall pass regardless of the order in which the IUT generates the UnconfirmedCOVNotification-Requests in each step.

8.3.X11 Unsubscribed COV Service Initiation Test

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

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests.

Test Concept: Configure one or more objects in IUT to produce unsubscribed UnconfirmedCOVNotifications.

Test Steps:

```
1.  MAKE (the IUT issue an unsubscribed UnconfirmedCOVNotification)
2.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedCOVNotification-Request,
```

DESTINATION = (any valid address),
'Subscriber Process Identifier' = 0,
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = (any object present in IUT),
'Time Remaining' = 0,
'List of Values' = (any valid set of values)

8.3.X12 UnconfirmedCOVNotification Pulse Converter changing Present_Value

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Present_Value property.

Test Concept: This test is the same as 8.2.X9 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

8.3.X13 UnconfirmedCOVNotification Pulse Converter changing Status_Flags

Purpose: To verify the correct operation of COV in the Pulse Converter object. The Pulse Converter initiates periodic COV Notifications every COV_Period, even when there are no changes in the object, in addition to the COV notifications that this object type generates due to changes in the Status_Flags property.

Test Concept: This test is the same as 8.2.X10 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

8.4 ConfirmedEventNotification Service Initiation Tests

Reason for Change: This test was incorrect when used to test an Event Enrollment Object. This change is not included in any SSPC proposal.

8.4.X1 DOUBLE_OUT_OF_RANGE Tests (ConfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.8.

Purpose: To verify the correct operation of the DOUBLE_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of DOUBLE_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.4.6 except that the Event_Type is DOUBLE_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.4.X2 SIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.8.

Purpose: To verify the correct operation of the SIGNED_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of SIGNED_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.4.6 except that the Event_Type is SIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.4.X3 UNSIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.8.

Purpose: To verify the correct operation of the UNSIGNED_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of UNSIGNED_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.4.6 except that the Event_Type is UNSIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.4.X4 CHANGE_OF_CHARACTERSTRING Tests (ConfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.

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

BACnet Reference Clauses: <update these as appropriate>

Purpose: To verify the correct operation of the CHANGE_OF_CHARACTERSTRING event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_CHARACTERSTRING and to intrinsic event reporting for CharacterString Value objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to a value that is one of the values designated in List_Of_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a different value in the List_Of_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a value corresponding to a NORMAL state. After the time delay the object should enter the NORMAL state and transmit an event notification message. The transition to and from FAULT is also tested.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

The object shall be configured with a non-empty Alarm_Values property and a non-empty Fault_Values property if possible.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. IF (the object, or referenced object, if using Event Enrollment, has a non-empty Alarm_Values property) THEN
3. IF (Present_Value is writable) THEN
   WRITE Present_Value = (a value x: x = one of the Alarm_Values)
   ELSE
   MAKE (Present_Value have a value x: x = one of the Alarm_Values)
4. WAIT (Time_Delay)

5. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
        'Time Stamp' = (Toffnormal: the current local time),
        'Notification Class' = (the configured notification class),
        'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
        'Event Type' = CHANGE_OF_CHARACTERSTRING,
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = OFFNORMAL,
        'Event Values' = Present_Value, Status_Flags
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (the object being tested is NOT an Event Enrollment object) THEN
    VERIFY Status_Flags = (TRUE, FALSE,?,?)
8. VERIFY Event_State = OFFNORMAL
9. VERIFY Event_Time_Stamps = (Toffnormal, *, *)

10. IF (the object, or referenced object, if using Event Enrollment, has a Alarm_Values property with more than 1 entry) THEN
11. IF (Present_Value is writable) THEN
    WRITE Present_Value = (a value x: x = one of the Alarm_Values not used in prior steps)
    ELSE
    MAKE (Present_Value have a value x: x = one of the Alarm_Values not used in prior steps)
12. WAIT (Time_Delay)
13. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
        'Time Stamp' = (Toffnormal: the current local time),
        'Notification Class' = (the configured notification class),
        'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
        'Event Type' = CHANGE_OF_CHARACTERSTRING,
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = OFFNORMAL,
        'Event Values' = Present_Value, Status_Flags
14. TRANSMIT BACnet-SimpleACK-PDU
15. IF (the object being tested is NOT an Event Enrollment object) THEN
    VERIFY Status_Flags = (TRUE, FALSE,?,?)
16. VERIFY Event_State = OFFNORMAL
17. VERIFY Event_Time_Stamps = (Toffnormal, *, *)
18. IF (the object, or referenced object, if using Event Enrollment, has a non-empty Alarm_Values property) THEN
19. IF (Present_Value is writable) THEN
    WRITE Present_Value = (a value x: x corresponds to a NORMAL state)
    ELSE
    MAKE (Present_Value have a value x: x corresponds to a NORMAL state)
20. WAIT (Time_Delay)
21. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the Event Enrollment object being tested),
'Time Stamp' = (Tnormal: the current local time),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = CHANGE_OF_CHARACTERSTRING,
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = NORMAL,
'Event Values' = Present_Value, Status_Flags

22. TRANSMIT BACnet-SimpleACK-PDU
23. IF (the object being tested is NOT an Event Enrollment object) THEN
    VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
24. VERIFY Event_State = NORMAL
25. VERIFY Event_Time_Stamps = (Toffnormal, *, Tnormal)
26. IF (the object, or referenced object, if testing Event Enrollment, is configured with a non-empty Fault_Values property) THEN
27. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: x = one of the Fault_Values)
    ELSE
        MAKE (Present_Value have a value x: x = one of the Fault_Values)
28. WAIT (Time_Delay)
29. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the intrinsic reporting object being tested),
        'Time Stamp' = (Tfault: the current local time),
        'Notification Class' = (the configured notification class),
        'Priority' = (the value configured to correspond to a TO-FAULT transition),
        'Event Type' = CHANGE_OF_CHARACTERSTRING,
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = FAULT,
        'Event Values' = Present_Value, Status_Flags
30. TRANSMIT BACnet-SimpleACK-PDU
31. IF (the object being tested is NOT an Event Enrollment object) THEN
    VERIFY Status_Flags = (TRUE, TRUE, ?, ?)
32. VERIFY Event_State = FAULT
33. VERIFY Event_Time_Stamps = (Toffnormal,, Tfault, Tnormal)
34. VERIFY Reliability = MULTI_STATE_FAULT
35. IF (Present_Value is writable) THEN
    WRITE Present_Value = (a value x: x corresponds to a NORMAL state)
    ELSE
    MAKE (Present_Value have a value x: x corresponds to a NORMAL state)
36. WAIT (Time_Delay)
37. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the intrinsic reporting object being tested),
        'Time Stamp' = (Tfault: the current local time),

'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_CHARACTERSTRING,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	Present_Value, Status_Flags

38. TRANSMIT BACnet-SimpleACK-PDU
39. IF (the object being tested is NOT an Event Enrollment object) THEN
    VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
40. VERIFY Event_State = NORMAL
41. VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification.

8.4.X5 Proprietary Algorithm Tests (ConfirmedEventNotifications)

This test has not be developed and shall be skipped.

8.4.X6 Extended Algorithm Tests (ConfirmedEventNotifications)

This test has not be developed and shall be skipped.

8.4.X7 UNSIGNED_RANGE ConfirmedEventNotification Test

Reason for Change: New algorithm test.

Purpose: To verify the correct operation of the UNSIGNED_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of UNSIGNED_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.4.6 except that the Event_Type is UNSIGNED_RANGE instead of OUT_OF_RANGE, and there is no Deadband. If the pMonitoredValue property is not under the tester's control in IUT, then pHighLimit and/or pLowLimit are modified to generate Event notifications. The object begins the test in a NORMAL state. The pMonitoredValue is raised to a value that is above the high limit. After the time delay expires the object should enter the HIGH_LIMIT state and transmit an event notification message. The pMonitoredValue is lowered to a value that is below the high limit. After the time delay expires the object should enter the NORMAL state and issue an event notification. The same process is repeated to test the low limit.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO_OFFNORMAL and TO_NORMAL transitions, if possible. pLimitEnable property shall have a value of TRUE for both HighLimit and LowLimit events, if possible. 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, thus ensuring that notifications are emitted. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (x > pHighLimit))
   ELSE
    MAKE (pMonitoredValue have a value x: (x > pHighLimit))
3. WAIT (pTimeDelay)

4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object being tested),
   'Time Stamp' = (Toffnormal: the current local time),
   'Notification Class' = (the configured notification class),
   'Priority' = (the value configured to correspond to a TO_OFFNORMAL transition),
   'Event Type' = UNSIGNED_RANGE,
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = HIGH_LIMIT,
   'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (the object being tested is not an Event Enrollment object OR
   (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
   VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = HIGH_LIMIT
8. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
   VERIFY Event_Time_Stamps = (Toffnormal, *, *)
9. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = (a value x: (pLowLimit < x < pHighLimit))
   ELSE
   MAKE (pMonitoredValue have a value x: (pLowLimit < x < pHighLimit))
10. WAIT (pTimeDelayNormal)
11. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object being tested),
    'Time Stamp' = (Tnormal: the current local time),
    'Notification Class' = (the configured notification class),
    'Priority' = (the value configured to correspond to a TO_NORMAL transition),
    'Event Type' = UNSIGNED_RANGE,
    'Notify Type' = EVENT | ALARM,
    'AckRequired' = TRUE | FALSE,
    'From State' = HIGH_LIMIT,
    'To State' = NORMAL,
    'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
12. TRANSMIT BACnet-SimpleACK-PDU
13. IF (the object being tested is not an Event Enrollment object OR
    (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
14. VERIFY pCurrentState = NORMAL
15. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (Toffnormal, *, Tnormal)
16. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (x < pLowLimit))
    ELSE
    MAKE (pMonitoredValue have a value x: (x < pLowLimit))
17. WAIT (pTimeDelay)
18. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any valid process ID),
    'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the object being tested),
'Time Stamp' = (Tlowlimit: the current local time),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_OFFNORMAL transition),
'Event Type' = UNSIGNED_RANGE,
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = LOW_LIMIT,
'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit

19. TRANSMIT BACnet-SimpleACK-PDU
20. IF (the object being tested is not an Event Enrollment object OR
    (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
        VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
21. VERIFY pCurrentState = LOW_LIMIT
22. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (Tlowlimit, *, Tnormal)
23. IF (pMonitoredValue is writable) THEN
    WRITE pMonitoredValue = (a value x: (Low_Limit < x < High_Limit))
    ELSE
    MAKE (pMonitoredValue have a value x: (Low_Limit < x < High_Limit))
24. WAIT (pTimeDelayNormal)
25. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the object being tested),
        'Time Stamp' = (Tlowtonormal: the current local time),
        'Notification Class' = (the configured notification class),
        'Priority' = (the value configured to correspond to a TO_NORMAL transition),
        'Event Type' = UNSIGNED_RANGE,
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = LOW_LIMIT,
        'To State' = NORMAL,
        'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit
26. TRANSMIT BACnet-SimpleACK-PDU
27. IF (the object being tested is not an Event Enrollment object OR
    (Protocol_Revision is present AND Protocol_Revision ≥ 13)) THEN
        VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
28. VERIFY pCurrentState = NORMAL
29. IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (Tlowlimit, *, Tlowtonormal)

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification.

8.4.X8 CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 13.

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

BACnet Reference Clauses: <update these as appropriate>

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

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed such that a logical AND of pMonitoredValue and pSelectedFlags results in at least one bits set. After pTimeDelay expires the object shall enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed such that a logical AND of pMonitoredValue and pSelectedFlags results in no bits set. After pTimeDelayNormal expires 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 Issue_Confirmed_Notifications property 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 AND pSelectedFlags <> {FALSE, FALSE, FALSE, FALSE})
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 for the transition),
        'Event Type' = CHANGE_OF_STATUS_FLAGS,
        'Notify Type' = EVENT | ALARM,
        'Message Text' = (any valid message text),
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = OFFNORMAL,
        'Event Values' = pPresentValue, pMonitoredValue
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Status_Flags = {TRUE, FALSE,?,?}
7. VERIFY Event_State = OFFNORMAL
8. MAKE (pMonitoredValue AND pSelectedFlags = {FALSE, FALSE, FALSE, FALSE})
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 for the transition),
        'Event Type' = CHANGE_OF_STATUS_FLAGS,
        'Notify Type' = EVENT | ALARM,
        'Message Text' = (any valid message text),
        'AckRequired' = TRUE | FALSE,
        'From State' = OFFNORMAL,
        'To State' = NORMAL,
        'Event Values' = pPresentValue, pMonitoredValue
11. TRANSMIT BACnet-SimpleACK-PDU

12. VERIFY Status_Flags = {FALSE, FALSE, ?, ?}
13. VERIFY Event_State = NORMAL

8.5 UnconfirmedEventNotification Service Initiation Tests

8.5.X1 DOUBLE_OUT_OF_RANGE Tests (UnconfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.9.

Purpose: To verify the correct operation of the DOUBLE_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of DOUBLE_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.5.6 except that the Event_Type is DOUBLE_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.5.X2 SIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.9.

Purpose: To verify the correct operation of the SIGNED_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of SIGNED_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.5.6 except that the Event_Type is SIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.5.X3 UNSIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.9.

Purpose: To verify the correct operation of the UNSIGNED_OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of UNSIGNED_OUT_OF_RANGE and to object types that generate this event type intrinsically.

Test Concept: This test is the same as 8.5.6 except that the Event_Type is UNSIGNED_OUT_OF_RANGE instead of OUT_OF_RANGE.

8.5.X4 CHANGE_OF_CHARACTERSTRING Tests (UnconfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 10.
Dependencies: ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: <update these as appropriate>.

Purpose: To verify the correct operation of the CHANGE_OF_CHARACTERSTRING event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_CHARACTERSTRING and to intrinsic event reporting for CharacterString Value objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to a value that is one of the values designated in List_Of_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a different value in the List_Of_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a value corresponding to a NORMAL state. After the time delay the object should enter the NORMAL state and transmit an event notification message. The transition to and from FAULT is also tested.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

The object shall be configured with a non-empty Alarm_Values property and a non-empty Fault_Values property if possible.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.X4 except that the event notification 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.X4 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request.

8.5.X5 Proprietary Algorithm Tests (UnconfirmedEventNotifications)

This test has not be developed and shall be skipped.

8.5.X6 Extended Algorithm Tests (UnconfirmedEventNotifications)

This test has not be developed and shall be skipped.

8.5.X7 UNSIGNED_RANGE UnconfirmedEventNotification Test

Reason for Change: New algorithm test.
Purpose: To verify the correct operation of the UNSIGNED_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of UNSIGNED_RANGE and to object types that generate this event type intrinsically.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions, if possible. pLimitEnable shall have a value of TRUE for both HighLimit and LowLimit events, if possible. '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, thus ensuring that notifications are emitted. The event-generating objects 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.X7 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.X7 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.X8 CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification)

Reason for Change: New algorithm for Protocol_Revision 13.

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

BACnet Reference Clauses: <update these as appropriate>

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

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed such that a logical AND of pMonitoredValue and pSelectedFlags results in at least one bits set. After pTimeDelay expires the object shall enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed such that a logical AND of pMonitoredValue and pSelectedFlags results in no bits set. After pTimeDelayNormal expires 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 property 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.X8 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.X8 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.X9 CHANGE_OF_RELIABILITY Tests

8.5.X9.1 CHANGE_OF_RELIABILITY with No Fault Algorithm

Purpose: To verify the correct operation of an object that supports first stage reliability evaluation and does not apply a standardized fault algorithm.

Test Concept: Select an object, O1 that supports first stage reliability evaluation and does not apply a standardized fault algorithm. Ensure that no other fault conditions exist for the object. Create a fault condition. Verify the transition to fault is generated with Reliability set to R1. Remove the fault condition and verify the object transitions out of fault.

Test Configuration: O1 is configured to detect and report faults using unconfirmed event notifications. O1 is configured to have no fault conditions present and the Event_State is NORMAL.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE(O1 enter a fault condition)
4. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request
        'Process Identifier' = (any valid process identifier),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1,
        'Time Stamp' = (the current local time or sequence number),
        'Notification Class' = (the notification class configured for O1),
        'Priority' = (the value configured for the transition),

'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = (R1 any valid BACnetReliability,
(T, T, ?, ?),
(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)

5. VERIFY pCurrentReliability = R1
6. VERIFY Event_State = FAULT
7. MAKE(O1clear the fault condition)
8. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request
'Process Identifier' = (any valid process identifier),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = CHANGE_OF_RELIABILITY,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = FAULT,
'To State' = NORMAL,
'Event Values' = (NO_FAULT_DETECTED,
(F, F, ?, ?),
(A list of valid values for properties required to be reported
for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL

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

8.5.X9.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm

Purpose: To verify the correct operation of the FAULT_CHARACTERSTRING fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_CHARACTERSTRING algorithm, and no other fault conditions exist for the object. pMonitoredValue is changed to a fault string and back to a non-fault string. It is verified that O1 generates the correct transitions.

Test Configuration: O1 is configured to detect and report faults, to have no fault conditions present, and to be in the NORMAL state. FVSET is the set of character strings defined as fault values for O1. ONVSET is the set of character strings defined as offnormal values for O1. FV1 contain a substring that exists in FVSET. If the empty string is included in the FVSET, then FV1 should be the empty string. NFV1 is a string value that does not contain substrings from FVSET or ONVSET.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL

3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MULTI_STATE_FAULT,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. VERIFY Event_State = FAULT
7. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NFV1
   ELSE
   MAKE (pMonitoredValue = NFV1)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL

Notes to Tester: Note that a string is considered a substring of itself. Values required and allowed for O1 are described in standard 135 as "Properties Reported in CHANGE_OF_RELIABILITY Notifications" (Table 13-5 in 135-2016) along with supporting paragraphs.

8.5.X9.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm

Purpose: To verify the correct operation of the FAULT_EXTENDED fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_EXTENDED algorithm, and either pMonitoredValue is configured. Ensure that no other fault conditions exist for the object. In object O1, a condition is created that is detected as a fault by the FAULT_EXTENDED algorithm configured. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Test Configuration: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present, and has an Event_State of NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE (a fault condition exist)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = ((R1: any valid reliability value),
   (T, T, ?, ?),
   (a vendor specified set of values)
)
5. VERIFY pCurrentReliability = R1
6. VERIFY Event_State = FAULT
7. MAKE (remove the fault condition)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (a vendor specified set of values)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL

8.5.X9.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm

Purpose: To verify the correct operation of the FAULT_LIFE_SAFETY fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_LIFE_SAFETY algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_LIFE_SAFETY fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and has an Event_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MULTI_STATE_FAULT,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. VERIFY Event_State = FAULT
7. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),

```
        'Notification Class' =      (the notification class configured for O1),
        'Priority' =                (the value configured for the transition),
        'Event Type' =              CHANGE_OF_RELIABILITY,
        'Message Text' =            (optional, any valid message text),
        'Notify Type' =             ALARM | EVENT,
        'AckRequired' =             TRUE | FALSE,
        'From State' =              FAULT,
        'To State' =                NORMAL,
        'Event Values' =            ( NO_FAULT_DETECTED,
                                      (F, F, ?, ?),
                                      (A list of valid values for properties required to be reported
                                      for O1, and 0 or more other properties of O1)
                                    )
9.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL
```

8.5.X9.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm

Purpose: To verify the correct operation of the FAULT_STATE fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_STATE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_STATE fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and an Event_State of NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:

```
1.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
2.  VERIFY Event_State = NORMAL
3.  IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = FV1
    ELSE
        MAKE (pMonitoredValue = FV1)
4.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedEventNotification-Request
            'Process Identifier' =      (any valid process identifier),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       O1,
            'Time Stamp' =              (the current local time or sequence number),
            'Notification Class' =      (the notification class configured for O1),
            'Priority' =                (the value configured for the transition),
            'Event Type' =              CHANGE_OF_RELIABILITY,
            'Message Text' =            (optional, any valid message text),
            'Notify Type' =             ALARM | EVENT,
            'AckRequired' =             TRUE | FALSE,
            'From State' =              NORMAL,
            'To State' =                FAULT,
            'Event Values' =            ( MULTI_STATE_FAULT,
                                          (T, T, ?, ?),
                                          (A list of valid values for properties required to be reported
```

for O1, and 0 or more other properties of O1)
)

5. VERIFY pCurrentReliability = MULTI_STATE_FAULT
6. VERIFY Event_State = FAULT
7. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = NV1
   ELSE
   MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL

8.5.X9.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm

Purpose: To verify the correct operation of the FAULT_STATUS_FLAGS fault algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_STATUS_FLAGS algorithm. Ensure that no other fault conditions exist for the object. Set pMonitoredValue to FV1, a value which indicates a FAULT_STATUS_FLAGS fault condition. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredValue to NV1, a value which indicates NO_FAULT_DETECTED and verify the correct transition is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL. FV1 is a value for pMonitoredValue which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. IF (pMonitoredValue is writable) THEN
   WRITE pMonitoredValue = FV1
   ELSE
   MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),

```
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        O1,
            'Time Stamp' =                     (the current local time or sequence number),
            'Notification Class' =             (the notification class configured for O1),
            'Priority' =                       (the value configured for the transition),
            'Event Type' =                     CHANGE_OF_RELIABILITY,
            'Message Text' =                   (optional, any valid message text),
            'Notify Type' =                    ALARM | EVENT,
            'AckRequired' =                    TRUE | FALSE,
            'From State' =                     NORMAL,
            'To State' =                       FAULT,
            'Event Values' =                   ( MEMBER_FAULT,
                                                 (T, T, ?, ?),
                                                 (A list of valid values for properties required to be reported
                                                 for O1, and 0 or more other properties of O1)
                                               )
5.  VERIFY pCurrentReliability = MEMBER_FAULT
6.  VERIFY Event_State = FAULT
7.  IF (pMonitoredValue is writable) THEN
        WRITE pMonitoredValue = NV1
    ELSE
        MAKE (pMonitoredValue = NV1)
8.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedEventNotification-Request
            'Process Identifier' =             (any valid process identifier),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        O1,
            'Time Stamp' =                     (the current local time or sequence number),
            'Notification Class' =             (the notification class configured for O1),
            'Priority' =                       (the value configured for the transition),
            'Event Type' =                     CHANGE_OF_RELIABILITY,
            'Message Text' =                   (optional, any valid message text),
            'Notify Type' =                    ALARM | EVENT,
            'AckRequired' =                    TRUE | FALSE,
            'From State' =                     FAULT,
            'To State' =                       NORMAL,
            'Event Values' =                   ( NO_FAULT_DETECTED,
                                                 (F, F, ?, ?),
                                                 (A list of valid values for properties required to be reported
                                                 for O1, and 0 or more other properties of O1)
                                               )
9.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL
```

8.5.X9.7 Event Enrollment Fault Condition Precedence Tests

8.5.X9.7.1 Internal Faults Take Precedence Over Monitored Object Faults

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults in the monitored object.

Test Concept: Select an Event Enrollment object EE1 which can detect internal faults and which monitors an object O1 that can detect faults. Test that an internal unreliable operational fault takes precedence over a monitored object fault.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE(a condition exist which will cause O1 to transition into fault)
4. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
5. VERIFY Event_State = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R1)
7. VERIFY pCurrentReliability = R1
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
10. MAKE(clear the condition that caused O1 to transition into fault)
11. VERIFY pCurrentReliability = NO_FAULT_DETECTED
12. VERIFY Event_State = NORMAL

8.5.X9.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the monitored object over faults detected by fault algorithm.

Test Concept: Select an Event Enrollment object EE1 which applies a fault algorithm and which monitors an object O1 that can detect faults. Test that a monitored object fault takes precedence over a standard fault algorithm fault.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)
4. VERIFY pCurrentReliability = R1
5. VERIFY Event_State = FAULT
6. MAKE(a condition exist which will cause O1 to transition into fault)
7. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
8. MAKE(clear the condition that caused O1 to transition into fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO_FAULT_DETECTED
12. VERIFY Event_State = NORMAL

8.5.X9.7.3 Internal Faults Take Precedence Over Fault Algorithms

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to internal unreliable operational faults over faults detected by fault algorithm.

Test Concept: Select an Event Enrollment object EE1 which can detect internal faults and which applies a fault algorithm. Test that an internal unreliable operational fault takes precedence over a standard fault algorithm fault.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)

4. VERIFY pCurrentReliability = R1
5. VERIFY Event_State = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R2, different from R1)
7. VERIFY pCurrentReliability = R2
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO_FAULT_DETECTED
12. VERIFY Event_State = NORMAL

8.5.X9.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault

Purpose: To verify the proper operation of the Event Enrollment object's fault detection when the monitored object enters the fault state.

Test Concept: Select an Event Enrollment object EE1 that monitors an object M1 that can transition into FAULT. Starting with both objects in a NORMAL state, cause a condition which results in a fault in M1. Verify EE1 reports the fault. Clear the condition and verify EE1 reports the return to NORMAL.

Test Configuration: EE1 is configured to process faults in M1 and to report those using unconfirmed event notifications. EE1 and M1 are each initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:
1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE (M1 enter any fault state)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = EE1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for EE1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (MONITORED_OBJECT_FAULT,
   (T, T, ?, ?),
   M1,
   (optional, property value of M1),
   (optional, M1 Status_Flags, (T, T, ?, ?)),
   (0 or more other properties of M1)
)
5. VERIFY pCurrentReliability = MONITORED_OBJECT_FAULT
6. VERIFY Event_State = FAULT
7. MAKE (M1 clear fault state)
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),

```
        'Initiating Device Identifier' =   IUT,
        'Event Object Identifier' =        EE1,
        'Time Stamp' =                     (the current local time or sequence number),
        'Notification Class' =             (the notification class configured for EE1),
        'Priority' =                       (the value configured for the transition),
        'Event Type' =                     CHANGE_OF_RELIABILITY,
        'Message Text' =                   (optional, any valid message text),
        'Notify Type' =                    ALARM | EVENT,
        'AckRequired' =                    TRUE | FALSE,
        'From State' =                     FAULT,
        'To State' =                       NORMAL,
        'Event Values' =                   ( NO_FAULT_DETECTED,
                                             (F, F, ?, ?),
                                             M1,
                                             (optional, property value of M1),
                                             (optional, M1 Status_Flags, (?, F, ?, ?)),
                                             (0 or more other properties of M1)
                                           )
9.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL
```

8.5.X9.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault

Purpose: To verify the Event Enrollment object generates a fault event when the object enters into fault due to an internal unreliable operation.

Test Concept: Select an Event Enrollment object EE1 that can be made to enter into fault due to an internal unreliable operation. Starting EE1 in a NORMAL state, cause a condition which results in an internal fault. Verify that EE1 reports the fault. Clear the condition and verify that EE1 reports the return to NORMAL.

Test Configuration: EE1 is configured to be able to enter a fault state and to report those using unconfirmed event notifications. EE1 is initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:

```
1.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
2.  VERIFY Event_State = NORMAL
3.  MAKE (EE1 enter any internal fault state)
4.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedEventNotification-Request
        'Process Identifier' =             (any valid process identifier),
        'Initiating Device Identifier' =   IUT,
        'Event Object Identifier' =        EE1,
        'Time Stamp' =                     (the current local time or sequence number),
        'Notification Class' =             (the notification class configured for EE1),
        'Priority' =                       (the value configured for the transition),
        'Event Type' =                     CHANGE_OF_RELIABILITY,
        'Message Text' =                   (optional, any valid message text),
        'Notify Type' =                    ALARM | EVENT,
        'AckRequired' =                    TRUE | FALSE,
        'From State' =                     NORMAL,
        'To State' =                       FAULT,
        'Event Values' =                   ( (R1: any value other than
                                                 MONITORED_OBJECT_FAULT
                                                 and NO_FAULT_DETECTED),
```

```
                                        (T, T, ?, ?),
                                        (M1, any valid monitored object),
                                        (optional, property value of M1),
                                        (optional, M1 Status_Flags, (?, F, ?, ?)),
                                        (0 or more other properties of M1)
                                       )
5.  VERIFY pCurrentReliability = R1
6.  VERIFY Event_State = FAULT
7.  MAKE (EE1 clear fault state)
8.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedEventNotification-Request
            'Process Identifier' =          (any valid process identifier),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' =       EE1,
            'Time Stamp' =                  (the current local time or sequence number),
            'Notification Class' =          (the notification class configured for EE1),
            'Priority' =                    (the value configured for the transition),
            'Event Type' =                  CHANGE_OF_RELIABILITY,
            'Message Text' =                (optional, any valid message text),
            'Notify Type' =                 ALARM | EVENT,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  FAULT,
            'To State' =                    NORMAL,
            'Event Values' =                ( NO_FAULT_DETECTED,
                                              (F, F, ?, ?),
                                              M1,
                                              (optional, property value of M1),
                                              (optional, M1 Status_Flags, (?, F, ?, ?)),
                                              (0 or more other properties of M1)
                                            )
9.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL
```

8.5.X9.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL

Purpose: To verify that objects go to the NORMAL state after leaving the FAULT state, then transition to OFFNORMAL if the condition still exists.

Test Concept: Select a fault detecting object O1 which is able to detect OFFNORMAL conditions. Make O1 transition to an OFFNORMAL state and then transition to FAULT. Remove the condition causing the FAULT and verify O1 transitions from FAULT to NORMAL, then verify that the object transitions from NORMAL to the original OFFNORMAL state.

Test Configuration: O1 is configured to detect and report unconfirmed events and faults. O1 is configured to have no fault conditions present, and Event_State is OFFNORMAL.

Test Steps:

```
1.  VERIFY pCurrentReliability = NO_FAULT_DETECTED
2.  VERIFY Event_State = NORMAL
3.  MAKE(O1transition to an off normal state)
4.  BEFORE Notification Fail Time
        RECEIVE UnconfirmedEventNotification-Request
            'Process Identifier' =          (any valid process identifier),
            'Initiating Device Identifier' =  IUT,
```

'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = (ET1, any valid off normal event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (property-values appropriate for O1)

5. VERIFY Event_State = OFFNORMAL
6. MAKE(O1 enter a fault state)
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = OFFNORMAL,
   'To State' = FAULT,
   'Event Values' = ((R1 any valid BACnetReliability),
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
8. MAKE(O1 clear the fault condition)
9. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1)
)
10. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,
'Time Stamp' = (the current local time or sequence number),
'Notification Class' = (the notification class configured for O1),
'Priority' = (the value configured for the transition),
'Event Type' = ET1,
'Message Text' = (optional, any valid message text),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (property-values appropriate for O1)

11. VERIFY pCurrentReliability = NO_FAULT_DETECTED
12. VERIFY Event_State = OFFNORMAL

8.5.X9.11 CHANGE_OF_RELIABILITY with First Stage Object Fault

Purpose: To verify that fault conditions due to first stage faults are detected and reported.

Test Concept: An object in the IUT, O1, which can detect at least one first stage fault is selected. One of O1's detectable first stage faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO_FAULT_DETECTED. The first stage fault condition, R1, is made to exist and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO_FAULT_DETECTED and the appropriate event notification message is generated.

Test Configuration: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present, and Event_State is NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY Event_State = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = NORMAL,
   'To State' = FAULT,
   'Event Values' = (R1,
   (T, T, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
5. VERIFY pCurrentReliability = R1
6. VERIFY Event_State = FAULT
7. MAKE (pCurrentReliability = NO_FAULT_DETECTED)

8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (any valid process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (the current local time or sequence number),
   'Notification Class' = (the notification class configured for O1),
   'Priority' = (the value configured for the transition),
   'Event Type' = CHANGE_OF_RELIABILITY,
   'Message Text' = (optional, any valid message text),
   'Notify Type' = EVENT | ALARM,
   'AckRequired' = TRUE | FALSE,
   'From State' = FAULT,
   'To State' = NORMAL,
   'Event Values' = (NO_FAULT_DETECTED,
   (F, F, ?, ?),
   (A list of valid values for properties required to be reported
   for O1, and 0 or more other properties of O1)
)
9. VERIFY pCurrentReliability = NO_FAULT_DETECTED
10. VERIFY Event_State = NORMAL

8.11 SubscribeCOVProperty Service Initiation Tests

8.11.1 Confirmed Notifications Subscription

Reason for Change: Added test concept and variables to simplify test.

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request for confirmed notifications *to any valid object, X*.

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

Test Steps:

1. *MAKE (the IUT send a SubscribeCOVProperty-Request),*
2. RECEIVE SubscribeCOVProperty-Request
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = ~~(any valid object identifier)~~*X*
   'Issue Confirmed Notifications' = TRUE,
   'Lifetime' = ~~(any non-zero value)~~*L*,
   'Monitored Property Identifier' = ~~(any valid property identifier)~~*(the property Y to be monitored),*
   'COV Increment' = (~~any valid value~~*any REAL value* -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

8.11.2 Unconfirmed Notifications Subscription

Reason for Change: Added test concept and variables to simplify test.

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request for unconfirmed notifications *to any valid object, X*.

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

Test Steps:

1. *MAKE (the IUT send a SubscribeCOVProperty-Request),*
2. RECEIVE SubscribeCOVProperty-Request
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = ~~(any valid object identifier)~~*X*
   'Issue Confirmed Notifications' = FALSE,
   'Lifetime' = ~~(any non-zero value)~~*L,*
   'Monitored Property Identifier' = (any valid property identifier)*(the property Y to be monitored),*
   'COV Increment' = (~~any valid valudA~~*any REAL value* -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

8.11.3 Canceling a Subscription

Reason for Change: Added test concept and variables to simplify test.

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request to cancel a subscription *to any valid object, X*.

Test Steps:

1. *MAKE (the IUT send a SubscribeCOVProperty-Request),*
2. RECEIVE SubscribeCOVProperty-Request,
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = ~~(any valid object identifier)~~*X*
   'Monitored Property Identifier' = ~~(any valid property identifier)~~*(the property Y to be monitored),*
   'COV Increment' = (~~any valid value~~*any REAL value* -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

8.11.X1 Change of Value Notification Tests

8.11.X1.1 Change of Value Notification

Reason for Change: Added new test to support DS-COVP-A testing.
Purpose: To verify that the IUT can execute COVNotification requests from object types that provides a Property and Status_Flags properties in COV notifications.

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

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = X
   'Issue Confirmed Notifications' = TRUE | FALSE,
   'Lifetime' = L,
   'Monitored Property Identifier' = (the property Y to be monitored),
   'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. BEFORE Notification Fail Time
   IF (the subscription was for confirmed notifications) THEN

TRANSMIT ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the process identifier used in step 1),
'Initiating Device Identifier' = TD,
'Monitored Object Identifier' = X
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status_Flags)
RECEIVE BACnet-SimpleACK-PDU
ELSE
TRANSMIT UnconfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the process identifier used in step 1),
'Initiating Device Identifier' = TD,
'Monitored Object Identifier' = X
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status_Flags)

5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

8.11.X1.2 Change of Value Notifications with Invalid Process Identifier

Reason for Change: Added new test to support DS-COVP-A testing.

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a process identifier that does not match any current subscriptions.

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

Test Steps:
1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
'Subscriber Process Identifier' = (any valid process identifier),
'Monitored Object Identifier' = X
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L,
'Monitored Property Identifier' = (the property Y to be monitored),
'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (a process identifier different from the one used in step 1),
'Initiating Device Identifier' = TD,
'Monitored Object Identifier' = X
'Time Remaining' = (any value appropriate for the Lifetime selected),
'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status_Flags)

5. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
RECEIVE
BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = (UNKNOWN_SUBSCRIPTION) |
(BACnet-SimpleACK-PDU)
ELSE
RECEIVE

```
BACnet-Error-PDU,
    Error Class  =      SERVICES,
    Error Code  =       (any valid error code for class SERVICES) |
(BACnet-SimpleACK-PDU)
```

8.11.X1.3 Change of Value Notification Arrives after Subscription has Expired

Reason for Change: Added new test to support DS-COVP-A testing.

Purpose: To verify that an appropriate error is returned if a COV notification arrives after the subscription time period has expired.

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

Test Steps:

```
1.  MAKE (the IUT send a SubscribeCOVProperty-Request),
2.  RECEIVE SubscribeCOVProperty-Request
        'Subscriber Process Identifier' =     (any valid process identifier),
        'Monitored Object Identifier' = X
        'Issue Confirmed Notifications' =     TRUE,
        'Lifetime' =              L,
        'Monitored Property Identifier' =     (the property Y to be monitored),
        'COV Increment' =         (Any REAL value -- optional)
3.  TRANSMIT BACnet-SimpleACK-PDU
4.  BEFORE Notification Fail Time
        TRANSMIT ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =  (the process identifier used in step 1),
            'Initiating Device Identifier' =   TD,
            'Monitored Object Identifier' =    X
            'Time Remaining' =                 (any value appropriate for the Lifetime selected),
            'List of Values' =                 (values appropriate to the property Y subscribed to, and any other
                                               properties the IUT provides with it, such as Status_Flags)
        RECEIVE BACnet-SimpleACK-PDU

5.  WAIT (a value two times Lifetime)
6.  TRANSMIT ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' =      (the process identifier used in step 1),
        'Initiating Device Identifier' =       TD,
        'Monitored Object Identifier' =        X
        'Time Remaining' =                     (any value appropriate for the Lifetime selected),
        'List of Values' =                     (values appropriate to the property Y subscribed to, and any other
                                               properties the IUT provides with it, such as Status_Flags)
7.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE
        BACnet-Error-PDU,
                Error Class  =      SERVICES,
                Error Code  =       (UNKNOWN_SUBSCRIPTION) |
        (BACnet-SimpleACK-PDU)
    ELSE
        RECEIVE BACnet-Error-PDU,
                Error Class  =      SERVICES,
                Error Code  =       (any valid error code for class SERVICES) |
        (BACnet-SimpleACK-PDU)
```

8.11.X1.4 Change of Value Notifications with Invalid Monitored Object Identifier

Reason for Change: Added new test to support DS-COVP-A testing.

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

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

Test Steps:

```
1.  MAKE (the IUT send a SubscribeCOVProperty-Request),
2.  RECEIVE SubscribeCOVProperty-Request
        'Subscriber Process Identifier' =      (any valid process identifier),
        'Monitored Object Identifier' = X
        'Issue Confirmed Notifications' =    TRUE,
        'Lifetime' =                L,
        'Monitored Property Identifier' =     (the property Y to be monitored),
        'COV Increment' =           (Any REAL value -- optional)
3.  TRANSMIT BACnet-SimpleACK-PDU
4.  TRANSMIT ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' =         (the process identifier used in step 1),
        'Initiating Device Identifier' =          TD,
        'Monitored Object Identifier' =           (any object Y supporting COV notification except X),
        'Time Remaining' =                        (any value appropriate for the Lifetime selected),
        'List of Values' =                        (any value)
5.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =          SERVICES,
                Error Code  =           (UNKNOWN_SUBSCRIPTION) |
            (BACnet-SimpleACK-PDU)
    ELSE
        RECEIVE
            BACnet-Error-PDU,
                Error Class  =          SERVICES,
                Error Code  =           (any valid error code for class SERVICES) |
            (BACnet-SimpleACK-PDU)
```

8.11.X1.5 Change of Value Notifications with Invalid Monitored property

Reason for Change: Added new test to support DS-COVP-A testing.

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

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

Test Steps:

```
1.  MAKE (the IUT send a SubscribeCOVProperty-Request),
2.  RECEIVE SubscribeCOVProperty-Request
```

'Subscriber Process Identifier' = (any valid process identifier),
'Monitored Object Identifier' = X
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L,
'Monitored Property Identifier' = (the property Y to be monitored),
'COV Increment' = (Any REAL value -- optional)

3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the process identifier used in step 1),
   'Initiating Device Identifier' = TD,
   'Monitored Object Identifier' = X
   'Time Remaining' = (any value appropriate for the Lifetime selected),
   'List of Values' = (any property supporting COV notification except Y),
5. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   RECEIVE
   BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = (UNKNOWN_SUBSCRIPTION) |
   (BACnet-SimpleACK-PDU)
   ELSE
   RECEIVE
   BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = (any valid error code for class SERVICES) |
   (BACnet-SimpleACK-PDU)

8.11.X4 Requests 8 Hour Lifetimes

Reason for Change: Added new test to support DS-COVP-A testing.

Purpose: To verify that the IUT correctly generates subscription requests with lifetimes less than or equal to 8 hours. Either confirmed or unconfirmed notifications may be used, but at least one of these options shall be supported by the IUT.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request,
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = X
   'Issue Confirmed Notifications' = TRUE | FALSE,
   'Lifetime' = (any valid lifetime between 1 and 28800),
   'Monitored Property Identifier' = (the property Y to be monitored),
   'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU

8.20 ReadPropertyMultiple Service Initiation Tests

8.20.5 Cases In Which ReadProperty Shall Be Used After ReadPropertyMultiple Fails

The tests defined in this clause are used to verify that an IUT which intiates ReadPropertyMultiple is able to obtain external property values via the ReadProperty service when interoperating with a device that does not support the ReadPropertyMultiple service.

8.20.5.1 The IUT Determines the TD does not Support the ReadPropertyMultiple Service

Reason for Change: Modified test to allow multiple objects in addition to single objects.

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the IUT determines the TD does not support the ReadPropertyMultiple service.

Test Concept: The IUT is configured in a manner that would normally cause it to access one or more properties in the TD via the ReadPropertyMultiple service. Prior to sending a ReadPropertyMultiple request, however, the IUT determines that the TD does not support the ReadPropertyMultiple service. The IUT instead attempts to access the TD's property values via the ReadProperty service (it is assumed that the IUT will make this determination by reading the TD's Protocol_Services_Supported property, but this test specifically does not attempt to verify this behavior).

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it is ~~capable of~~ accessing one or more properties of a single *or multiple* objects in the TD via the ReadProperty and ReadPropertyMultiple services. ~~If the IUT cannot be configured in this way, then this test shall be omitted.~~

Test Steps:

1. MAKE (a condition in the IUT that would normally cause it to send a ReadPropertyMultiple request to the TD to access one or more property~~ies~~ values~~ of a single object~~)
2. WAIT (a time interval specified by the vendor as sufficient for the IUT to determine that the TD does not support the ReadPropertyMultiple service)
3. REPEAT X = (the properties that the IUT is to read) DO {
   RECEIVE ReadProperty-Request,
   'Object Identifier' = (object identifier referenced by X),
   'Property Identifier' = (property identifier referenced by X)
   TRANSMIT ReadProperty-Ack,
   'Object Identifier' = (object identifier referenced by X),
   'Property Identifier' = (property identifier referenced by X),
   'Property Value' = (any valid value)

8.21 ReadRange Service Initiation Tests

8.21.1 Reading Values with no Specified Range

Reason for change: 135-2008u-3.

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that does not specify any range of values to be returned.

Test Steps:

1. RECEIVE ReadRange-Request,
   'Object Identifier' = (*O*, any ~~Trend Log~~ object),
   'Property Identifier' = ~~Log_Buffer~~*(P, any list property the IUT can read)*
2. *TRANSMIT ReadRange-ACK*
   'Object Identifier' = O,

'Property Identifier' = P,
'Result Flags' = (TRUE, (bLast), (NOT bLast)),
'Item Count' = (C: any valid value)
'Item Data' = (C valid records for the requested property)

3. *CHECK(that the IUT performs the vendor specified action)*

8.21.3 Reading a Range of Values by Position

Reason for change: 135-2008u-3.

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies the range of values to be returned by position.

Test Steps:

1. RECEIVE ReadRange-Request,
   'Object Identifier' = (*O*, any ~~Trend Log~~ object),
   'Property Identifier' = ~~Log_Buffer~~*(P, any list property)*,
   'Reference Index' = (any Unsigned value),
   'Count' = (*C1*, any INTEGER value)
2. *TRANSMIT ReadRange-ACK*
   'Object Identifier' = O,
   'Property Identifier' = P,
   'Result Flags' = ((TRUE if the first was requested, FALSE otherwise), ?, ?),
   'Item Count' = (C2: any valid value <= |C|)
   'Item Data' = (C2 valid records for the requested property)
3. *CHECK(that the IUT performs the vendor specified action)*

8.21.9 Presents Log Records ~~Containing a Specific Datatype~~

Reason for Change: Modified the name of the test and improved the wording of the Purpose.

Purpose: To verify that the IUT can initiate one or more ReadRange requests that access and present a tester-specified portion of log records~~ having a specific datatype, using any valid range~~. It is a generic test used to test data presentation requirements.

Test Concept: Run test ~~in Clause~~ *135.1-2013 - 8.21.8*~~X3~~ and verify that the data presentation meets the criteria specified by the BIBB being tested.

Note to Tester: The values presented by the IUT may differ from the values transmitted on the wire due to rounding, truncation, formatting, language, conversion, etc.

Note to Tester: The IUT is not required to display records containing log-status values.

8.22 WriteProperty Service Initiation Tests

8.22.X4 Writing Array Properties as a Whole Array

Reason for Change: No test exists for this functionality. This test is not included in any SSPC proposal.

Purpose: This test verifies that the IUT is writing the entire array to the TD without the use of the array index.

Configuration Requirements: For this test, the tester shall choose a property, P_1, from an object, O_1. The TD shall be configured to not support execution of WritePropertyMultiple.

The WriteProperty request initiated by IUT shall contain array of elements in P_1, which shall fit in the APDU and segment limitations of the IUT.

Test Steps:

1. MAKE (the IUT accept a new value for P_1 including all elements of the array from the user)
2. RECEIVE WriteProperty-Request,
   'Object Identifier' = O_1,
   'Property Identifier' = P_1,
   'Property Value' = (the value provided to the IUT for P_1)
3. TRANSMIT BACnet-SimpleACK-PDU

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion etc.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of WritePropertyMultiple, the IUT may initiate a WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using WriteProperty upon receipt of the correct BACnetReject-PDU from the TD, indicating that WritePropertyMultiple is not supported.

Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

8.24 DeviceCommunicationControl Service Initiation Tests

8.24.1 Indefinite Duration, Disable, No Password

Reason For Change: This test was modified to include the responding ACK.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE,
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.2 Indefinite Duration, Disable, Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE,
   'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.3 Time Duration, Disable, Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Time Duration' = (any unsigned value > 0),
   'Enable/Disable' = DISABLE,
   'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.4 Enable, Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Enable/Disable' = ENABLE,
   'Password' = ~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.5 Enable, No Password

Reason For Change: This test was modified to include the responding ACK.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Enable/Disable' = ENABLE,
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.6 Time Duration, Disable, No Password

Reason For Change: This test was modified to include the responding ACK.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and do not convey a password. If the IUT does not support the “no password” option, this test shall not be performed.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Time Duration' = (any unsigned value > 0),
   'Enable/Disable' = DISABLE

2. *TRANSMIT BACnet-SimpleACK-PDU*

8.24.7 Time Duration, Disable-Initiation, Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and that convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,
   'Time Duration' = (any unsigned value in the range from 1 to 65535),
   'Enable/Disable' = DISABLE
   'Password' = (a password of up to 20 characters)
2. *TRANSMIT BACnet-SimpleACK-PDU*

8.27 ReinitializeDevice Service Initiation Tests

8.27.2 COLDSTART with a Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a COLDSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,
   'Reinitialized State of Device' = COLDSTART,
   'Password' =~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

8.27.4 WARMSTART with a Password

Reason For Change: This test was modified to remove the requirement of a minimum password length of 5 but include the requirement of up to 20 characters.

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a WARMSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,
   'Reinitialized State of Device' = WARMSTART,
   'Password' =~~(a password of at least 5 characters)~~ *(a password of up to 20 characters)*
2. TRANSMIT BACnet-SimpleACK-PDU

8.32 Who-Has Service Initiation Tests

8.32.3 Object Identifier Selection with a Device Instance Range

Reason for Change: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The corresponding tests incorrectly set the low limit to 1.

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Test Steps:
1. RECEIVE
   DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,
   SOURCE = IUT,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (any integer X: ~~1~~*0* <= X <= 'Device Instance Range High Limit'),
   'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' <= Y <= 4,194,303),
   'Object Identifier' = (any object identifier)

8.32.4 Object Name Selection with a Device Instance Range

Reason for Change: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The corresponding tests incorrectly set the low limit to 1.

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Test Steps:
1. RECEIVE
   DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,
   SOURCE = IUT,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (any integer X: ~~1~~*0* <= X <= 'Device Instance Range High Limit'),
   'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' <= Y <= 4,194,303),
   'Object Name' = (any CharacterString)

8.34 Who-Is Service Initiation Tests

8.34.2 Who-Is Request with a Device Instance Range

Reason for Change: The allowed device instance range is from 0 - 4194303 and is specified in sections 16.9.1.1.1 and 16.10.1.1.1. The corresponding tests incorrectly set the low limit to 1.

Purpose: To verify that the IUT can initiate Who-Is service requests with a device instance range. If the IUT cannot be caused to issue a Who-Is request of this form, then this test shall be omitted.

Test Steps:
1. RECEIVE
   DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,
   SOURCE = IUT,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (any integer X: ~~1~~*0* <= X <= 'Device Instance Range High Limit'),
   'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' <= Y <= 4,194,303)

9. APPLICATION SERVICE EXECUTION TESTS

The test cases defined in this clause shall be used to verify that a BACnet device correctly implements the service procedure for the specified application service. BACnet devices shall be tested for the proper execution of each application service for which the PICS indicates execution is supported.

For each application service included in this clause several test cases are defined that collectively test the various options and features defined for the service in the BACnet standard. A test case is a sequence of one or more messages that are exchanged between the implementation under test (IUT) and the testing device (TD) in order to determine if a particular option or feature is correctly implemented. Multiple test cases that have a similar or related purpose are collected into test groups.

Under some circumstances an IUT may be unable to demonstrate conformance to a particular test case because the test applies to a feature that requires a particular BACnet object or optional property that is not supported in the IUT. For example, a device may support the File Access services but restrict files to stream access only. Such a device would have no way to demonstrate that it could implement the record access features of the File Access services. When this type of situation occurs the IUT shall be considered to be in conformance with BACnet provided the PICS documentation clearly indicates the restriction. Failure to document the restriction shall constitute nonconformance to the BACnet standard. All features and optional parameters for BACnet application services shall be supported unless a conflict arises because of unsupported objects or unsupported optional properties.

For each application service the tests are divided into two types, positive tests and negative tests. The positive tests verify that the IUT can correctly handle cases where the service is expected to be successfully completed. The negative tests verify correct handling for various error cases that may occur. Negative tests include inappropriate service parameters but they do not include cases with encoding errors or otherwise malformed PDUs. Tests to ensure that the IUT can handle malformed PDUs are defined in 13.4.

Many test cases allow flexibility in the value to be used in a service parameter. The tester is free to choose any value within the constraints defined in the test case. The IUT shall be able to respond correctly to any valid selection the tester might make. The EPICS is considered to be a definitive reference indicating the BACnet functionality supported and the configuration of the object database. Any discrepancies between the BACnet functionality ~~or the value of properties in the object database as defined in the EPICS, and the values returned in messages defined for a test case constitutes a failure of the test. For example, if a test step involved reading a property of an object in the database the returned value must match the value provided in the EPICS.~~ *Defined in the EPICS and the functionality demonstrated by the device during testing shall constitute a failure. For example, it is considered a failure if a test step involves writing to a property and the EPICS indicates the property is writable but the device returns an error indicating 'write access denied'.*

9.1 AcknowledgeAlarm Service Execution Tests

9.1.1 Positive AcknowledgeAlarm Service Execution Tests

9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least one other device~~ *all other recipients in the Recipient_List*. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with ~~at least~~ one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have

been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (Time_Delay)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE, TRUE, TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   'Time of Acknowledgment' = (the TD's current time using a Time format)
9. RECEIVE BACnet-Simple-ACK-PDU
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),

'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3)

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the current time or sequence number),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3)

11. TRANSMIT BACnet-SimpleACK-PDU
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN

BEFORE **Notification Fail Time**

RECEIVE
DESTINATION = (at least one device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the timestamp or sequence number received in step 10),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3)

ELSE

BEFORE **Notification Fail Time**

RECEIVE
DESTINATION = (at least one device other than the TD),
SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (the process identifier configured for this event),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the timestamp or sequence number received in step 10),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Notify Type' = ACK_NOTIFICATION

13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 5. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.*

9.1.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least one other device~~ *one other device*. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_Class object, skip all steps related to receipt of the second notification.*

9.1.1.3 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least one other device~~ *one other device*. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet *if the IUT supports multiple recipients* device shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.*

9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps:
1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (Time_Delay)
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN
   RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),

'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
'Time Stamp' = (the time stamp conveyed in the notification),
'Time of Acknowledgment' = (the TD's current time using a Time format)

7. RECEIVE BACnet-Simple-ACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
        RECEIVE
            DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE = IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the current time or sequence number),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = (the 'To State' used in step 3 or 4)
    ELSE
    BEFORE **Notification Fail Time**
        RECEIVE
            DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE = IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the current time or sequence number),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION
9. IF (the notification in step 8 was not broadcast) THEN
    IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    RECEIVE
            DESTINATION = (at least one device other than the TD),
            SOURCE = IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the timestamp or sequence number from the notification in step 8),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = (the 'To State' used in step 3 or 4)
    ELSE
    RECEIVE
            DESTINATION = (at least one device other than the TD),
            SOURCE = IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),

'Time Stamp' =	(the timestamp or sequence number from the notification in step 8),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event type),
'Event Type' =	(any valid event type),
'Notify Type' =	ACK_NOTIFICATION,

10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.4. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

9.1.1.6 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.4. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 9.*

9.1.1.8 Successful Alarm Acknowledgment of Confirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Reason for Change: Added 'Notes to Tester' to clarify what to do if the TD only supports one recipient. Modified 'Configuration Requirements' to allow for only one recipient.

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, when the acknowledgement contains a mismatched or unmatched 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or an unknown 'Acknowledging Process Identifier' (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.

Configuration Requirements: The IUT shall be configured with at least one object, Object1, that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and at least one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification, and shall use different Process Identifiers.

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any Process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	Object1,
'Time Stamp' =	(the current time or sequence number),
'Notification Class' =	(the Notification Class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(any valid event type),
'Notify Type' =	ALARM or EVENT,
'AckRequired' =	TRUE,
'From State' =	(any appropriate event state),
'To State' =	(any appropriate event state),
'Event Values' =	(values appropriate to the event type)

4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (at least one device other than the TD),

SOURCE = IUT,
ConfirmedEventNotification-Request,
'Process Identifier' = (any Process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = Object1,
'Time Stamp' = (the current time or sequence number),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (any valid event type),
'Notify Type' = ALARM | EVENT,
'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)

6. TRANSMIT
   DESTINATION = IUT,
   SOURCE = (DESTINATION in step 5),
   BACnet-SimpleACK-PDU
7. VERIFY (Object1), Acked_Transitions = (one bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (Any mismatched or unknown value),
   'Event Object Identifier' = Object1,
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the timestamp conveyed in the notification),
   'Time of Acknowledgment' = (the current timestamp)
9. RECEIVE BACnet-SimpleACK-PDU
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (any Process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = Object1,
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the Notification Class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (any valid event type),
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (any appropriate event state)
11. TRANSMIT BACnet-SimpleACK-PDU
12. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
    'Process Identifier' = (any Process ID),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = Object1,
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (any valid event type),
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (any appropriate event state)
13. TRANSMIT
    DESTINATION = IUT,
    SOURCE = (DESTINATION in step 5),
    BACnet-SimpleACK-PDU
14. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The ConfirmedEventNotification-Request messages can be received in either order. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5 and 6.*

9.1.1.9 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Reason for Change: Added 'Notes to Tester' to handle cases with only one recipient. Updated 'Test Concept' to handle cases with only one recipient.

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, when the acknowledgement contains a mismatched or unmatched 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or unknown (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.

Configuration Requirements: The IUT shall be configured with at least one object, Object1, that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and at least one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification, configured to receive different Process Identifiers.

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (any Process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = Object1,
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the Notification Class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = ALARM or EVENT,
   'AckRequired' = TRUE,
   'From State' = (any appropriate event state),
   'To State' = (any appropriate event state),
   'Event Values' = (values appropriate to the event type)
4. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (any Process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE,
'From State' = (any appropriate event state),
'To State' = (any appropriate event state),
'Event Values' = (values appropriate to the event type)

5. VERIFY (Object1), Acked_Transitions = (one bit FALSE, the others TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (Any mismatched or unknown value),
   'Event Object Identifier' = Object1,
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the timestamp conveyed in the notification),
   'Time of Acknowledgment' = (the current timestamp)
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (any Process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = Object1,
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the Notification Class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (any appropriate event state)
9. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (any Process ID),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = Object1,
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (any appropriate event state)
10. VERIFY (Object1), Acked_Transitions = (TRUE,TRUE,TRUE)

Note to Tester: The UnconfirmedEventNotification-Request messages can be received in either order. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit step 4.*

9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful re-acknowledgment of an event signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status.

Test Concept: An event is triggered and, after the specified Time_Delay of the event-generating object, the IUT notifies the TD and ~~at least~~ one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol_Revision 7 and so this test shall only be performed if Protocol_Revision is present (i.e., Protocol_Revision ≥ 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends confirmed notifications. The Acked_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the event notification.

Test Steps:

1. MAKE (a change that triggers the detection of an event in the IUT)
2. WAIT (Time_Delay)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate offnormal event state),
   'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for this event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate offnormal event state),
   'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'011'~~*(FALSE, TRUE, TRUE)*
8. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   'Acknowledgment Source' = (a character string)
   'Time of Acknowledgment' = (any of the forms specified for this parameter)
9. RECEIVE BACnet-Simple-ACK-PDU
10. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the current time or sequence number),

'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event),
'Event Type' = (the event type included in step 3),
'Notify Type' = ACK_NOTIFICATION,
'To State' = (the 'To State' used in step 3)

11. TRANSMIT BACnet-SimpleACK-PDU
12. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-initiating object),
        'Time Stamp' = (the timestamp or sequence number received in step 10),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event),
        'Event Type' = (the event type included in step 3),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = (the 'To State' used in step 3)
13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*
15. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (the time stamp conveyed in the notification),
    'Acknowledgment Source' = (a character string)
    'Time of Acknowledgment' = (any of the forms specified for this parameter)
16. RECEIVE BACnet-SimpleACK-PDU
17. BEFORE **Notification Fail Time**
    RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-initiating object),
        'Time Stamp' = (the current time or sequence number),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event),
        'Event Type' = (the event type included in step 3),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = (the 'To State' used in step 3)
18. TRANSMIT BACnet-SimpleACK-PDU
19. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    ConfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-initiating object),
        'Time Stamp' = (the timestamp or sequence number received in step 17),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event),
        'Event Type' = (the event type included in step 3),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = (the 'To State' used in step 3)
20. TRANSMIT BACnet-SimpleACK-PDU
21. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*

Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 12, 13, 19, and 20.*

9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful re-acknowledgment of an event signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status.

Test Concept: An event is triggered and, after the specified Time_Delay of the event-generating object, the IUT notifies the TD and ~~at least~~ one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol_Revision 7 and so this test shall only be performed if Protocol_Revision is present (i.e., Protocol_Revision ≥ 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends unconfirmed notifications. The Acked_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the event notification.

Test Steps:

1. MAKE (a change that triggers the detection of an offnormal event in the IUT)
2. WAIT (Time_Delay)
3. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (any valid event type),
    'Notify Type' = (the notify type configured for this event),
    'AckRequired' = TRUE,
    'From State' = NORMAL,
    'To State' = (any appropriate offnormal event state),
    'Event Values' = (the values appropriate to the event type)
4. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the timestamp or sequence number received in step 3),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (any valid event type),
    'Notify Type' = (the notify type configured for this event),
    'AckRequired' = TRUE,

'From State' = NORMAL,
'To State' = (any appropriate offnormal event state),
'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'011'~~*(FALSE, TRUE, TRUE)*
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   'Acknowledgment Source' = (a character string)
   'Time of Acknowledgment' = (any of the forms specified for this parameter)
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 3),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (the 'To State' used in step 3)
9. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the event-initiating object),
   'Time Stamp' = (the timestamp or sequence number received in step 8),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event),
   'Event Type' = (the event type included in step 3),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = (the 'To State' used in step 3)
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*
11. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (the time stamp conveyed in the notification),
    'Acknowledgment Source' = (a character string)
    'Time of Acknowledgment' = (any of the forms specified for this parameter)
12. RECEIVE BACnet-SimpleACK-PDU
13. BEFORE **Notification Fail Time**
    RECEIVE UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the event-initiating object),
    'Time Stamp' = (the current time or sequence number),
    'Notification Class' = (the notification class configured for this event),
    'Priority' = (the priority configured for this event),
    'Event Type' = (the event type included in step 3),
    'Notify Type' = ACK_NOTIFICATION,
    'To State' = (the 'To State' used in step 3)

14. RECEIVE
    DESTINATION = (at least one device other than the TD),
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the event-initiating object),
        'Time Stamp' = (the timestamp or sequence number received in step 13),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event),
        'Event Type' = (the event type included in step 3),
        'Notify Type' = ACK_NOTIFICATION,
        'To State' = (the 'To State' used in step 3)
15. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = ~~B'111'~~*(TRUE, TRUE, TRUE)*

Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4, 9, and 14.*

9.1.1.X3 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is either High-Limit or Low-Limit, Revision 5 and higher only

Reason for Change: No test exists for this functionality. There is no new SSPC proposal. The differences shown are from 135.1-2011 for clarity. Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked_Transitions status when the 'To State' parameter is either High-Limit or Low-Limit and the 'Event State Acknowledged' parameter is Off-Normal.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device with an 'To State' event of either High-Limit or Low-Limit. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of 'Off-Normal' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipeients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.1.1 shall be followed except that the 'To State' parameter shall be either High-Limit or Low-Limit. When acknowledging the alarm the TD shall use an 'Event State Acknowledged' parameter of Off-Normal.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.1.1. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, skip all steps related to receipt of the second notification.*

9.1.2 Negative AcknowledgeAlarm Service Execution Tests

9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (Time_Delay)
3. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the current time or sequence number),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   ConfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number received in step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)

8. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (a time stamp older than the one conveyed in the notification),
    'Time of Acknowledgment' = (the current time using a Time format)
9. RECEIVE BACnet-Error-PDU
    Error Class = SERVICES,
    Error Code = INVALID_TIME_STAMP
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
11. TRANSMIT AcknowledgeAlarm-Request,
    'Acknowledging Process Identifier' = (the process identifier configured for this event),
    'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
    'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
    'Time Stamp' = (the time stamp conveyed in the notification),
    'Time of Acknowledgment' = (the current time using a Time format)
12. RECEIVE BACnet-Simple-ACK-PDU
13. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
        RECEIVE
        ConfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the current time or sequence number),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION,
            'To State' = (the 'To State' used in step 3 or 5)
    ELSE
    BEFORE **Notification Fail Time**
        RECEIVE
            ConfirmedEventNotification-Request,
            'Process Identifier' = (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the object detecting the alarm),
            'Time Stamp' = (the current time or sequence number),
            'Notification Class' = (the notification class configured for this event),
            'Priority' = (the priority configured for this event type),
            'Event Type' = (any valid event type),
            'Notify Type' = ACK_NOTIFICATION
14. TRANSMIT BACnet-SimpleACK-PDU
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    RECEIVE
        DESTINATION = (at least one device other than the TD),
        SOURCE = IUT,
        ConfirmedEventNotification-Request,
        'Process Identifier' = (the process identifier configured for this event),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the object detecting the alarm),
        'Time Stamp' = (the timestamp or sequence number from the notification in step 13),
        'Notification Class' = (the notification class configured for this event),
        'Priority' = (the priority configured for this event type),
        'Event Type' = (any valid event type),
        'Notify Type' = ACK_NOTIFICATION,

```
            'To State' =                    (the 'To State' used in step 3 or 5)
    ELSE
        RECEIVE
            DESTINATION =                   (at least one device other than the TD),
            SOURCE =                        IUT,
            ConfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the timestamp or sequence number from the notification in step 13),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event type),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 ACK_NOTIFICATION
16. TRANSMIT BACnet-SimpleACK-PDU
17. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)
```

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event Object Identifier' is Invalid

Reason for Change: This test was updated to account for revision 5 specifications. There is no new SSPC proposal.

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist or is not consistent with the other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an improper 'Event Object Identifier' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event Object Identifier' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall specify an object that does not support or is not configured for alarming, or which does not exist..

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the *Error Class and* Error Code in step 7 shall be *OBJECT and UNKNOWN_OBJECT if the object referenced by 'Event Object Identifier' does not exist or OBJECT and NO_ALARM_CONFIGURED if the object exists but does not support or is not configured for alarming. For devices claiming a Protocol Revision less than 5, an Error Class and Error Code of SERVICES and* INCONSISTENT_PARAMETERS *or Error Class of OBJECT and Error Code of OTHER shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

9.1.2.4 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Reason for Change: This test was updated to account for revision 5 specifications. There is no new SSPC proposal.

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the ~~other parameters~~ *Event_State* that ~~define~~ *was provided in the notification which is*~~the alarm~~ being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an invalid event state and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event state and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification*, the 'To State' in the notification shall be any offnormal transition* and the 'Event State Acknowledged' shall have a*n offnormal* value that is different from the 'To State' in the event notification *and shall not be OFFNORMAL (2).*

Notes to Tester: A passing result is the same message sequence described *as the passing result* in 9.1.2.1 except that the Error Code ~~in step 7~~ shall be *INVALID_EVENT_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of* INCONSISTENT_PARAMETERS *shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 5, 6, 15, and 16.*

9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT Time_Delay
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),

'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the object detecting the alarm),
'Time Stamp' = (the current time or sequence number),
'Notification Class' = (the notification class configured for this event),
'Priority' = (the priority configured for this event type),
'Event Type' = (any valid event type),
'Notify Type' = (the notify type configured for the event),
'AckRequired' = TRUE,
'From State' = NORMAL,
'To State' = (any appropriate non-normal event state),
'Event Values' = (the values appropriate to the event type)

4. IF (the notification in step 3 was not a broadcast) THEN
   RECEIVE
   DESTINATION = (at least one device other than the TD),
   SOURCE = IUT,
   UnconfirmedEventNotification-Request,
   'Process Identifier' = (the process identifier configured for this event),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = (the object detecting the alarm),
   'Time Stamp' = (the timestamp or sequence number from step 3),
   'Notification Class' = (the notification class configured for this event),
   'Priority' = (the priority configured for this event type),
   'Event Type' = (any valid event type),
   'Notify Type' = (the notify type configured for the event),
   'AckRequired' = TRUE,
   'From State' = NORMAL,
   'To State' = (any appropriate non-normal event state),
   'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (a time stamp older than the one conveyed in the notification),
   'Time of Acknowledgment' = (the TD's current time using a Time format)
7. RECEIVE BACnet-Error-PDU
   Error Class = SERVICES,
   Error Code = INVALID_TIME_STAMP
8. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (FALSE,TRUE,TRUE)
9. TRANSMIT AcknowledgeAlarm-Request,
   'Acknowledging Process Identifier' = (the process identifier configured for this event),
   'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
   'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
   'Time Stamp' = (the time stamp conveyed in the notification),
   'Time of Acknowledgment' = (the TD's current time using a Time format)
10. RECEIVE BACnet-Simple-ACK-PDU
11. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    BEFORE **Notification Fail Time**
    RECEIVE
    DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    SOURCE = IUT,
    UnconfirmedEventNotification-Request,
    'Process Identifier' = (the process identifier configured for this event),
    'Initiating Device Identifier' = IUT,
    'Event Object Identifier' = (the object detecting the alarm),
    'Time Stamp' = (the current time or sequence number),

```
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event type),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 ACK_NOTIFICATION,
            'To State' =                    (the 'To State' used in step 3 or 4)
    ELSE
        BEFORE Notification Fail Time
            RECEIVE
            DESTINATION =                   LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE =                        IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the current time or sequence number),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event type),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 ACK_NOTIFICATION
12. IF (the notification in step 11 was not broadcast) THEN
        IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
            RECEIVE
            DESTINATION =                   (at least one device other than the TD),
            SOURCE =                        IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the timestamp or sequence number from the notification in step 11),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event type),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 ACK_NOTIFICATION,
            'To State' =                    (the 'To State' used in step 3 or 4)
        ELSE
            RECEIVE
            DESTINATION =                   (at least one device other than the TD),
            SOURCE =                        IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the timestamp or sequence number from the notification in step 11),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event type),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 ACK_NOTIFICATION
13. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = (TRUE,TRUE,TRUE)
```

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the Referenced Object Does Not Exist

Reason For Change: Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an invalid event object identifier and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event object identifier and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall have a value that is different from the 'Event Object Identifier' in the event notification and for which no object exists in the IUT.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Class in step 7 shall be OBJECT and the Error Code in step 7 shall be UNKNOWN_OBJECT. For devices that claim a Protocol_Revision of 5 or prior, an Error Class of SERVICES with an Error Code of INCONSISTENT_PARAMETERS *or Error Class of OBJECT and Error Code of OTHER* shall also be accepted. *If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

9.1.2.7 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Reason for Change: This test was updated to account for revision 5 specifications. There is no new SSPC proposal. Made changes to allow cases where only one Recipient_List entry is supported.

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the ~~other parameters~~*Event_State* that ~~define~~ *was provided in the notification which is* ~~the alarm~~ being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and ~~at least~~ one other device. The TD acknowledges the alarm using an invalid 'Event State Acknowledged' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event State Acknowledged' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and ~~at least~~ one other BACnet device *if the IUT supports multiple recipients* shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event*, the 'To State' in the notification shall be any offnormal transition* and the 'Event State Acknowledged' shall have a*n offnormal* value that is different from the 'To State' in the event notification *and shall not be OFFNORMAL (2).*

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Code ~~in step 7~~ shall be *INVALID_EVENT_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of* INCONSISTENT_PARAMETERS *shall also be allowed. If the IUT can only be configured with one recipient in the Recipient_List property of the issuing Notification_class object, omit steps 4 and 12.*

9.1.X1 Unsupported Message Text Character Set AcknowledgeAlarm Test

Reason for Change: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.

Reference: 13.5.2

Purpose: To verify that the IUT does not fail to process an AcknowledgeAlarm request because the Acknowledgment Source parameter is of a character set that the IUT does not support.

Test Concept: Cause an event-initiating object, O1, in the IUT to transition to Event_State ES1. Acknowledge the transition and, in the AcknowledgeAlarm service, provide an 'Acknowledgment Source' parameter, AS1, which has a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Acknowledgment Source' uses a character set that the IUT does not support, and that the IUT accepts and applies that Acknowledgment request, irrespective of the 'Acknowledgment Source'.

Configuration Requirements: Configure an event-initiating object, O1 which references a Notification Class object N1. Configure O1 such that it needs an acknowledgment when it transitions out of its current state. DELAY shall represent the time delay appropriate to the transition being tested (i.e. Time_Delay for to-offnormal, 0 for to-fault, and either Time_Delay or To_Normal_Time_Delay for to-normal). AS1 shall be a character string short enough for the IUT to receive and encoded in a character set that the IUT does not support. If the IUT supports all character sets, this test shall be skipped.

Test Steps:

1. MAKE(a condition exist which will cause O1 to transition)
2. WAIT DELAY
3. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = (TS1: any valid timestamp),
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any valid text),
   'Notify Type' = ALARM | EVENT,
   'AckRequired' = TRUE,
   'From State' = (any valid event state),
   'To State' = (ES1: any valid event state),
   'Event Values' = (any values appropriate to the event type)
4. IF (ES1 = NORMAL) THEN
   VERIFY Acked_Transitions = (?,?,F)
   ELSE IF (ES1 = FAULT) THEN
   VERIFY Acked_Transitions = (?,F,?)
   ELSE
   VERIFY Acked_Transitions = (F,?,?)
5. TRANSMIT AcknowledgeAlarm-Request
   'Acknowledging Process Identifier' = (any valid value),
   'Event Object Identifier' = O1,
   'Event State Acknowledged' = ES1,
   'Time Stamp' = TS1,

'Acknowledgment Source' = AS1,
'Time of Acknowledgment' = (any valid timestamp)

6. RECEIVE BACnet-SimpleACK-PDU
7. BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedEventNotification-Request
   'Process Identifier' = (any valid process identifier),
   'Initiating Device Identifier' = IUT,
   'Event Object Identifier' = O1,
   'Time Stamp' = TS1
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (any standard event type),
   'Message Text' = (any valid text),
   'Notify Type' = ACK_NOTIFICATION,
   'To State' = ES1
8. IF (ES1 = NORMAL) THEN
   VERIFY Acked_Transitions = (?,?,T)
   ELSE IF (ES1 = FAULT) THEN
   VERIFY Acked_Transitions = (?,T,?)
   ELSE
   VERIFY Acked_Transitions = (T,?,?)

Notes to Tester: The use of UnconfirmedEventNotification is specified in this test, solely to simplify the expression of the test. The behavior being tested applies to the ConfirmedEventNotification service as well.

9.2 ConfirmedCOVNotification Service Execution Tests

9.2.1 Positive ConfirmedCOVNotification Service Execution Tests

9.2.1.X4 Change of Value Notification from Proprietary Objects

This test has not been developed and shall be skipped.

9.2.2 Negative ConfirmedCOVNotification Service Execution Tests

9.2.2.1 Change of Value Notification Arrives after Subscription has Expired

Reason for Change: Corrected tests per BTL-CR-0299 and added Configuration Requirements section.
Purpose: To verify that an appropriate error is returned if a COV notification arrives after the subscription time period has expired.

Test Steps:

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

1. RECEIVE SubscribeCOV-*Request*,
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = (any object X of a type that supports COV notification),
   'Issue Confirmed Notifications ' = TRUE,
   'Lifetime' = (~~a value no greater than one minute~~*any valid Lifetime*)
2. TRANSMIT BACnet-SimpleACK-PDU
3. *TRANSMIT ConfirmedCOVNotification-Request,*
   'Subscriber Process Identifier' = (the process identifier used in step 1),
   'Initiating Device Identifier' = TD,
   'Monitored Object Identifier' = X,

'Time Remaining' = (any amount of time greater than 0),
'List of Values' = (a list of values appropriate to object X)

4. MAKE (the IUT stop resubscribing, if it resubscribes automatically)

5~~3~~. WAIT (~~a value two times~~ *at least* Lifetime, *but sufficient to ensure the subscription has expired*)

6~~4~~. TRANSMIT ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (the process identifier used in step ~~2~~*1*),
'Initiating Device Identifier' = TD,
'Monitored Object Identifier' = X,
'Time Remaining' = (any amount of time greater than 0),
'List of Values' = (a list of values appropriate to object X)

7~~5~~. IF (Protocol_Revision is present and Protocol_Revision >= 10) THEN
RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = UNKNOWN_SUBSCRIPTION |
(BACnet-SimpleACK-PDU)
ELSE
RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = (any valid error code for class SERVICES) |
(BACnet-SimpleACK-PDU)

9.2.2.2 Change of Value Notifications with Invalid Process Identifier

Reason for Change: 135-2008h allows for a SimpleAck or a specific error code to return if a subscription does not exist.

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a process identifier that does not match any current subscriptions.

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

Test Steps:

1. RECEIVE SubscribeCOV-*Request*,
'Subscriber Process Identifier' = (any valid process identifier),
'Monitored Object Identifier' = (any object X of a type that supports COV notification),
'Issue Confirmed Notifications ' = TRUE,
'Lifetime' = (~~a value no greater than one minute~~*any valid Lifetime*)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = (a process identifier different from the one used in step ~~2~~*1*),
'Initiating Device Identifier' = TD,
'Monitored Object Identifier' = X,
'Time Remaining' = (any amount of time greater than 0),
'List of Values' = (a list of values appropriate to object X)
4. IF (Protocol_Revision is present and Protocol_Revision >= 10) THEN
RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = UNKNOWN_SUBSCRIPTION |
(BACnet-SimpleACK-PDU)
ELSE
RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = (any valid error code for class SERVICES*)* |
(BACnet-SimpleACK-PDU)

9.2.2.4 Change of Value Notifications with Invalid Monitored Object Identifier

Reason for Change: 135-2008h allows for a SimpleAck or a specific error code to return if a subscription does not exist.

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

Test Steps:

1. RECEIVE SubscribeCOV-*Request*,
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = (any object X of a type that supports COV notification),
   'Issue Confirmed Notifications ' = TRUE,
   'Lifetime' = (~~a value no greater than one minute~~*any valid Lifetime*)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the process identifier used in step ~~2~~*1*),
   'Initiating Device Identifier' = TD,
   'Monitored Object Identifier' = (any object Y *in the IUT* ~~supporting COV notification~~ except X, *and for which IUT does not already have an active subscription*),
   'Time Remaining' = (any amount of time greater than 0),
   'List of Values' = (a list of values appropriate to object Y)
4. *IF (Protocol_Revision is present and Protocol_Revision >= 10) THEN*
   RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = UNKNOWN_SUBSCRIPTION |
   (BACnet-SimpleACK-PDU)
   ELSE
   RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = (any valid error code for class SERVICES) |
   *(*BACnet-SimpleACK-PDU)

Notes to Tester: If possible, select an object Y for which IUT supports COV Subscription.

9.3 UnconfirmedCOVNotification Service Execution Tests

9.3.X9 Change of Value Notification from Proprietary Objects

This test has not been developed and shall be skipped.

9.4 ConfirmedEventNotification Service Execution Tests

9.4.5 ConfirmedEventNotification Simple Presentation

Reason for Change: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.
Purpose: This test case verifies that the IUT is capable of minimally displaying ConfirmedEventNotifications.

Configuration: For this test, the tester shall choose one event-generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,

'Process Identifier' =	(a valid process identifier specified by the IUT vendor),
'Initiating Device Identifier' =	TD,
'Event Object Identifier' =	O1,
'Time Stamp' =	(current time in any format),
'Notification Class' =	(any valid notification class),
'Priority' =	(any valid priority),
'Event Type' =	(any standard event type),
'Message Text' =	(any character string),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	(state S1, any valid state for this event type),
'To State' =	(state S2, any valid state for this event type that can follow S1),
'Event Values' =	(any values appropriate to the event type)

2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, and the event Message Text)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

Passing Result: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 32 characters. The IUT shall not truncate Message Text that is less than or equal to 32 characters in length. *A device shall not fail to process an EventNotification service request containing a 'Message Text' parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.*

9.4.6 ConfirmedEventNotification Full Presentation

Reason for Change: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.

Purpose: This test case verifies that the IUT is capable of displaying ConfirmedEventNotifications.

Configuration: For this test, the tester shall choose one event generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,

'Process Identifier' =	(a valid process identifier specified by the IUT vendor),
'Initiating Device Identifier' =	TD,
'Event Object Identifier' =	O1,
'Time Stamp' =	(current time in any format),
'Notification Class' =	(any valid notification class),
'Priority' =	(any valid priority),
'Event Type' =	(any standard event type),
'Message Text' =	(any character string),
'Notify Type' =	ALARM | EVENT,
'AckRequired' =	TRUE | FALSE,
'From State' =	(state S1, any valid state for this event type),
'To State' =	(state S2, any valid state for this event type that can follow S2),
'Event Values' =	(any values appropriate to the event type)

2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, the event Message Text, Notification Class, Priority, Notify Type, Ack Required, To State and Event Values)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

Passing Result: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 255 characters. The IUT shall not truncate Message Text that is less than or equal to 255 characters in length. *A device shall not fail to process an EventNotification service request containing a 'Message Text' parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.*

9.4.X1 Unsupported Message Text Character Set ConfirmedEventNotificationTest

Reason for Change: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.

Reference: 13.8.2

Purpose: To verify that the IUT correctly receives and processes ConfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT returns a Result(+) and performs the vendor specified actions.

Configuration Requirements: Configure the TD as though it has an event-initiating object, O1 which references a Notification Class object N1. Configure N1 to direct notifications to the IUT using a vendor specified Process Id, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
   'Process Identifier' = PID1,
   'Initiating Device Identifier' = TD,
   'Event Object Identifier' = O1,
   'Time Stamp' = (any valid timestamp),
   Notification Class' = (N1: the Notification_Class configured in O1),
   'Priority' = (any valid priority),
   'Event Type' = (the standard event type associated with O1),
   'Notify Type' = ALARM | EVENT,
   'Message Text' = T1,
   'AckRequired' = FALSE,
   'From State' = (any valid event state),
   'To State' = (any valid event state),
   'Event Values' = (any values appropriate to the event type)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

9.5 UnconfirmedEventNotification Service Execution Tests

9.5.X1 Unsupported Message Text Character Set UnconfirmedEventNotificationTest

Reason for Change: Addendum 135-2010af added language to ensure that notifications are not ignored due to unsupported character sets.

Reference: 13.9.2

Purpose: To verify that the IUT correctly receives and processes UnconfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT performs the vendor specified actions.

Configuration Requirements: Configure TD to direct notifications to the IUT using a vendor specified Process Identifier, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps: The test steps for this test case are identical to the test steps in 9.4.X1 except that the UnconfirmedEventNotification requests are used instead of ConfirmedEventNotification requests and the IUT does not acknowledge receiving the notifications.

9.7 GetEnrollmentSummary Service Execution Tests

9.7.1 Required GetEnrollmentSummary Filters

9.7.1.1 Enrollment Summary with Zero Summaries

Reason for change: BTL-CRR-0089_9.7.1.1.doc clarified that it is not important what filter parameter or parameter is used to engender the return of a summary with zero summaries.

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when there are no enrollments to report.

Configuration Requirements: The IUT shall be configured with no enrollments to report.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ~~ALL~~ *NOT_ACKED*
2. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (an empty list)

Notes to Tester: If the IUT cannot be configured with no enrollments to report, then the GetEnrollmentSummary-Request shall be transmitted with a further constrained argument so that the resulting filtered enrollment summary yields zero summaries.

9.7.2 User Selectable GetEnrollmentSummary Filters

9.7.2.3 Event Type Filter

Reason for Change: Revise test for new Event Types.

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Event Type Filter' is used.

~~Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects for each of the event types CHANGE_OF_BITSTRING, CHANGE_OF_STATE, CHANGE_OF_VALUE, COMMAND_FAILURE, FLOATING_LIMIT, and OUT_OF_RANGE. If only a subset of these event types are supported as many of them as possible shall be configured.~~

~~Test Steps:~~

~~1. TRANSMIT GetEnrollmentSummary-Request,~~
   ~~'Acknowledgment Filter' = ALL,~~
   ~~'Event Type Filter' = CHANGE_OF_BITSTRING~~
~~2. RECEIVE GetEnrollmentSummary-ACK,~~
   ~~'List of Enrollment Summaries' = (all configured event-generating objects with~~
   ~~Event_Type = CHANGE_OF_BITSTRING)~~
~~3. TRANSMIT GetEnrollmentSummary-Request,~~

~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = CHANGE_OF_STATE~~
~~4. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_STATE)~~
~~5. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = CHANGE_OF_VALUE~~
~~6. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_VALUE)~~
~~7. TRANSMIT GetEnrollmentSummary-Request,~~
~~'Acknowledgment Filter' = ALL,~~
~~'Event Type Filter' = FLOATING_LIMIT~~
~~8. RECEIVE GetEnrollmentSummary-ACK,~~
~~'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = FLOATING_LIMIT)~~

Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects for each of its supported event types. If the IUT cannot be configured in such a way all at once, then the test shall be repeated so that each of its supported event types is tested. If only a subset of these event types are supported as many of them as possible shall be configured.

Test Steps:

1. REPEAT Y = (All the configurations that will be tested) DO {
REPEAT X = (All the Event Types currently configured) DO {
TRANSMIT GetEnrollmentSummary-Request,
'Acknowledgment Filter' = ALL,
'Event Type Filter' = X
RECEIVE GetEnrollmentSummary-ACK,
'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = X)
}
}

9.8 GetEventInformation Service Execution Tests

9.8.6 Chaining Test

Reason for Change: Corrects the 'max=APDU-length-accepted' value to represent 128 bytes instead of 50 bytes.

Purpose: This test case exercises the chaining capabilities using multiple GetEventInformation messages.

Configuration Requirements: The IUT shall be configured so that there are more event states than can be conveyed in a single APDU of 128 bytes. The IUT shall be configured to contain enough events to trigger the chaining effect. If the IUT can not be configured to contain enough active events to trigger chaining, this test may be skipped.

Test Concept: In steps 1-4, the test first tests proper chaining by requesting two lists from the IUT and verifying that the second list is properly distinct from the first. In steps 5-9, to test the “fixed object processing order” as defined in BACnet 13.12.1.1.1, it requests the first list again, and then, before requesting the second list, the tester makes the last object in the first list no longer have any active event states. When the TD requests the second list using the object identifier of the now-normal device, the IUT should respond with the same second list as it did before.

Test Steps:

1. TRANSMIT GetEventInformation-Request,
   'max-APDU-length-accepted' = ~~B'0000'~~*B'0001'*,
   'segmented-response-accepted' = FALSE
2. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (an arbitrary list),
   'More Events' = TRUE
3. TRANSMIT GetEventInformation-Request,
   'Last Received Object Identifier' = the last object identifier of the list received in step 2)
4. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (a list of object identifiers not including any received in step 2)
5. TRANSMIT GetEventInformation-Request,
   'max-APDU-length-accepted' = ~~B'0000'~~*B'0001'*,
   'segmented-response-accepted' = FALSE
6. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (an arbitrary list),
   'More Events' = TRUE
7. MAKE (the object identified by the last object identifier in the list received in step 6 have no active event states)
8. TRANSMIT GetEventInformation-Request,
   'Last Received Object Identifier' = (the last object identifier of the list received in step 6)
9. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (the same list received in step 4)

9.10 SubscribeCOV Service Execution Tests

9.10.1 Positive SubscribeCOV Service Execution Tests

The purpose of this test group is to verify the correct execution of the SubscribeCOV service request under circumstances where the service is expected to be successfully completed.

9.10.1.7 Finite Lifetime Subscriptions

Reason for change: Updates description of 'Time Remaining' and adds validation that this value counts down as expected.

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with a temporary lifetime. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

1. TRANSMIT SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any valid process identifier),
   'Monitored Object Identifier' = (any object supporting COV notifications),
   'Issue Confirmed Notifications' = TRUE | FALSE,
   'Lifetime' = (a value between 60 seconds and 300 seconds)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN
   BEFORE **Notification Fail Time**
   RECEIVE ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the same identifier used in the subscription),
   'Initiating Device Identifier' = IUT,
   'Monitored Object Identifier' = (the same object used in the subscription),
   'Time Remaining' = (A value approximately equal to, but not greater than, the requested subscription lifetime)
   'List of Values' = (values appropriate to the object type of the monitored object)
   TRANSMIT BACnet-SimpleACK-PDU
   ELSE
   BEFORE **Notification Fail Time**
   RECEIVE UnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = (the same object used in the subscription),
'Time Remaining' = (A value approximately equal to, but not greater than, the requested subscription lifetime),
'List of Values' = (values appropriate to the object type of the monitored object)

4. MAKE (a change to the monitored object that ~~should~~ causes a COV notification)
5. IF (the subscription was for confirmed notifications) THEN
    BEFORE **Notification Fail Time**
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same identifier used in the subscription),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            'Time Remaining' = (*TR:* a value greater than 0 and less than *or equal to* the requested subscription lifetime),
            'List of Values' = (values appropriate to the object type of the monitored object)
    TRANSMIT BACnet-SimpleACK-PDU
ELSE
    BEFORE **Notification Fail Time**
        RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same identifier used in the subscription),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            'Time Remaining' = (*TR:* a value greater than 0 and less than *or equal to* the requested subscription lifetime),
            'List of Values' = (values appropriate to the object type of the monitored object including the changed value of that triggered the notification)
6. *WAIT (a time that should change the 'Time Remaining' and which is less than the lifetime of the subscription)*
7. *MAKE (a change to the monitored object that causes a COV notification)*
8. *IF (the subscription was for confirmed notifications) THEN*
    *BEFORE **Notification Fail Time***
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same identifier used in the subscription),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            'Time Remaining' = (a value greater than 0 and less than the TR),
            'List of Values' = (values appropriate to the object type of the monitored object)
    TRANSMIT BACnet-SimpleACK-PDU
ELSE
    *BEFORE **Notification Fail Time***
        RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same identifier used in the subscription),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            'Time Remaining' = (a value greater than 0 and less than TR),
            'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

~~7~~*9*. WAIT (the lifetime of the subscription)
~~8~~*10*.MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
~~9~~*11*.CHECK (verify that the IUT did not transmit a COV notification message)

9.10.1.X1 Ensuring 5 Concurrent COV Subscribers

Reason For Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: This test case verifies that the IUT can support 5 concurrent subscriptions.

Test Concept: Have the TD subscribe with 5 different process identifiers, V_1 through V_5, and then check to ensure that 5 notifications are sent when the monitored object changes.

Test Steps

1. REPEAT (X=V_1 to V_5) DO {
    TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' = X,
        'Monitored Object Identifier' = (any object supporting COV notifications),
        'Issue Confirmed Notifications' = FALSE,
        'Lifetime' = (any valid value that will allow the subscription to outlast the test)
    RECEIVE BACnet-SimpleACK-PDU
  WAIT **Notification Fail Time**
  IF (if confirmed notifications were requested) THEN
    RECEIVE ConfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = X,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = (the same object used in the subscription),
        'Time Remaining' = (any valid value),
        'List of Values' = (the initial Present_Value and initial Status_Flags)
    TRANSMIT BACnet-SimpleACK-PDU
  ELSE
    RECEIVE UnconfirmedCOVNotification-Request,
        'Subscriber Process Identifier' = X,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = (the same object used in the subscription),
        'Time Remaining' = (any valid value),
        'List of Values' = (the initial Present_Value and initial Status_Flags)
  }
2. MAKE (Present_Value = any value that differs from "initial Present_Value" such that a COV notification would be generated)
3. REPEAT (X=V_1 to V_5) DO {
    IF (if confirmed notifications were requested) THEN
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = X,
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            ‘Time Remaining' = (any valid value),
            'List of Values' = (the new Present_Value and Status_Flags)
        TRANSMIT BACnet-SimpleACK-PDU
    ELSE
        RECEIVE UnconfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = X,
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the same object used in the subscription),
            'Time Remaining' = (any valid value),
            'List of Values' = (the new Present_Value and Status_Flags)
  }

Passing Result: The notification in step 3 can be received in any order by the TD.

9.10.2 Negative SubscribeCOV Service Execution Tests

9.10.2.1 The Monitored Object Does Not Support COV Notification

Reason For Change: Added configuration requirements.

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the monitored object does not support COV notifications.

Configuration Requirements: This test shall only be executed if IUT contains objects which will not accept a COV subscription. If every object in IUT will accept a COV subscription, then this test shall be skipped.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =      (any valid process identifier),
        'Monitored Object Identifier' = (any object that does not support COV notifications),
        'Issue Confirmed Notifications' =    TRUE,
        'Lifetime' =              60
2.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        RECEIVE BACnet-Error PDU,
            'Error Class' =          OBJECT,
            'Error Code' =           OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
    ELSE
        RECEIVE
            (BACnet-Error PDU,
                'Error Class' =          OBJECT,
                'Error Code' =           OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
            (BACnet-Error PDU,
                Error Class  =           SERVICES,
                Error Code  =            SERVICE_REQUEST_DENIED | OTHER) |
        (BACnet-Error PDU,
                'Error Class' =          PROPERTY,
                'Error Code' =           NOT_COV_PROPERTY)
```

9.10.2.X1 The Monitored Object Does Not Exist

Reason for Change: 135-2008h allows for a SimpleAck or a specific error code to return if a subscription does not exist.

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the monitored object does not exist.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =      (any valid process identifier),
        'Monitored Object Identifier' = (any object of a type that supports COV and an instance which does not exist
                                         in the IUT),
        'Issue Confirmed Notifications' =    TRUE,
        'Lifetime' =                  60
2.  IF (Protocol_Revision is present and Protocol_Revision >= 10) THEN
         RECEIVE BACnet-Error PDU,
                Error Class  =           OBJECT,
                Error Code  =            UNKNOWN_OBJECT
    ELSE
         RECEIVE BACnet-Error PDU,
                Error Class  =           SERVICES,
                Error Code  =            SERVICE_REQUEST_DENIED | OTHER
```

```
| (BACnet-Error PDU,
        Error Class  =          OBJECT,
        Error Code  =           UNKNOWN_OBJECT)
```

Note to tester: If the IUT is able to support objects other than those that currently exist, and none of those objects that currently do not exist would support COV notification if they did, then the IUT may return an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED instead of UNKNOWN_OBJECT.

9.10.2.X2 There Is No Space For A Subscription

Reason for Change: 135-2008h.5.

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error. This test only applies to IUTs that claim a Protocol_Revision of 10 or higher.

Test Conditionality: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test may be skipped.

Test Steps:

```
REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until the IUT returns
                an Error-PDU) {
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     PID,
        'Monitored Object Identifier' =       (any object of that supports COV),
        'Issue Confirmed Notifications' =     TRUE,
        'Lifetime' =                          6000
2.  RECEIVE BACnet-SimpleACK-PDU |
        (BACnet-Error-PDU,
                Error Class  =     RESOURCES,
                Error Code  =      NO_SPACE_TO_ADD_LIST_ELEMENT)
3.  READ ACS = (Active_COV_Subscriptions)
4.  IF  (a BACnet-Simple-Ack was received in step 2) THEN
        CHECK (that the subscription is in ACS)
    ELSE
        CHECK (that the subscription is not in ACS)
}
```

9.10.2.X3 The Lifetime Parameter is Out of Range

Reason for Change: 135-2008h.5. Modified to relax allowed rejection response.

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     (any valid process identifier),
        'Monitored Object Identifier' = (any object in the IUT that supports COV),
        'Issue Confirmed Notifications' =     TRUE,
        'Lifetime' =              (a value larger than that supported by the IUT)
2.  IF (Protocol_Revision is present and Protocol_Revision >= 10) THEN
```

```
        RECEIVE BACnet-Error-PDU,
            Error Class  =      SERVICES,
            Error Code  =       VALUE_OUT_OF_RANGE
    ELSE
        RECEIVE BACnet-Error-PDU,
            Error Class  =      SERVICES,
            Error Code  =       VALUE_OUT_OF_RANGE | SERVICE_REQUEST_DENIED | OTHER
            | (RECEIVE BACnet-Reject-PDU,
                Reject Reason =   PARAMETER_OUT_OF_RANGE)
```

9.10.3 Positive Unsubscribed COVNotification Execution Tests

9.10.3.X1 Unsubscribed COVNotification Execution Test

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

Purpose: To verify that the IUT executes UnconfirmedCOVNotification service requests, with 'Process Identifier' equal to 0.

Test Concept: Using any received and supported unsubscribed UnconfirmedCOVNotification, observe the effect of its execution.

Test Steps:

```
1.  TRANSMIT UnconfirmedCOVNotification-Request,
        'Subscriber Process Identifier' =       0,
        'Initiating Device Identifier' =   TD,
        'Monitored Object Identifier' = (any object present in TD),
        'Time Remaining' =          0,
        'List of Values' =          (any valid set of values)
2.  CHECK (for any vendor-defined observable actions)
```

9.14 AddListElement Service Execution Tests

9.14.2 Negative AddListElement Service Execution Tests

9.14.2.2 Adding a List Element With an Invalid Datatype

Reason for change: Added the additional error conditions that are now accepted. Added 'Note to Tester' that was missing in 135.1-2013.

Purpose: To verify the ability of the IUT to correctly respond to an AddListElement service request to add an element with an invalid datatype to a list.

Test Steps:

1. TRANSMIT AddListElement-Request,
   'Object Identifier' = L,
   'Property Identifier' = ListProp,
   'List of Elements' = (a single element with a datatype inappropriate for this property)
2. RECEIVE AddListElement-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE,
   'First Failed Element' = 1 |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_PARAMETER_DATATYPE) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_TAG)

Notes to Tester: value selected for step 1 is 'inappropriate', not a value which is 'allowed' but not supported by this instance of the property. I.e. it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE_NOT_SUPPPORTED is only correct when the datatype requested is supported, for example in a CHOICE, by this property in this object type, but not supported by this instance of the property.

9.14.2.3 An AddListElement Failure Part Way Through a List

Reason For Change: Updated test to include additional error codes. Added 'Notes to Tester' which was missing in 135.1-2013.

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add multiple elements to a list where one of the elements cannot be added. Upon failure, the AddListElement service should leave the list unchanged.

Test Steps:

1. READ InitialList = (L), ListProp2
2. TRANSMIT AddListElement-Request,
   'Object Identifier' = L,
   'Property Identifier' = ListProp
   'List of Elements' = (two or more elements to be added to the list with the second element having an inappropriate datatype)
3. IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN
   RECEIVE AddListElement-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE,
   'First Failed Element' = 2
   | (RECEIVE BACnet-Reject-PDU,
   Reject Reason = INVALID_TAG | INVALID_PARAMETER_DATA_TYPE)
   ELSE

RECEIVE AddListElement-Error,
Error Class = SERVICES,
Error Code = INVALID_PARAMETER_DATATYPE
'First Failed Element' = 2
| *(AddListElement-Error,*
Error Class = PROPERTY,
Error Code = INVALID_DATATYPE)
'First Failed Element' = 2
| *(BACnet-Reject-PDU,*
Reject Reason = INVALID_TAG | INVALID_PARAMETER_DATA_TYPE)

4. VERIFY (L), ListProp = InitialList

Notes to Tester: value selected for step 3 is 'inappropriate', not a value which is 'allowed' but not supported by this instance of the property. I.e. it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE_NOT_SUPPPORTED is only correct when the datatype requested is supported, for example in a CHOICE, by this property in this object type, but not supported by this instance of the property.

9.15 RemoveListElement Service Execution Tests

9.15.2 Negative RemoveListElement Service Execution Tests

9.15.2.2 A RemoveListElement Failure Part Way Through a List

Reason For Change: The test specified an incorrect error code. .

Purpose: To verify the ability of the IUT to respond to a RemoveListElement service request to remove multiple elements from a list where one of the elements cannot be removed. Upon failure, the RemoveListElement service should leave the list unchanged.

Test Steps:

1. READ InitialList = (L), ListProp
2. TRANSMIT RemoveListElement-Request,
   'Object Identifier' = L,
   'Property Identifier' = ListProp
   'List of Elements' = (one element from InitialList, followed by an element of the correct datatype that is not in InitialList, followed by one or more elements from InitialList)
3. If (Protocol_Revision is present and Protocol_Revision >= 7) THEN
   RECEIVE RemoveListElement-Error,
   Error Class = ~~PROPERTY~~ *SERVICES,*
   Error Code = ~~INVALID_DATA+TYPE~~*LIST_ELEMENT_NOT_FOUND*
   'First Failed Element' = 2
   ELSE
   RECEIVE RemoveListElement-Error
   ~~Error Class = SERVICES | PROPERTY,~~
   ~~Error Code = OTHER~~
   ~~'First Failed Element' = 2~~
4. VERIFY (L), ListProp = InitialList

9.16 CreateObject Service Execution Tests

9.16.1 Positive CreateObject Service Execution Tests

9.16.1.2 Creating Objects by Specifying the Object Identifier with No Initial Values

Reason For Change: Added clarification that the IUT can place a restriction on the instance used. This correction is not in any SSPC proposal.

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any unique object identifier of a type that is creatable *and an instance number that is creatable*)
2. RECEIVE CreateObject-ACK,
   'Object Identifier' = (the object identifier specified in step 1)
3. VERIFY (the object identifier of the newly created object),
   (any required property of the specified object) = (any value of the correct datatype for the specified property)
4. VERIFY (the IUT's Device object), Object_List = (any object list containing the newly created object)

9.16.1.4 Creating Objects by Specifying the Object Identifier and Providing Initial Values

Reason For Change: Added clarification that the IUT can place restrictions on the instance and initial values allowed for creation. This change is not in any SSPC proposal.

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier and a list of initial property values is provided.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any unique object identifier of a type that is creatable *and an instance number that is creatable*)
   'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept)

2. RECEIVE CreateObject-ACK,
   'Object Identifier' = (the object identifier specified in step 1)
3. REPEAT X = (properties initialized in the CreateObject-Request) DO {
   VERIFY (the object identifier for the newly created object),
   X = (the value specified in the 'List Of Initial Values' parameter of the CreateObject-Request)
   }
4. VERIFY (the IUT's Device object), Object_List = (any object list containing the newly created object)

9.16.2 Negative CreateObject Service Execution Tests

The purpose of this test group is to verify correct execution of the CreateObject service requests under circumstances where the service is expected to fail.

9.16.2.1 Attempting to Create an Object That Does Not Have a Unique Object Identifier

Reason For Change: Corrected the parameter used in the service request. This is not in any SSPC proposal.

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier that already exists in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any object identifier representing an object that already exists having an object type for which dynamic creation is supported)
2. RECEIVE CreateObject-Error,
   Error Class = OBJECT,
   Error Code = OBJECT_IDENTIFIER_ALREADY_EXISTS
   'First Failed Element Number' = 0

9.16.2.4 Attempting to Create an Object with an Object Type Specifier and an Error in the Initial Values

Reason for Change: Added Test Concept and Configuration Requirements.

Purpose: To verify the correct execution of the CreateObject service request when an object type is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule_Default which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.

Test Steps:

1. READ X1 = Object_List
2. TRANSMIT CreateObject-Request,
   'Object ~~Type~~*Specifier*' = (any creatable object type),
   'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
3. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE CreateObject-Error~~-PDU~~,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
   ELSE
   RECEIVE CreateObject-Error,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE | OTHER
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
4. CHECK(Verify that the new object was not created)
5. TRANSMIT CreateObject-Request,
   'Object ~~Type~~*Specifier*' = (object type of *step 2*),
   'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being an inappropriate datatype)
6. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE

CreateObject-Error,
Error Class = PROPERTY,
Error Code = INVALID_DATATYPE
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |
(BACnet-Reject-PDU
Reject Reason = INVALID_PARAMETER_DATATYPE | INVALID_TAG)

ELSE
RECEIVE CreateObject-Error,
Error Class = PROPERTY,
Error Code = VALUE_OUT_OF_RANGE | INVALID_DATATYPE | OTHER
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |
(BACnet-Reject-PDU
Reject Reason = INVALID_PARAMETER_DATATYPE | INVALID_TAG)

7. READ X2 = Object_List
8. CHECK (X1=X2)

9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values

Reason for Change: Added Test Concept and Configuration Requirements to clarify usage.

Purpose: To verify the correct execution of the CreateObject service request when an object identifier is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule_Default which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object ~~Identifier~~*Specifier*' = (any unique object identifier of a type that is creatable and an instance number that is creatable),
   'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
2. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE CreateObject-Error~~-PDU~~,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)

   ELSE
   RECEIVE CreateObject-Error,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE | *OTHER*
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
3. CHECK(Verify that the new object was not created)
4. TRANSMIT CreateObject-Request,
   'Object Specifier' = (object identifier from step 1),
   'List Of Initial Values' = (a list of ~~two~~*one* or more properties and their initial values, that the IUT will accept initial values for, with one of the values being an

inappropriate datatype)

5. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE
   CreateObject-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_PARAMETER_DATATYPE) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_TAG)
   ELSE
   RECEIVE
   CreateObject-Error,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE | INVALID_DATATYPE | OTHER
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offendingvalue) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_PARAMETER_DATATYPE | INVALID_TAG)
6. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the 'Object Identifier' used in step 1),
   'Property Identifier' = Object_Name
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE BACnet-Error-PDU,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT
   ELSE
   RECEIVE BACnet-Error-PDU
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT | NO_OBJECTS_OF_SPECIFIED_TYPE | OTHER

9.16.2.6 Attempting to Create an Object with an instance of 4194303

Reason For Change: Corrected parameter for service request. This change is not in any SSPC proposal.

Purpose: This test case verifies the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier with an instance of 4194303. This test shall be performed only if the Protocol_Revision property is present in the Device object and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any object identifier representing a creatable object-type with an instance of 4194303)
2. RECEIVE BACnet-Reject-PDU,
   'Reject Reason' = PARAMETER_OUT_OF_RANGE

9.16.2.X1 Attempting to Create a non-Supported Object Type (by Object Type)

Reason for Change: Addendum 135-2008u-2

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object type that is not supported in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any unsupported object type)
2. IF (Protocol_Revision >= 10)
   RECEIVE CreateObject-Error,
   Error Class = OBJECT,
   Error Code = UNSUPPORTED_OBJECT_TYPE
   'First Failed Element Number' = 0.
   ELSE
   RECEIVE CreateObject-Error,
   Error Class = (any valid error class),
   Error Code = (any valid error code)
   ‘First Failed Element Number’ = 0
3. VERIFY (the IUT's Device object),
   Object_List = (any object list that does not contain the object specified in step 1)

9.16.2.X2 Attempting to Create a non-Supported Object Type (by Object Identifier)

Reason for Change: Addendum 135-2008u-2

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier for an object type that is not supported in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,
   'Object Specifier' = (any object identifier having an unsupported object type)
2. IF (Protocol_Revision >= 10)
   RECEIVE CreateObject-Error,
   Error Class = OBJECT,
   Error Code = UNSUPPORTED_OBJECT_TYPE
   'First Failed Element Number' = 0
   ELSE
   RECEIVE CreateObject-Error,
   Error Class = (any valid error class),
   Error Code = (any valid error code)
   ‘First Failed Element Number’ = 0
3. VERIFY (the IUT's Device object),
   Object_List = (any object list that does not contain the object specified in step 1)

Notes to tester: If the IUT limits the instances that can be created, this shall be taken into account when selecting an object identifier in step 1.

9.17 DeleteObject Service Execution Tests

9.17.2 Negative DeleteObject Service Execution Tests

9.17.2.1 Attempting to Delete an Object That is Not Deletable

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the correct response to an attempt to delete an object that is not deletable.

Configuration Requirements: The IUT shall be configured with an object X that cannot be deleted.

Test Steps:
1. *READ V1 = Object_Name*
2. TRANSMIT DeleteObject-Request,
   'Object Identifier' = X
3. RECEIVE BACnet-Error-PDU,
   Error Class = OBJECT,
   Error Code = OBJECT_DELETION_NOT_PERMITTED
4. VERIFY (X), Object_Name = *V1* ~~(the Object_Name specified in the EPICS)~~
5. VERIFY (X), Object_List = (any object list that contains X)

9.18 ReadProperty Service Execution Tests

9.18.1 Positive ReadProperty Service Execution Tests

9.18.1.2 Reading a Single Element of an Array

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array and a single element of the array is requested.

Test Steps:
1. READ V = (Device, X), Object_List ARRAY_INDEX=1
2. CHECK (V is of type object-identifier)

~~1. VERIFY (Device, X),~~
~~Object_List = (the first element of the Object_List array as specified in the EPICS),~~
~~ARRAY INDEX = 1~~

~~Passing Result: The returned value should be of type object-identifier.~~

9.18.1.X1 Reading Properties Based on Data Type

Reason for Change: A general ReadProperty test is not supplied by 135.1 that can be used in a variety of situations. The BTL-WG has kept this test to ensure that all data types are tested. Modified test to remove dependency on EPICS values.

Purpose: This test case verifies that the IUT can execute ReadProperty service requests for requested properties of each of the supported base data types.

Test Concept: This test is repeated once for each base data type that the IUT supports. For each execution of the test a property, P1, shall be selected that is of the data type being tested and the object containing P1 is designated Object1 in the test description.

Test Steps:
1. READ V = (Object1), P1
2. CHECK (V returns any valid value of the correct data type for property P1)

9.18.1.X3 Respects max-segments-accepted bit pattern

Reason for Change: There is no SSPC proposal for this change.

Purpose: To verify that the IUT abides by the 'max-segments-accepted' parameter, when the size of the response does require segmentation.

Configuration Requirements: Use a very small 50 octet 'max-APDU-length-accepted' size in the request. The BACnet-Confirmed-Request-PDU shall be one where the response size will exceed 2 times 'max-APDU-length-accepted' and so

require at least three segments. If the largest response that the IUT can return is 100 or fewer octets, then this test shall be skipped.

Test Steps:

1. TRANSMIT BACnet-Confirmed-Request-PDU,
   'segmented-response-accepted' = TRUE
   'max-segments-accepted' = 2
2. RECEIVE BACnet-Abort-PDU,
   'Abort Reason' = BUFFER_OVERFLOW

Hints to Tester: An attempt to read the whole Object_List might suffice. Or a ReadRange or ReadPropertyMultiple or AtomicReadFile request, if any of those services are executed.

9.20 ReadPropertyMultiple Service Execution Tests

9.20.1 Positive ReadPropertyMultiple Service Execution Tests

9.20.1.1 Reading a Single Property from a Single Object

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to read a single property from a single object.

Test Concept: A single supported property is read from the Device object. The property is selected by the TD and is designated as P1 in the test description.

Test Steps:
1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1 | Object2,
   'Property Identifier' = P1
2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = (the object selected in step 1),
   'Property Identifier' = P1,
   'Property Value' = (*any valid value*~~the value of P1 specified in the EPICS~~)

9.20.1.2 Reading Multiple properties from a Single Object

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to read multiple properties from a single object.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1 | Object 2,
   'Property Identifier' = P1,
   'Property Identifier' = P2

-- ... (Two properties are required but more may be selected.)

2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = (the object selected in step 1),
   'Property Identifier' = P1,
   'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),
   'Property Identifier' = P2,
   'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~)

-- … (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

9.20.1.3 Reading a Single Property from Multiple Objects

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to read a single property from multiple objects.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Object Identifier' = Object2,
   'Property Identifier' = P2

-- … (Two properties are required but more may be selected.)

2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),
   'Object Identifier' = Object2,
   'Property Identifier' = P2,
   'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~)

-- … (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

9.20.1.4 Reading Multiple Properties from Multiple Objects

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to read multiple properties from multiple objects.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Identifier' = P2,
   'Property Identifier' = P3,
   'Object Identifier' = Object2,
   'Property Identifier' = P4,
   'Property Identifier' = P5,
   'Property Identifier' = P6

-- … (Two objects must be included but but more may be selected.)

2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),
   'Property Identifier' = P2,
   'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~),
   'Property Identifier' = P3,
   'Property Value' = (*any valid value for P3*~~the value of P3 specified in the EPICS~~),
   'Object Identifier' = Object2,
   'Property Identifier' = P4,
   'Property Value' = (*any valid value for P4*~~the value of P4 specified in the EPICS~~)
   'Property Identifier' = P5,
   'Property Value' = (*any valid value for P5*~~the value of P5 specified in the EPICS~~),
   'Property Identifier' = P6
   'Property Value' = (*any valid value for P6*~~the value of P6 specified in the EPICS~~)

-- … (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains a specification for an unsupported property.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Identifier' = P2,
   'Property Identifier' = (any property, P3, not supported in this object),
   'Property Identifier' = P4
2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),
   'Property Identifier' = P2,
   'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~),
   'Property Identifier' = P3,
   'Error Class' = PROPERTY,
   'Error Code' = UNKNOWN_PROPERTY,
   'Property Identifier' = P4,
   'Property Value' = (*any valid value for P4*~~the value of P4 specified in the EPICS~~)

9.20.1.6 Reading Multiple Properties with Multiple Embedded Access Errors

Reason For Change: Modified Test to remove dependency on EPICS values.

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for multiple unsupported properties.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Identifier' = P2,
   'Property Identifier' = (any property, P3, not supported in this object),
   'Property Identifier' = (any property, P4, not supported in this object),
   'Object Identifier' = (any non-existent object, Object2, which is of a type supported by the IUT),
   'Property Identifier' = P5,
   'Property Identifier' = P6
2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (*any valid value for P1*~~the value of P1 specified in the EPICS~~),
   'Property Identifier' = P2,
   'Property Value' = (*any valid value for P2*~~the value of P2 specified in the EPICS~~),
   'Property Identifier' = P3,
   'Error Class' = PROPERTY,
   'Error Code' = UNKNOWN_PROPERTY,
   'Property Identifier' = P4,
   'Error Class' = PROPERTY,

'Error Code' = UNKNOWN_PROPERTY,
'Object Identifier' = Object2,
'Property Identifier' = P5,
'Error Class' = OBJECT,
'Error Code' = (UNKNOWN_OBJECT),
'Property Identifier' = P6,
'Error Class' = OBJECT,
'Error Code' = (UNKNOWN_OBJECT)

9.20.1.7 Reading ALL Properties

Reason for Change: Modified test to remove dependency on EPICS values. Addendum 135-2008x. Addendum 135-2010ao-5.

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier ALL. One instance of each object-type supported is tested.

Test Steps:
1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = ObjectX,
        'Property Identifier' = ALL
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = ObjectX,
    REPEAT P = (each property supported by ~~Object1~~*ObjectX*) DO {
        'Property Identifier' = P,
        'Property Value' = (*any valid value for P*~~the value of P specified in the EPICS~~)
    }
}

Notes to Tester: Any proprietary properties that are supported for the object-type shall also be returned (see BACnet 15.7.3.1.2). *If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ_ACCESS_DENIED for that property. Property_List(371) shall not appear in the List of Results.*

9.20.1.8 Reading OPTIONAL Properties

Reason for Change: Modified test to remove dependency on EPICS values. Addendum 135-2008x.

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier OPTIONAL. One instance of each object-type supported is tested. *The property identifier OPTIONAL means that only those standard properties present in the object that have a conformance code "O" shall be returned.*

Test Steps:
1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = ~~Object1~~*ObjectX*,
        'Property Identifier' = OPTIONAL
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = ~~Object1~~*ObjectX*,
    REPEAT P = (each optional property supported by ~~Object1~~*ObjectX*) DO {
        'Property Identifier' = P,
        'Property Value' = (*any valid value for P*~~the value of P specified in the EPICS~~)
    }
}

Notes to Tester: If no optional properties are supported then an empty 'List of Results' shall be returned for the specified property. *If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol_Revision >= 7, then the entry shall contain Error Class: PROPERTY and 'Error-Code': READ_ACCESS_DENIED for that property.*

9.20.1.9 Reading REQUIRED Properties

Reason for Change: Modified test to remove dependency on EPICS values. Addendum 135-2008x. Addendum 135-2010ao-5

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier REQUIRED. One instance of each object-type supported is tested. *The property identifier REQUIRED means that only those standard properties having a conformance code of "R" or "W" shall be returned.*

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
    TRANSMIT ReadPropertyMultiple-Request,
        'Object Identifier' = ObjectX,
        'Property Identifier' = REQUIRED
    RECEIVE ReadPropertyMultiple-ACK,
        'Object Identifier' = ObjectX,
    REPEAT P = (each required property defined for ~~Object1~~*ObjectX*) DO {
        'Property Identifier' = P,
        'Property Value' = (*any valid value for P*~~the value of P specified in the EPICS~~)
    }
}

Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ_ACCESS_DENIED for that property. Property_List (371) shall not appear in the List of Results.

9.20.1.X1 Reading Properties Based on Data Type

Reason For Change: A general ReadPropertyMultiple test is not supplied by 135.1 that can be used in a variety of situations. This test is not in any SSPC proposal.

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests for requested properties of each of the supported base data types.

Test Concept: The test 9.18.1.X1 Reading Properties Based on Data Type is repeated using ReadPropertyMultiple instead of ReadProperty.

9.21 ReadRange Service Execution Tests

9.21.1 Positive ReadRange Service Execution Tests

9.21.1.X1 ReadRange Support for All List Properties

Reason for change: Need a ReadRange test for non-Log_Buffer list properties.

Purpose: To verify that all list properties of all objects can be read using the 3 by position forms of the ReadRange service.

Test Steps:

```
1.  REPEAT X = (all objects in the IUT's database) DO {
        REPEAT Y = (all list properties in object X) DO {
            TRANSMIT ReadRange-Request
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
            RECEIVE ReadRange-ACK
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
                        'Result Flags' =   (?, ?, ?),
                        'Item Count' =     (C: up to number of  items in Y)
                        'Item Data' =      (the first C elements of Y)
            TRANSMIT ReadRange-Request
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
                        'Reference Index' =    1,
                        'Count' =          (C: any valid positive value)
            RECEIVE ReadRange-ACK
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
                        'Result Flags' =   (TRUE, ?, ?),
                        'Item Count' =     (C2: up to C)
                        'Item Data' =      (the first C2 elements of Y)
            TRANSMIT ReadRange-Request
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
                        'Reference Index' =    (the number of elements in Y),
                        'Count' =          (C: any valid negative value)
            RECEIVE ReadRange-ACK
                        'Object Identifier' =     X,
                        'Property Identifier' =  Y,
                        'Result Flags' =   (?, TRUE, ?),
                        'Item Count' =     (C2: up to abs(C))
                        'Item Data' =      (the last C2 elements of Y)
            }
        }
```

9.21.2 Negative ReadRange Service Execution Tests

9.21.2.1 Attempting to Read a Property That Does not Exist

Reason For Change: 135-2008u-3.

Purpose: To verify the correct execution of the ReadRange service request when the requested property does not exist. This test is only applied to devices with a Protocol_Revision of 10 or higher.

Test Steps:

```
1.  TRANSMIT ReadRange-Request,
        'Object Identifier' =     (any object that exists in the IUT),
        'Property Identifier' =  (any list property not supported by the IUT),
2.  RECEIVE BACnet-Error-PDU,
        'Error Class' =           PROPERTY,
        'Error Code' =            UNKNOWN_PROPERTY
```

9.21.2.2 Attempting to Read a Property That is not a List

Reason For Change: 135-2008u-3. Corrected the error class returned from test

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not a list. This test is only applied to devices with a Protocol_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,
   'Object Identifier' = (any object that exists in the IUT),
   'Property Identifier' = (any non-list property supported by and present in the IUT),
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = ~~PROPERTY~~, *SERVICES*,
   'Error Code' = PROPERTY_IS_NOT_A_LIST

9.21.2.3 Attempting to Read a non-Array Property with an Array Index

Reason For Change: 135-2008u-3.

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not an array of lists. This test is only applied to devices with a Protocol_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,
   'Object Identifier' = (any object that exists in the IUT),
   'Property Identifier' = (any non-array list property supported by and present in the IUT),
   'Array Index' = (any valid value)
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = PROPERTY,
   'Error Code' = PROPERTY_IS_NOT_AN_ARRAY

9.22 WriteProperty Service Execution Tests

9.22.1 Positive WriteProperty Service Execution Tests

9.22.1.1 Writing a Single Element of an Array

Reason for Change: Modified test to remove dependency on EPICS values.
Purpose: To verify that the IUT can execute WriteProperty service requests when the property is an array and a single array element is written.

Test Concept: The TD shall select an object in the IUT that contains a writable array property. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writing array values it shall be configured with at least one writable property that can be used for this test.

Test Steps:
1. READ X = (Object1), P1 ARRAY INDEX = (any value N: 1 ≤ N ≤ the size of the array)
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Array Index' = *N*~~(any value N: 1 ≤ N ≤ the size of the array)~~
   'Property Value' = (any *valid* value of the correct datatype *subject to the restrictions specified*

in the EPICS as defined in 4.4.2 for this array, except the value *X* read for this element in step 1)

3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2), ARRAY INDEX = N

9.22.1.2 Writing a Commandable Property Without a Priority

Reason for Change: Modified test to remove dependency on EPICS values.
Purpose: To verify that the IUT can execute WriteProperty service requests when the property is commandable but a priority is not specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is commandable and has no internal algorithm writing to it at priority 16. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports commandable properties that have no internal algorithm writing at priority 16, it shall be configured with at least one such property that can be used for this test.

Test Steps:
1. READ X = (Object1), Priority_Array, ARRAY INDEX = 16
~~1. VERIFY (Object1), Priority_Array =(the value defined for this property in the EPICS), ARRAY INDEX = 16~~
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = Present_Value,
   'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1)
3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), Priority_Array = (the value used in step 2), ARRAY INDEX = 16

9.22.1.3 Writing a Non-Commandable Property with a Priority

Reason for Change: Modified test to remove dependency on EPICS values.
Purpose: To verify that the IUT can execute WriteProperty service requests when the property is not commandable but a priority is specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is not commandable and has no internal algorithm writing to it. If no suitable property can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports non-commandable properties that have no internal algorithm writing to them, it shall be configured with at least one such property that can be used for this test.

Test Steps:
1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Priority' = (any valid priority)
   'Property Value' = (any valid value defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1)
3. RECEIVE BACnet-BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

9.22.1.X1 Writing an Array Size

Reason For Change: No test exists for this functionality. This test was covered by CN-039 but the SSPC rejected the test in favour of the tests outlined in WS-030. The BTL-WG has chosen to keep this specific test in order to allow the tester to test individual properties. Modified this test to remove dependency on EPICS values.

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to the array size of a writable, non-fixed size array property.

Test Concept: The TD shall select an object in the IUT that contains a writable array property of a non-fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writable non-fixed size array properties it shall be configured with at least one writable non-fixed size array property that can be used for this test.

Test Steps:
1. READ X = (Object1), P1 ARRAY INDEX = 0
~~1. VERIFY (Object1), P1[0] = (the array size defined for this array property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Array Index' = 0
   'Property Value' = (any valid array size defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1)
3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), P1[0] = (the value used in step 2)

9.22.1.X2 Writing to Properties Based on Data Type

Reason for Change: A general WriteProperty test is not supplied by 135.1 that can be used in a variety of situations. The BTL-WG has kept this test to ensure that all data types are tested.

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to specific data types supported by the IUT.

Test Concept: For the specified base data type, the TD shall select an object in the IUT that contains a writable property of that data type. This property is designated P1.

Configuration Requirements: The IUT shall be configured with at least one writable property of the specified data type to be used for this test.

Test Steps:
1. X = READ (Object1), P1
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any valid value defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X determined in step 1)
3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

9.22.2 Negative WriteProperty Service Execution Tests

9.22.2.1 Writing Non-Array Properties with an Array Index

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can execute WriteProperty service requests when the property value is not an array but an array index is included in the service request.

Test Concept: The TD shall select an object in the IUT that contains a writable scalar property designated P1. An attempt will be made to write to this property using an array index. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:
1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions
        specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),
    'Property Array Index' = (any positive integer)
3. IF (Protocol_Revision is present and Protocol_Revision **>=** 4) THEN
    RECEIVE BACnet-Error PDU,
        Error Class = PROPERTY,
        Error Code = PROPERTY_IS_NOT_AN_ARRAY
   ELSE
    RECEIVE BACnet-Error PDU,
        Error Class = SERVICES,
        Error Code = INCONSISTENT_PARAMETERS
4. VERIFY (Object1), P1 = *X*~~(the value defined for this property in the EPICS)~~

9.22.2.2 Writing Array Properties with an Array Index that is Out of Range

Reason for Change: Modified test to remove dependency on EPICS values. Fixed array index descrption per BTL-CR-0373.

Purpose: To verify that the IUT can execute WriteProperty service requests when the requested property value is an array but the array index is out of range.

Test Concept: The TD shall select an object in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an array index that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:
1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WriteProperty-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any value of the correct datatype for this property subject to the restrictions
        specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),

'Property Array Index' = (any *value* ~~positive integer~~ that is larger *than*~~that~~ the *current*~~supported~~ size *of*~~if~~ the array)

3. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = INVALID_ARRAY_INDEX
4. VERIFY (Object1), P1 = *X*~~(the value defined for this property in the EPICS)~~

9.22.2.3 Writing with a Property Value Having the Wrong Datatype

Reason for Change: Updated Test Concept and Added Configuration Requirements.

Purpose: To verify that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but ~~which is invalid for the property~~ *which is not compliant with the property definition given by the BACnet standard*.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value with an invalid datatype)
3. RECEIVE
   (BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_PARAMETER_DATATYPE) |
   (BACnet-Reject-PDU
   Reject Reason = INVALID_TAG)
4. VERIFY (Object1), P1 = ~~V~~

9.22.2.4 Writing with a Property Value that is Out of Range

Reason for Change: Modified test to remove dependency on EPICS values.

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

Test Concept: The TD attempts to write to a property using a value that is outside of the supported range. If the IUT does not contain any writable properties that have restricted ranges, then this test shall be skipped.

Test Steps:

1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),~~
2. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (Object1, any object with writable properties),
   'Property Identifier' = (P1, any writable property with a restricted range of values),
   'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. IF (Protocol_Revision is present and Protocol_Revision **>=** 4) THEN
   RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,

```
        Error Code = VALUE_OUT_OF_RANGE
    ELSE
        RECEIVE (BACnet-Error-PDU,
            Error Class = PROPERTY,
            Error Code = VALUE_OUT_OF_RANGE) |
        (BACnet-Reject-PDU,
            Reject Reason = PARAMETER_OUT_OF_RANGE)
4.  VERIFY (Object1), P1 =   X~~(the value defined for this property in the EPICS)~~
```

Notes to tester: The value used in step 2 shall be of the correct datatype. For bit string types, the bit count shall be correct, for Date and Time values, the value shall be within the range defined by the standard for the datatype, for constructed values, the constructed value shall match the structure defined by the ASN.1 and all field values shall be within the ranges defined by the standard for those field values.

9.22.2.X1 Writing Non-Array Read-only Property with an Array Index

Reason for Change: Existing test 9.22.2.1 forbids the testing of a read-only property, to observe the response when an array index is included in the service request.

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value when the property value is not an array but an array index is included in the service request, and the property specified in the service request is not writable.

Test Concept: Select an object, designated Object1, in the IUT that contains a non-writable scalar property designated P1. An attempt will be made to write to this property with an array index included. If no object supports non-writable scalar properties, then this test shall be omitted.

Test Steps:

```
1.  TRANSMIT WriteProperty-Request,
        'Object Identifier' = Object1,
        'Property Identifier' = P1,
        'Property Value' = (any value of the correct datatype for this property)
        'Property Array Index' = (any positive integer)
2.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE BACnet-Error PDU,
            Error Class = PROPERTY,
            Error Code = WRITE_ACCESS_DENIED | PROPERTY_IS_NOT_AN_ARRAY
    ELSE
        RECEIVE (BACnet-Error PDU,
            Error Class = SERVICES,
            Error Code = INCONSISTENT_PARAMETERS) |
        (BACnet-Error PDU,
            Error Class = PROPERTY,
            Error Code =  WRITE_ACCESS_DENIED | PROPERTY_IS_NOT_AN_ARRAY)
```

9.23 WritePropertyMultiple Service Execution Tests

9.23.1 Positive WritePropertyMultiple Service Execution Tests

9.23.1.1 Writing a Single Property to a Single Object

Reason for Change: Modified test to remove dependency on EPICS values

Purpose: To verify the ability to write a single property to a single object.

Test Concept: This test case attempts to write to a single scalar property, P1, that is not commandable. If no such writable property exists the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation.

Test Steps:
1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1)
3. RECEIVE BACnet-Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value specified in step 2)

9.23.1.2 Writing Multiple properties to a Single Object

Reason for Change: Modified test to remove dependency on EPICS values
Purpose: To verify the ability to write multiple properties to a single object.

Test Concept: This test case attempts to write to multiple scalar properties, P1 and P2, that are not commandable. If two such writable properties don't exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any object that has two writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured, if possible, with writable array or commandable properties and the test steps modified to account for this variation. If no object type is supported that has two or more writable properties this test may be omitted. The IUT must support either the configuration required for this test or a configuration required for test 9.23.1.3

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
~~2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)~~
3. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),
    'Property Identifier' = P2,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y* ~~except for the one~~ read in step 2)
4. RECEIVE BACnet-Simple-ACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step ~~2~~*3*)
6. VERIFY (Object1), P2 = (the value specified for P2 in step ~~2~~*3*)

9.23.1.3 Writing a Single Property to Multiple Objects

Reason for Change: Modified test to remove dependency on EPICS values

Purpose: To verify the ability to write a single property from multiple objects.

Test Concept: This test case attempts to write to single scalar properties, P1 and P2, that reside in different objects but are not commandable. If two such writable properties don't exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:
1. *READ X = (Object1), P1*
2. *READ Y = (Object2), P2*

~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
~~2. VERIFY (Object2), P2 = (the value specified for this property in the EPICS)~~

3. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),
   'Object Identifier' = Object2,
   'Property Identifier' = P2,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y* ~~except for the one~~ read in step 2)
4. RECEIVE BACnet-Simple-ACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object2), P2 = (the value specified for P2 in step 3)

9.23.1.4 Writing Multiple Properties to Multiple Objects

Reason for Change: Modified test to remove dependency on EPICS values

Purpose: To verify the ability to write multiple properties to multiple objects.

Test Concept: This test case attempts to write properties, P1 and P2, that reside in Object1, and properties P3 and P4 that reside in Object2. P1, P2, P3 and P4 are not commandable properties. If four such writable properties do not exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:
1. *READ X = (Object1), P1*
2. *READ Y = (Object1), P2*
3. *READ Z = (Object2), P3*
4. *READ A = (Object2), P4*

~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
~~2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)~~
~~3. VERIFY (Object2), P3 = (the value specified for this property in the EPICS)~~
~~4. VERIFY (Object2), P4 = (the value specified for this property in the EPICS)~~

5. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),
   'Property Identifier' = P2,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y* ~~except for the one~~ read in step 2),
   'Object Identifier' = Object2,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Z* ~~except for the one~~ read in step 3),
   'Object Identifier' = Object2,
   'Property Identifier' = P4,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions*

specified in the EPICS as defined in 4.4.2, except the value A ~~except for the one~~ read in step 4)

6. RECEIVE BACnet-BACnet-SimpleACK-PDU
7. VERIFY (Object1), P1 = (the value specified for P1 in step 5)
8. VERIFY (Object1), P2 = (the value specified for P2 in step 5)
9. VERIFY (Object2), P3 = (the value specified for P3 in step 5)
10. VERIFY (Object2), P4 = (the value specified for P4 in step 5)

9.23.1.X4 Writing an Array Size

Reason For Change: No test exists for this functionality. This test is not contained in any SSPC proposal.

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests to the array size of a writable, non-fixed size array property.

Test Concept: Repeat test 9.22.1.X1 Writing an Array Size using WritePropertyMultiple instead of WriteProperty.

9.23.2 Negative WritePropertyMultiple Service Execution Tests

9.23.2.1 Writing Multiple Properties with a Property Access Error

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for an unsupported property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is not supported for this object. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the unsupported property used for this test.

Test Steps:

1. READ X = (Object1), P1

~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~

2. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),
   'Property Identifier' = P2,
   'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2*)
3. RECEIVE WritePropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = UNKNOWN_PROPERTY,
   'Object Identifier' = Object1,
   'Property Identifier' = P2
4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

9.23.2.2 Writing Multiple Properties with an Object Access Error

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write

Access Specifications' contains a specification for an unsupported object.

Test Concept: An attempt is made to write to a single property in two different objects. The first object is supported and the property is writable. The second object is not supported. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 and P1 will be used to designate the writable object and property used for this test. The designation BadObject will be used to indicate an object that is not supported.

Test Steps:
1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),
    'Object Identifier' = BadObject,
    'Property Identifier' = P2,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2*)
3. RECEIVE WritePropertyMultiple-Error,
    'Error Class' = OBJECT,
    'Error Code' = UNKNOWN_OBJECT,
    'Object Identifier' = BadObject,
    'Property Identifier' = P2
4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

9.23.2.3 Writing Multiple Properties with a Write Access Error

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for a read only property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is supported but read only. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the read only property used for this test.

Test Steps:
1. READ X = (Object1), P1
2. READ Y = (Object1), P2
~~1. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)~~
~~2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)~~
3. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = Object1,
    'Property Identifier' = P1,
    'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X* ~~except for the one~~ read in step 1),

'Property Identifier' = P2,
'Property Value' = (any *valid* value of the appropriate datatype *for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y* ~~except for the one~~ read in step 1)

4. RECEIVE WritePropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = WRITE_ACCESS_DENIED,
   'Object Identifier' = Object1,
   'Property Identifier' = P2
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object1), P2 = *Y*~~(the value specified for this property in the EPICS)~~

9.23.2.4 Writing Non-Array Properties with an Array Index

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property value is not an array but an array index is included in the service request. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable scalar property designated P1. An attempt will be made to write to this property using an array index. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1

~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~

2. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),
   'Property Array Index' = (any positive integer)
3. RECEIVE WritePropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = PROPERTY_IS_NOT_AN_ARRAY,
   'Object Identifier' = Object1,
   'Property Identifier' = P1
4. VERIFY (Object1), P1 = *X*~~(the value defined for this property in the EPICS)~~

9.23.2.5 Writing Array Properties with an Array Index that is Out of Range

Reason for Change: Modified test to remove dependency on EPICS values. Fixed array index descrption per BTL-CR-0373.

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the requested property value is an array but the array index is out of range. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an array index that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1

~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~

2. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any *valid* value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value *X* read in step 1),
   'Property Array Index' = (any *value*~~positive integer~~ that is larger *than*~~that~~ the *current*~~supported~~ size of the array)
3. RECEIVE WritePropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = INVALID_ARRAY_INDEX,
   'Object Identifier' = Object1,
   'Property Identifier' = P1
4. VERIFY (Object1), P1 = *X*~~(the value defined for this property in the EPICS)~~

9.23.2.6 Writing with a Property Value Having the Wrong Datatype

Reason for Change: Added configuration requirements to clarify usage.

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but ~~which is invalid for the property~~ *which is not compliant with the property definition given by the BACnet standard.*

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value with an invalid datatype)
3. RECEIVE WritePropertyMultiple-Error,
   'Error Class' = PROPERTY,
   'Error Code' = INVALID_DATATYPE,
   'Object Identifier' = Object1,
   'Property Identifier' = P1
   | (BACnet-Reject-PDU
   'Reject Reason' = INVALID_PARAMETER_DATATYPE)
   | (BACnet-Reject-PDU
   'Reject Reason' = INVALID_TAG)
4. VERIFY (Object1), P1 = *X*~~(the value defined for this property in the EPICS)~~

9.23.2.7 Writing with a Property Value that is Out of Range

Reason for Change: Modified test to remove dependency on EPICS values. Modified to allow this test to be used on all protocol revisions.

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that is outside of the supported range~~. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.~~

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. The TD attempts to write to a property using a value that is outside of the supported range.

Test Steps:

1. READ X = (Object1), P1
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),~~
2. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = (Object1, any object with writable properties),
    'Property Identifier' = (P1, any property with a restricted range of values),
    'Property Value' = (any value that is outside the supported range)
3. IF (Protocol_Revision < 4)
    RECEIVE
        (WritePropertyMultiple-Error,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE,
            'Object Identifier' = Object1,
            'Property Identifier' = P1) |
        (BACnet-Reject-PDU,
            'Reject Reason' = PARAMETER_OUT_OF_RANGE)
ELSE
    RECEIVE
        WritePropertyMultiple-Error,
            'Error Class' = PROPERTY,
            'Error Code' = VALUE_OUT_OF_RANGE,
            'Object Identifier' = Object1,
            'Property Identifier' = P1
~~4. VERIFY (OBJECT1), P1 = (the value defined for this property in the EPICS)~~

9.23.2.X1 WritePropertyMultiple Reject Test

Reason for Change: Addendum 135-2008u section 1.

Purpose: This test case verifies that the IUT does not send a Reject-PDU after applying part of a WritePropertyMultiple.

Test Concept: Two writable properties, P1 and P2 are written to the IUT but the portion of the WritePropertyMultiple specifying P2 is made invalid by omitting the ‘Property Value’ parameter. If the IUT returns a Reject, then the value of the first property is checked to ensure it has not changed.

Test Steps:
1. READ OldValue = O1, P1
2. TRANSMIT WritePropertyMultiple-Request,
    'Object Identifier' = O1,
    'Property Identifier' = P1,
    'Property Value' = (NewValue: any value other than OldValue that would be accepted by the IUT for P1)
    'Object Identifier' = O2,
    'Property Identifier' = P2
3. RECEIVE WritePropertyMultiple-Error,
    'Error Class' = SERVICES,
    'Error Code' = INVALID_TAG
    'Object Identifier' = O2

'Property Identifier' = P2) |
RECEIVE BACnet-Reject-PDU,
'Reject Reason' = INVALID_TAG | MISSING_REQUIRED_PARAMETER |
INCONSISTENT_PARAMETERS | INVALID_PARAMETER_DATA_TYPE |
TOO_MANY_ARGUMENTS)

4. IF (a WritePropertyMultiple-Error was received in step 3) THEN
   VERIFY (O1), P1 = NewValue
   ELSE -- a Reject-PDU was received
   VERIFY (O1), P1 = OldValue

9.24 DeviceCommunicationControl Service Execution Test

9.24.1 Positive DeviceCommunicationControl Service Execution Tests

9.24.1.5 Finite Time Duration Restored by ReinitializeDevice

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when a finite time duration is specified and communication is restored using the ReinitializeDevice service.

Test Steps:

1. *READ Y = (Device, X), Object_Name*
2. TRANSMIT DeviceCommunicationControl-Request,
   'Time Duration' = (a value T > 1, in minutes, selected by the tester)
   'Enable/Disable' = DISABLE,
   'Password' = (any appropriate password as described in the Test Concept)
3. RECEIVE BACnet-SimpleACK-PDU
4. WAIT **Internal Processing Fail Time**
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (any required non-array property of the Device object)
6. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester, and < T as specified in the DeviceCommunicationControl-Request)
7. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
8. TRANSMIT ReinitializeDevice-Request,
   'Reinitialize State of Device' = WARMSTART,
   'Password' = (any appropriate password as described in the Configuration Requirements)
9. RECEIVE BACnet-Simple-ACK-PDU
10. CHECK (Did the IUT perform a WARMSTART reboot?)
11. VERIFY (Device, X), *Object_Name = Y*~~(any required non-array property) = (the value for this property as described in the EPICS)~~

9.24.2 Negative DeviceCommunicationControl Service Execution Tests

9.24.2.3 Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'

Reason for Change: Added support for additional error codes per Addendum 12.0c-7.

Purpose: To verify the communications are not restored when a ReinitializeDevice request is received that contains one of the backup or restore related values for service parameter 'Reinitialized State of Device'.

Test Concept: Disable the IUT's communications for a period time, T, longer than it will take to complete the test. Verify that, while communications are disabled, the IUT correctly responds with a Result(-) when it receives a ReinitializeDevice request containing a backup or restore related values.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE
   'Password' = (any appropriate password),
   'Time Duration' = (a value T >= 1, in minutes) | (no value)
2. RECEIVE BACnet-Simple-ACK-PDU
3. WAIT **Internal Processing Fail Time**
4. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP | ENDBACKUP |
   STARTRESTORE | ENDRESTORE | ABORTRESTORE,
   'Password' = (any appropriate password)
5. IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN
   IF (Device supports DM-BR-B) THEN
   RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = COMMUNICATION_DISABLED
   ELSE
   RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = COMMUNICATION_DISABLED |
   OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
   ELSE
   CHECK(that the IUT responded with BACnet-Error-PDU with an Error Class of SERVICES and a*ny appropriate* Error Code ~~of COMMUNICATION_DISABLE~~, or *that* the IUT did not respond at all)
6. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = ENABLE
   'Password' = (any appropriate password),
7. RECEIVE BACnet-Simple-ACK-PDU

9.27 ReinitializeDevice Service Execution Tests

9.27.2 Negative ReinitializeDevice Service Execution Tests

9.27.2.3 COLDSTART with Missing *or Invalid* Password

Reason for Change: Updated test to also test invalid password usage per Addendum 12.0g-5.

Purpose: To verify that the correct BACnet Error PDU is returned when a COLDSTART is attempted and*the password is invalid or* a password is required but no password is provided.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = COLDSTART~~,~~
2. IF (*Protocol_Revision is present and* Protocol_Revision >= 7) THEN
   RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE
   ELSE
   (RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE) |
   (RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,

Error Code = SERVICE_REQUEST_DENIED) |
(BACnet-Error-PDU,
Error Class = SERVI~~V~~CES,
Error Code = MISSING_REQUIRED_PARAMETER)

3. CHECK (The IUT did NOT perform a COLDSTART reboot)
4. *TRANSMIT ReinitializeDevice-Request,*
'Reinitialized State of Device' = COLDSTART,
'Password' = (any invalid password)
5. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE
ELSE
(RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE) |
(RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = SERVICE_REQUEST_DENIED) |
(BACnet-Error-PDU,
Error Class = SERVI~~V~~CES,
Error Code = MISSING_REQUIRED_PARAMETER)
6. *CHECK (The IUT did NOT perform a COLDSTART reboot)*

9.27.2.4 WARMSTART with Missing *or Invalid* Password

Reason for Change: Updated test to also test invalid password usage per Addendum 12.0g-5.

Purpose: To verify that the correct BACnet Error PDU is returned when a WARMSTART is attempted and *the password is invalid or* a password is required but no password is provided.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART~~,~~
2. IF (*Protocol_Revision is present and* Protocol_Revision >= 7) THEN
RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE
ELSE
RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE |
(RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = SERVICE_REQUEST_DENIED) |
(BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = MISSING_REQUIRED_PARAMETER)
3. CHECK (The IUT did NOT perform a WARMSTART reboot)
4. *TRANSMIT ReinitializeDevice-Request,*
'Reinitialized State of Device' = WARMSTART,
'Password' = (any invalid password)
5. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,

Error Code = PASSWORD_FAILURE
ELSE
(RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE) |
(RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = SERVICE_REQUEST_DENIED) |
(BACnet-Error-PDU,
Error Class = SERVI~~V~~CES,
Error Code = MISSING_REQUIRED_PARAMETER)

6. *CHECK (The IUT did NOT perform a WARMSTART reboot)*

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

9.29 UnconfirmedTextMessage Service Execution Tests

9.29.1 UnconfirmedTextMessage With No Message Class

Reason for Change: Add test support for the text message services.
Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when no 'Message Class' is provided.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,
   'Text Message Source Device' = TD,
   'Message Priority' = NORMAL,
   'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

9.29.2 UnconfirmedTextMessage With an Unsigned Message Class

Reason for Change: Add test support for the text message services.
Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the Unsigned form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported Unsigned message classes.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,
   'Text Message Source Device' = TD,
   'Message Class' = (any Unsigned value from the list provided by the vendor),
   'Message Priority' = NORMAL,
   'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

9.29.3 UnconfirmedTextMessage With a CharacterString Message Class

Reason for Change: Add test support for the text message services.
Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the CharacterString form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported CharacterString message classes.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,
   'Text Message Source Device' = TD,
   'Message Class' = (any CharacterString value from the list provided by the vendor),
   'Message Priority' = NORMAL,
   'Message' = (any CharacterString)
2. CHECK(Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is

9.30 TimeSynchronization Service Execution Tests

Dependencies: ReadProperty Service Execution tests, 9.18.

BACnet Reference Clause: 16.7.

9.30.1 Positive TimeSynchronization Service Execution Tests

The purpose of this test group is to verify correct execution of TimeSynchronization service requests under circumstances where the service is expected to be successfully completed.

9.30.1.1 TimeSynchronization Local Broadcast

Reason for change: UTC_Offset and Daylight_Savings_Status are optional properties that are only required for the UTCTimeSynchronization service.

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast TimeSynchronization service request.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date
2. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date,
   'Property Value' = (any valid date referred to as "InitialDate" below)
3. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time
4. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time,
   'Property Value' = (any valid time referred to as "InitialTime" below)
~~5. TRANSMIT ReadProperty-Request,~~
   ~~'Object Identifier' = (the IUT's Device object),~~
   ~~'Property Identifier' = UTC_Offset~~

~~6. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = UTC_Offset,~~
~~'Property Value' = (any valid offset referred to as "InitialUTC_Offset" below)~~
~~7. TRANSMIT ReadProperty-Request,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Daylight_Savings_Status~~
~~8. RECEIVE ReadProperty-ACK,~~
~~'Object Identifier' = (the IUT's Device object),~~
~~'Property Identifier' = Daylight_Savings_Status,~~
~~'Property Value' = (any valid status referred to as "InitialDaylight_Savings_Status" below)~~

5. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = TimeSynchronization-Request,
   date = (any date other than InitialDate),
   time = (any time that does not correspond to InitialTime)
6. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date
7. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date,
   'Property Value' = (the date specified in step 5)
8. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time
9. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time,
   'Property Value' = (the time specified in step 5)

Notes to Tester: The time value returned by the IUT in step 9 shall agree with the time specified in step 5 within the resolution for time specified in the EPICS. If the time returned by the IUT indicates that a small amount of time has passed (< 1 second) since the TimeSynchronization request was received the result shall be considered to be a pass. If the time indicates that the day of week is unspecified but all other fields are correct the result shall be considered to be a pass.

9.30.1.2 TimeSynchronization Directed to the IUT

Reason for change: UTC_Offset and Daylight_Savings_Status are optional properties that are only required for the UTCTimeSynchronization service.

Purpose: To verify that the IUT resets its local time and date in response to a TimeSynchronization service request directed to the IUT's MAC address.

Test Steps: This test is identical to 9.30.1.1 except that the TimeSynchronization-Request in step ~~9~~5 shall be transmitted using the IUT's MAC address as the destination.

Notes to Tester: The passing results are identical to 9.30.1.1.

9.31 UTCTimeSynchronization Service Execution Tests

BACnet Reference Clause: 16.8.

9.31.1 Positive UTCTimeSynchronization Service Execution Tests

The purpose of this test group is to verify correct execution of UTCTimeSynchronization service request.

9.31.1.1 UTCTimeSynchronization Local Broadcast

Reason for change: UTC_Offset and Daylight_Savings_Status are needed here, as these optional properties are required for the UTCTimeSynchronization service.

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast UTCTimeSynchronization service request.

Test Steps:

1. *TRANSMIT ReadProperty-Request,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date
2. *RECEIVE ReadProperty-ACK,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Date,
   'Property Value' = (any valid date referred to as "InitialDate" below)
3. *TRANSMIT ReadProperty-Request,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time
4. *RECEIVE ReadProperty-ACK,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Local_Time,
   'Property Value' = (any valid time referred to as "InitialTime" below)
5. *TRANSMIT ReadProperty-Request,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = UTC_Offset
6. *RECEIVE ReadProperty-ACK,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = UTC_Offset,
   'Property Value' = (any valid offset referred to as "Initial_UTC_Offset")
7. *TRANSMIT*
   DA = LOCAL BROADCAST,
   SA = TD,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = UTCTimeSynchronization-Request,
   date = (any date other than InitialDate),
   time = (any time that does not correspond to InitialTime)
8. *TRANSMIT ReadProperty-Request,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Daylight_Savings_Status
9. *RECEIVE ReadProperty-ACK,*
   'Object Identifier' = (the IUT's Device object),
   'Property Identifier' = Daylight_Savings_Status,
   'Property Value' = (any valid status)
10. *TRANSMIT ReadProperty-Request,*
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Date
11. *RECEIVE ReadProperty-ACK,*
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Date,
    'Property Value' = (the date specified in step 9, corrected for Initial_UTC_Offset and Daylight_Savings_Status)

12. TRANSMIT ReadProperty-Request,
'Object Identifier' = (the IUT's Device object),
'Property Identifier' = Local_Time
13. RECEIVE ReadProperty-ACK,
'Object Identifier' = (the IUT's Device object),
'Property Identifier' = Local_Time,
'Property Value' = (the time specified in step 9, corrected for Initial_UTC_Offset and Daylight_Savings_Status)

Passing Results: The time value returned by the IUT in steps 11 and 13 shall agree, within the resolution for time specified in the EPICS, with the date and time specified in step 7, corrected for both Initial_UTC_Offset and Daylight_Savings_Status. It is the Daylight_Savings_Status from step 9 which should be used in the determination in steps 11 and 13. The IUT may update the Daylight_Savings_Status during the execution of the UTCTimeSynchronization request. If the time returned by the IUT indicates that a small amount of time has passed (< 1 second) since the UTCTimeSynchronization request was received, then the result shall be considered a pass.

9.31.1.2 UTCTimeSynchronization Directed to the IUT

Reason for change: UTC_Offset and Daylight_Savings_Status are needed here, as these optional properties are required for the UTCTimeSynchronization service.

Test Steps: This test is identical to 9.~~30~~*31*.1.1 except that in step 9 the UTCTimeSynchronization request is used and the date and time conveyed represent UTC and the UTCTimeSynchronization-Request shall be transmitted using the IUT's MAC address as the destination.

Notes to Tester: The passing results are identical to 9.31.1.1.

9.32 Who-Has Service Execution Tests

The purpose of this test group is to verify the correct execution of the Who-Has service request.

Dependencies: None.

BACnet Reference Clause: 16.9.

9.32.1 Execution of Who-Has Service Requests Originating from the Local Network

The purpose of this test group is to verify the correct execution of the Who-Has request service procedure for messages originating from the local network.

9.32.1.1 Object ID Version with No Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and does not restrict device ranges.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 = (Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,

SA = IUT,
I-Have-Request,
'Device Identifier' = (the IUT's Device object),
'Object Identifier' = *Object1*~~(the object identifier specified in step 1),~~
'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.1.2 Object Name Version with no Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and does not restrict device ranges.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Object Name' = *V1*~~(any object name specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in the EPICS for this object)~~,
   'Object Name' = *V1*~~(the object name specified in step 1)~~

9.32.1.3 Object ID Version with IUT Inside of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that includes the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (any value L: 0 ≤ L < the Device object instance number of the IUT),
   'Device Instance High Limit' = (any value H,: H > the Device object instance number of the IUT),
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~,
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
   'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.1.4 Object ID Version with IUT Outside of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT ignores a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that does not include the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (any value > 0: the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
   'Device Instance High Limit' = (any value > Device Instance Low Limit: the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

9.32.1.5 Object Name Version with IUT Inside of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and specifies a device range restriction that includes the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (any value L: 0 ≤ L < the Device object instance number of the IUT),
   'Device Instance High Limit' = (any value H: H > the Device object instance number of the IUT),
   'Object Name = *V1*~~(any object name specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in the EPICS for this object)~~,
   'Object Name' = *V1*~~(the object name specified in step 1)~~

9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object identifier form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (any value L: 0 ≤ L < the Device object instance number of the IUT),
   'Device Instance High Limit' = (The Device object instance number of the IUT),
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
   'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object identifier form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (The Device object instance number of the IUT),
   'Device Instance High Limit' = (any value H: H > the Device object instance number of the IUT),
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
   'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object name form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   Who-Has-Request,
   'Device Instance Low Limit' = (any value L: 0 ≤ L < the Device object instance number of the IUT),
   'Device Instance High Limit' = (The Device object instance number of the IUT),
   'Object Name = *V1*~~(any object name specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in the EPICS for this object)~~,
   'Object Name' = *V1*~~(the object name specified in step 1)~~

9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object name form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (The Device object instance number of the IUT),
   'Device Instance High Limit' = (any value H: H > the Device object instance number of the IUT),
   'Object Name = *V1*~~(any object name specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DA = LOCAL BROADCAST | GLOBAL BROADCAST,
   SA = IUT,
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
   'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.1.11 Object Name Version, Directed to a Specific MAC Address

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify that the IUT responds with a broadcast I-Have service request even if the Who-Has service requests was not transmitted with a broadcast address.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT Who-Has-Request,
   'Object Name' = *V1*~~(any object name specified in the EPICS)~~,
3. WAIT **Internal Processing Fail Time**
4. RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST,
SA = IUT,
I-Have-Request,
'Device Identifier' = (the IUT's Device object),
'Object Identifier' = *Object1*~~(the object identifier specified in the EPICS for this object)~~,
'Object Name' = *V1*~~(the object name specified in step 1)~~

9.32.2 Execution of Who-Has Service Requests Originating from a Remote Network

9.32.2.1 Object ID Version, Global Broadcast from a Remote Network

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability of the IUT to recognize the origin of a globally broadcast Who-Has service request and to respond such that the device originating the request receives the response.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SA = TD,
   DNET = GLOBAL BROADCAST,
   SNET = (any remote network number),
   SADR = (any MAC address valid for the specified network),
   Who-Has-Request,
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified in step 1),
   I-Have-Request,
   'Device Identifier' = (the IUT's Device object),
   'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
   'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.2.2 Object ID Version, Remote Broadcast

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: To verify the ability of the IUT to recognize the origin of a remotely broadcast Who-Has service request and to respond such that the device originating the request receives the response.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:
1. *READ V1 =(Object1), Object_Name*
2. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SA = TD,
   SNET = (any remote network number),
   SADR = (any MAC address valid for the specified network),
   Who-Has-Request,
   'Object Identifier' = *Object1*~~(any object identifier specified in the EPICS)~~
3. WAIT **Internal Processing Fail Time**
4. RECEIVE
   DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified in step 1),

I-Have-Request,
'Device Identifier' = (the IUT's Device object),
'Object Identifier' = *Object1*~~(the object identifier specified in step 1)~~,
'Object Name' = *V1*~~(the object name specified in the EPICS for this object)~~

9.32.2.X3 - Who-Has for Non-existent Object_Name

Reason for Change: No test exists for this functionality. This test is not contained in any SSPC proposal.

Purpose: Verifies correct responses to Who-Has service requests by 'Object Name' when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has service request with 'Object Name' when that named object does not exist in the IUT.

Configuration Requirements: Choose any character string value V1, which is not the Object_Name of any object in the IUT. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

Test Steps:

1. TRANSMIT Who-Has-Request,
   'Object Name' = V1
2. WAIT **Internal Processing Fail Time**
3. CHECK (the IUT does not respond with an I-Have request with 'Object Name' containing V1)

9.32.2.X5 Who-Has for Non-existent Object_Identifier

Reason for Change: No test exists for this functionality. This test is not contained in any SSPC proposal.

Purpose: Verifies correct responses to Who-Has service requests when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has request with that 'Object Identifier' parameter for an object which does not exist.

Configuration Requirements: Choose any standard object (Object1) that does not exist within the IUT, i.e. any unsupported Object Type or any supported Object Type for which the instance does not exist. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = Object_Identifier
2. RECEIVE BACnet-Error-PDU,
   'Error Class' = OBJECT,
   'Error Code' = UNKNOWN_OBJECT
3. TRANSMIT Who-Has-Request,
   'Object Identifier' = Object1
4. WAIT **Internal Processing Fail Time**
5. CHECK (the IUT does not respond with an I-Have request with 'Object Identifier' containing Object1)

9.33 Who-Is Service Execution Tests

9.33.1 Execution of Who-Is Service Requests Originating from the Local Network

9.33.1.3 Local Broadcast, Specific Device Inquiry with IUT Outside of the Device Range

Reason For Change: The allowed device instance range is from 0 - 4194303 and is specified in section and 16.10.1.1.1. The corresponding test incorrectly set the low limit greater than 0 when it should have been greater than or equal to.

Purpose: To verify that the IUT ignores Who-Is requests when it is excluded from the specified device range.

Test Steps:

1. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (any value >= 0 such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
   'Device Instance Range High Limit' = (any value >= Device Instance Low Limit such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

10. NETWORK LAYER PROTOCOL TESTS

10.1 Processing Application Layer Messages Originating from Remote Networks

Reason for Change: The test assumes that the IUT and the TD are located on the same network. For the IUT, the TD appears to be the appropriate router to the network specified in step 1. There is no SSPC proposal for this change. Modified test to remove dependency on EPICS values.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 6.5.4.

Purpose: To verify that the IUT can respond to requests that originate from a remote network.

Test Concept: The TD transmits a ReadProperty-Request message that contains network layer information indicating that it originated from a remote network. The response from the IUT shall include correct DNET and DADR information so that the message can reach the original requester. The MAC layer destination address in the response can be either a *local* broadcast, indicating that the IUT does not know the address of the router, or the MAC address of the ~~appropriate router~~*TD*.

Test Steps:

1. TRANSMIT
   DESTINATION = IUT,
   SOURCE = TD,
   SNET = (any network number that is not the local network),
   SADR = (any valid MAC address consistent with the source network),
   ReadProperty-Request,
   'Object Identifier' = (any supported object),
   'Property Identifier' = (any required property of the specified object)
2. RECEIVE
   DESTINATION = LOCAL BROADCAST | ~~(an appropriate router address)~~*TD*,
   SOURCE = IUT,
   DNET = (the SNET specified in step 1),

DADR = (the SADR specified in step 1),
Hop Count = 255,
ReadProperty-ACK,
'Object Identifier' = (the object specified in step 1),
'Property Identifier' = (the property specified in step 1),
'Property Value' = (*any valid value for this property*~~the value of the specified property as defined in the EPICS~~)

10.2 Router Functionality Tests

10.2.2 Processing Network Layer Messages

10.2.2.7.2 Unknown Network Layer Message Type

Reason for Change: Changed 'Reject Reason' to 'Rejection Reason' to distinguish it from the Reject PDU. Corrected DESTINATION and expected DNET in step 2.

Purpose: To verify that the IUT will reject a network layer message with an unknown message type in the range of message types reserved for use by ASHRAE.

Test Steps:

1. TRANSMIT PORT A,
   DESTINATION = IUT,
   SOURCE = D1A,
   Message Type = (any value in the range reserved for use by ASHRAE that is undefined in the protocol revision claimed by the device)
2. RECEIVE PORT A,
   DESTINATION = ~~TD~~*D1A*,
   SOURCE = IUT,
   Reject-Message-To-Network,
   Reject*ion* Reason = 3 (unknown network layer message type),
   DNET = ~~(any value)~~*1*

10.2.X1 Initiates Network-Number-Is on Startup

Reason for Change: Test added per 135-2008g.
References: 6.4.19, 6.4.20

Purpose: To verify that a router initiates Network-Number-Is on startup for each port with a known network number.

Test Concept: The IUT is reset and the tester verifies that the IUT broadcasts a Network-Number-Is message out each port. The vendor can specify a time, or physically observable event after reset, which marks the time at which IUT has completed its startup sequence, including the sending of the Network-Number-Is messages.

Configuration Requirements: The IUT is configured with a network number for each of its enabled ports. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

Test Steps:

1. MAKE (the IUT reset)
2. BEFORE the IUT has completed its startup sequence
   REPEAT X = (for each enabled port) DO {
      RECEIVE PORT X,
         DESTINATION = LOCAL BROADCAST,

Network-Number-Is,
Network Number = (the configured Network Number for port X)
}

10.2.X2 Routers Execute What-Is-Network-Number

Reason for Change: Test added per 135-2008g.
References: 6.4.19, 6.4.20

Purpose: To verify that a router responds to a What-Is-Network-Number request within 10 seconds.

Test Concept: A What-Is-Network-Number is broadcast on the local network and the tester verifies that the IUT responds with a Network-Number-Is message within 10 seconds.

Configuration Requirements: The IUT knows its network number, N1. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

Test Steps:
1. TRANSMIT What-Is-Network-Number,
   DESTINATION = LOCAL_BROADCAST
2. BEFORE 10s + **Internal Processing Fail Time**
   RECEIVE Network-Number-Is,
   Network Number = (the configured value),
   Configured =(any valid value)

10.6 Non-Router Functionality Tests

10.6.3 Ignore Router Commands

Reason for Change: Changed test to support a Reject or a discard per Addendum 12.0d-4.

~~BACnet Reference Clause: 6.6, 6.6.3.8, 6.6.3.10, 6.6.3.11~~

Purpose: This test case verifies that the non-router IUT will *either* quietly accept and discard network layer router services *or respond with a Reject-Message-To-Network*.

Test Concept: The TD transmits the Initialize-Routing-Table, Establish-Connection-To-Network, and Disconnect-Connection-To-Network services. ~~The IUT is required to silently drop the requests because it is not a router.~~

Test Steps:
1. TRANSMIT
   DA = IUT,
   SA = TD,
   Initialize-Routing-Table
   Number of Ports = 0
2. WAIT **Internal Processing Fail Time**
3. *(*CHECK (that the IUT did not send any packets in response to the Initialize-Routing-Table)*) |*
   (RECEIVE
   DESTINATION = TD,
   SOURCE = IUT,
   Reject-Message-To-Network
   Rejection-Reason = 0 (other) | 3 (unknown))
4. TRANSMIT
   DA = IUT,
   SA = TD,
   Establish-Connection-To-Network
   DNET = DNET3

Termination Time Value = 0

5. WAIT **Internal Processing Fail Time**
6. *(CHECK(that the IUT did not send any packets in response to the Establish-Connection-To-Network)) |*
   (RECEIVE
   DESTINATION = TD,
   SOURCE = IUT,
   Reject-Message-To-Network
   Rejection-Reason = 0 (other) | 3 (unknown))
7. TRANSMIT
   DA = IUT,
   SA = TD,
   Disconnect-Connection-To-Network
   DNET = NET3
8. WAIT Internal Processing Fail Time
9. *(*CHECK(that the IUT did not send any packets in response to the Disconnect-Connection-To-Network)*) |*
   (RECEIVE
   DESTINATION = TD,
   SOURCE = IUT,
   Reject-Message-To-Network
   Rejection-Reason = 0 (other) | 3 (unknown))

10.7 Router Functionality

10.7.2 Router Binding via Application Layer Services

Reason for Change: BTL-CR-0149 modified test to allow for directed unicast who-is requests.
Dependencies: ReadProperty Service Initiation Tests, 8.18, ReadProperty Service Execution Tests, 9.18, Who-Is Service Initiation Tests, 8.34

BACnet Reference Clause: 6.5.3

Purpose: To verify that the IUT can initiate requests to a remote network and respond to requests from a remote network after the IUT uses the Who-Is and I-Am Application Layer services to discover the MAC address of the router to that remote network.

Test Concept: The IUT broadcasts a Who-Is request to discover device D2A and notes the MAC address of the intervening router in the corresponding I-Am reply. The TD transmits a request to a device on the remote network and responds to a request from the remote network without performing any further form of dynamic router binding. If the IUT does not support application layer router binding, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted. The IUT may use the deviceInstanceRange form of Who-Is.

Clause 6.5.3 specifically mentions router binding via Who-Is and does not mention router binding by initiating other application layer services (such as Who-Has) or by lurking and noting the router MAC addresses for incoming application layer requests. For this reason the test only allows for router binding via Who-Is.

Test Steps:

1. MAKE (IUT transmit Who-Is to discover the device on the remote network)
2. RECEIVE
   DA = BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = who-Is
   | (DA = BROADCAST,

SA = IUT,
DNET = DNET2,
DADR= BROADCAST, *or D2A*
Hop Count = 255,
BACnet-Unconfirmed-Request-PDU,
'Service Choice' = who-Is)

3. TRANSMIT
   DA = BROADCAST,
   SA = TD,
   SNET = DNET2,
   SADR = D2A,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = I-Am,
   'I Am Device Identifier' = (device object, instance number of D2A),
   'Max APDU Length Accepted ' = (any valid value),
   'Segmentation Supported' = (any valid value),
   'Vendor ID ' = (any valid value)
4. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)
5. RECEIVE
   DA = TD,
   SA = IUT,
   DNET = DNET2,
   DADR= D2A,
   Hop Count = 255,
   BACnet-Confirmed-Request-PDU,
   'Service Choice' = ReadProperty-Request,
   'Object Identifier' = (O1, any BACnet standard object in D2A),
   'Property Identifier' = (P1, any required property of the specified object)
6. TRANSMIT
   DA = IUT,
   SA = TD,
   SNET = DNET2,
   SADR = D2A,
   BACnet-ComplexACK-PDU,
   'Service ACK Choice' = ReadProperty-ACK,
   'Object Identifier' = O1,
   'Property Identifier' = P1,
   'Property Value' = (any valid value)

10.8 Virtual Routing Functionality Tests

Some devices (typically gateways) can route BACnet packets between a physical BACnet LAN and one or more virtual BACnet LANs that contain one or more virtual BACnet devices. See H.1 and H.2 in the BACnet standard for a description of virtual BACnet LANs and virtual BACnet devices.

This clause defines the tests necessary to demonstrate routing functionality to/from virtual BACnet LANs. The tests assume that the routing device has two ports, one connected to a virtual BACnet LAN containing one or more virtual BACnet devices, and one connected to a physical BACnet LAN. IUT Port 1 is directly connected to Network 1 (a virtual BACnet LAN) and Port 2 is directly connected to Network 2 (a physical BACnet LAN). The logical configuration of the internetwork used for these tests is shown in Figure 10.8.X1. The test descriptions in this clause assume that the TD can physically connect to Network 2 and mimic all of the other devices. An acceptable alternative is to construct an internetwork with real devices as indicated. Logical network 3 shall use a LAN technology that has MAC addresses that are different in length from Network 2.

The logical devices included in the internetwork are:

IUT: implementation under test, a router between Networks 1 and 2
VD1A: virtual device on Network 1
VD1B: virtual device on Network 1
D2C: device on Network 2
D3D: device on Network 3
D4E: device on Network 4
R2-3: router between Network 2 and Network 3

General Configuration Requirements: The IUT shall be configured with routing tables indicating that Network 1 is directly connected to Port 1 and that Network 2 is directly connected to Port 2 as shown in Figure 10.8.X1. The routing table shall contain no other entries. The routing device shall be configured to have one or more virtual devices (VD1A, VD1B, etc.) on Network 1. Although the network numbers 1-3 are used above and below, the tester may configure the network using any legal network numbers and modify the tests accordingly. Furthermore, the tester shall appropriately modify the tests for devices that route to multiple virtual networks simultaneously.

Figure 10.8.X1. Logical internetwork configuration for virtual routing functionality tests

10.8.3 Routing of Unicast APDUs

10.8.3.1 Route Request Message from a Local Device to a Virtual Device and Route Response Message from the Virtual Device to the Local Device

Reason for Change: Added 'Note to Tester' that is missing from 135.1-2013.

Purpose: To verify that the IUT can route a unicast request message from a local device to a virtual device and route the response from the virtual device to the local device.

Note to tester: The destination device (VD1A) can be any virtual device in the IUT.

Test Steps:

1. TRANSMIT,
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (the object identifier of any object in the target device),
   'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)
2. RECEIVE,
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   ReadProperty-ACK,
   'Object Identifier' = (the object identifier used in step 1),
   'Property Identifier' = (the property identifier used in step 1),
   'Property Value' = (the contents of the specified property)
3. TRANSMIT,
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (the object identifier of any object in the target device),
   'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented, but not the same property as in step 1)
4. RECEIVE,
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   ReadProperty-ACK,
   'Object Identifier' = (the object identifier used in step 3),
   'Property Identifier' = (the property identifier used in step 3),
   'Property Value' = (the contents of the specified property)

10.8.3.2 Route Request Message from a Virtual Device to a Local Device

Reason for Change: Updated the notes to tester for clarification.

Purpose: To verify that the IUT can route a unicast request message from a virtual device to a local device.

Test Concept: Make one of the virtual devices generate a unicast request, and verify that the NPCI is correctly formed. This test shall be skipped if none of the IUT's virtual devices can issue a confirmed or unconfirmed request in a unicast message.

Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a unicast message to a particular target device on Network 2.

Notes to Tester: During the test, the TD shall answer any requests that the IUT generates while attempting to locate the route to the target device.

Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a unicast message to a device on a local network. Depending on the capabilities of the IUT this may involve sending a message from the target device to the IUT (unicast or broadcast), writing the network address of the target device to an object property in the IUT, writing the Device ID of the target device to an object property in the IUT, writing the Device Name of the target device to an object property in the IUT, or configuring the IUT using a proprietary method. The IUT may need to broadcast a Who-Is or Who-Has request in order to discover the network address of the target device if the network address is unknown.

Test Steps:

1. RECEIVE,
   DA = TD
   SA = IUT
   SNET = 1,
   SADR = (MAC address of any virtual device *on Network 1*),
   BACnet-Confirmed-Request-PDU or BACnet-Unconfirmed-Request-PDU

10.8.3.5 Unicast Messages That Should Not Be Routed

10.8.3.5.1 Unknown Network

Reason for Change: Added notes to tester for clarity.

Purpose: To verify that the IUT will not attempt to route a message directed to a device on an unknown network if the message was transmitted using a local broadcast MAC address.

Test Concept: Direct at one of the virtual devices a ReadProperty request that is correct in all aspects, except for the network number. Ensure that the virtual device does not reply. The request is sent as a local broadcast so that the IUT will receive it and not attempt to re-route it via another router to the unknown network.

Notes to Tester: Choose a virtual device on Network 1 for this test.

1. TRANSMIT,
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 59001,
   DADR = (the MAC address of the selected virtual device),
   Hop Count = 255,
   ReadProperty-Request,

'Object Identifier' = (any object identifier of an object in the virtual device),
'Property Identifier' = (any property of the specified object)

2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 59001…)
   or Reject-Message-To-Network (Network Number = 59001)
   or any message in response to the Read Property request on Network 2)

10.8.4 Routing of Broadcast APDUs to Virtual Devices

10.8.4.7 Route Remote Broadcast Message from a Virtual Device to a Local Network

Reason for Change: Added Configuration Requirements and Notes to Tester for clarity.

Purpose: To verify that the IUT can route a remote broadcast message from a virtual device to a local physical network.

Test Concept: Make one of the virtual devices generate a remote broadcast directed to the non-virtual network that the IUT is connected to, and verify that it is correctly formulated. This test shall be skipped if none of the IUT's virtual devices can issue a remote broadcast message.

Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a remote broadcast message to Network 2.

Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a broadcast message to a local (physical) network. Depending on the capabilities of the IUT this may involve sending a message from a device on the target network to the IUT (unicast or broadcast), writing a broadcast address to an object property in the IUT, or configuring the IUT using a proprietary method.

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message to the local network of the IUT)
2. RECEIVE,
   DA = LOCAL BROADCAST,
   SA = IUT,
   SNET = 1,
   SADR = (MAC address of a virtual device on Network 1),
   BACnet-Unconfirmed-Request-PDU

10.8.7 Multiple Devices on a Single Virtual Network

Note: If only one virtual device may be configured then VD1B may be any Device ID and MAC address not equal to those of VD1A.

10.8.7.4 Who-Is Specifying Unknown Device Ids

Reason for Change: No test exists for this functionality.

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

1. TRANSMIT,
   DA = IUT,
   SA = TD,
   DNET = 1,

DLEN = 0,
Hop Count = 255,
BACnet-Unconfirmed-Request-PDU,
Who-Is-Request,
'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)
'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)

2. CHECK (verify that the IUT does not transmit an I-Am-Request-PDU)

10.8.7.5 Who-Has Specifying Unknown Device Ids

Reason for Change: No test exists for this functionality.

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

1. TRANSMIT,
   DA = IUT,
   SA = TD,
   DNET = 1,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)
   'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)
   'Object Identifier' = (Device object identifier of VD1B)
2. CHECK (verify that the IUT does not transmit an I-Have-Request-PDU)

12. DATA LINK LAYER PROTOCOLS TESTS

12.1 MS/TP State Machine Tests

12.1.3 MS/TP Data Link Layer Tests (Alternate)

12.1.3.3 Verify T_{frame_gap}

Reason for Change: Added 'Configruation Requirements'.

Purpose: Verify that the maximum idle time between data octets when transmitting a frame is 20 bit times or less.

Configuration Requirements: Run the IUT and a Reference Master (or Router) on the same MS/TP network.

Test Steps:

1. Elicit the transmission of any frame from the IUT.
2. Measure the longest EIA-485 idle time that appears between octets within the data frame transmitted by the IUT. If there is no idle time between octets, pass the IUT.
3. Fail the IUT if the time measured in step 2 is greater than the time intervals shown below for each baud rate.

9600 baud:	fail if interval is greater than 2,083 microseconds
19200 baud:	fail if interval is greater than 1,042 microseconds
38400 baud:	fail if interval is greater than 521 microseconds
76800 baud:	fail if interval is greater than 261 microseconds
115200 baud:	fail if interval is greater than 173 microseconds
x baud:	fail if interval is greater than (20/x) seconds

13. SPECIAL FUNCTIONALITY TESTS

13.1 Segmentation

13.1.12.1 IUT Does Not Support Segmented Response

Reason for change: Adding 'Server' flag, in consequence of BTL-CRR-0177_server_in_Abort-PDU.doc

Purpose: To verify that the IUT returns the correct abort message when it does not support segmented responses and a response would be larger than 1 segment.

BACnet Reference Clause: 5.4.5.3.

Test Concept: The TD uses ReadPropertyMultiple to ask for more data than can fit in a single segment. The TD also specifies that the smallest (50 octet) segment size be used for the response. The data that are requested is the Object_Identifier property of the Device object of the IUT. The number of copies of the data that is requested is one more than the maximum number which would fit in a 50-octet segment.

Configuration Requirements: The IUT supports execution of the ReadPropertyMultiple service, but does not support transmission of segmented responses.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'max-APDU-length-accepted' = B'0000',
   'segmented-response-accepted' = TRUE,
   'Object Identifier' = (Device, X),
   'Property Identifier' = Object_Identifier,
   'Property Identifier' = Object_Identifier,
   'Property Identifier' = Object_Identifier,
   'Property Identifier' = Object_Identifier,
   'Property Identifier' = Object_Identifier
2. RECEIVE BACnet-Abort-PDU,
   'Server' = TRUE,
   'Abort Reason' = SEGMENTATION_NOT_SUPPORTED

13.8 Backup and Restore Procedure Tests

13.8.1 Backup and Restore Execution Tests

13.8.1.1 Execution of Full Backup and Restore Procedure

Reason For Change: Corrected the Backup_And_Restore_State in step 22.

Purpose: This test case verifies that the IUT can execute a full Backup and Restore procedure.

Test Concept: This test takes the IUT through a successful Backup and then a successful Restore procedure. The Database_Revision and Last_Restore_Time properties are noted before the procedure begins for later comparison. The IUT is then commanded to enter the Backup state; all the files are read, and the IUT is commanded to end the backup. If the Database_Revision property can be changed by means other than the restore procedure, it is modified and checked to ensure that it incremented correctly; then the IUT is commanded to enter the Restore state. If the file objects do not exist on the IUT, the TD will create them in the IUT. The files are then truncated to size 0, the file contents are written to the

IUT, and the IUT is commanded to end the restore. The Database_Revision and Last_Restore_Time properties are checked to ensure that they incremented or advanced correctly.

For IUTs that use Stream Access when performing the AtomicReadFile and AtomicWriteFile services, a Maximum Requested Octet Count (MROC) and a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max_APDU_Length_Accepted and the IUT's maximum transmittable APDU length. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max_APDU_Length_Accepted.

Test Steps:

```
1.  READ DR1 = Database_Revision
2.  READ LRT1 = Last_Restore_Time
3.  READ OL1 = Object_List
4.  REPEAT X = (1 through length of OL1) DO {
        READ NAMES[X] = (OL1[X]), Object_Name
    }
5   IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        READ BPT = Backup_Preparation_Time
        READ RPT = Restore_Preparation_Time
        READ RCT = Restore_Completion_Time
        VERIFY Backup_And_Restore_State = IDLE
6.  TRANSMIT ReinitializeDevice-Request,
        'Reinitialized State of Device' =      STARTBACKUP,
        'Password' = (any valid password)
7.  RECEIVE BACnet-Simple-ACK-PDU
8.  IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        WAIT BPT
        READ BRSTATE = Backup_And_Restore_State
        READ CF = Configuration_Files
        WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
            IF CF is an empty list THEN
                READ CF = Configuration_Files
            IF CF is a non-empty list THEN
                READ X = (the file referenced by Configuration_Files[1]).Name
        }
        CHECK (BRSTATE = PERFORMING_A_BACKUP)
9.  READ CF = Configuration_Files
10. CHECK (CF is a non-empty array of BACnetObjectIdentifiers referring to File objects)
11. REPEAT X = (each entry in CF) DO {
        READ Y = X, File_Access_Method
        IF (Y = RECORD_ACCESS)
            WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
                    TRANSMIT AtomicReadFile-Request,
                            'Object Identifier' = X,
                            'File Start Record' = (the next unread record),
                            'Requested Record Count' = 1
                    RECEIVE AtomicReadFile-ACK,
                            'End Of File' = TRUE | FALSE,
                            'File Start Record' = Z,
                            'Requested Record Count' = 1
                            'Returned Data' = (File contents)
                    | Error-PDU -- only acceptable for the first record and only when there are no records in the file
                            'Error Class' =          SERVICES,
```

```
                    'Error Code' =          INVALID_FILE_START_POSITION
            }
    ELSE
        WHILE (the last read did not indicate 'End Of File') DO {
                TRANSMIT AtomicReadFile-Request,
                    'Object Identifier' = X,
                    'File Start Position' = (the next unread octet),
                    'Requested Octet Count' = MROC
                RECEIVE AtomicReadFile-ACK,
                    'End Of File' = TRUE | FALSE,
                    'File Start Position' = (the next unread octet)
                    'File Data' = (File contents of length MROC if 'End Of File' is FALSE
                            or of length MROC or less if 'End Of File' is TRUE)
                | Error-PDU -- only acceptable for the first record and only when there are no records in the file
                    'Error Class' =         SERVICES,
                    'Error Code' =          INVALID_FILE_START_POSITION
            }
    }
12. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = ENDBACKUP,
        'Password' = (any valid password)
13. RECEIVE BACnet-Simple-ACK-PDU
14. VERIFY System_Status ! = BACKUP_IN_PROGRESS
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        VERIFY Backup_And_Restore_State = IDLE
16. IF (Database_Revision is changeable) THEN
        MAKE (the configuration in the IUT different, such that the Database_Revision property increments)
        VERIFY Database_Revision <> DR1
        READ DR2 = Database_Revision
        CHECK (DR1 <> DR2)
17. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = STARTRESTORE,
        'Password' = (any valid password)
18. RECEIVE BACnet-Simple-ACK-PDU
19. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        WAIT RPT
        READ BRSTATE = Backup_And_Restore_State
        WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
        }
        CHECK (BRSTATE = PERFORMING_A_RESTORE)
20. READ OL2 = Object_List
21. REPEAT X = (entry in CF) DO {
        IF (X is not in OL2)
            TRANSMIT CreateObject-Request
                    'Object Identifier' = X
            RECEIVE CreateObject-ACK
                    'Object Identifier' = X
        READ FS = X, File_Size
        IF (File_Size is not equal to the size of the backed up file)
            WRITE X, File_Size = 0
        IF (Y = RECORD_ACCESS)
            TRANSMIT AtomicWriteFile-Request
                    'File Identifier' = X
                    'File Start Record' = 0
```

```
                ‘Record Data’ = (file content for first record obtained in step 11)
            RECEIVE AtomicWriteFile-ACK
                ‘File Start Record’ = 0
            REPEAT REC = (each record in the backup of this file) {
                TRANSMIT AtomicWriteFile-Request
                    ‘File Identifier’ = X
                    ‘File Start Record’ = -1
                    ‘Record Count’ = 1
                    ‘Record Data’ = REC
                RECEIVE AtomicWriteFile-ACK
                    ‘File Start Record’ = (the record number)
            }
        ELSE
            REPEAT Z = (0 through the file size, in increments of MWDL) DO {
                TRANSMIT AtomicWriteFile-Request
                    ‘File Identifier’ = X
                    ‘File Start Position’ = Z
                    ‘Record Data’ = (file contents obtained from the backup, the number of octets
                        being the lesser of (file size - Z) and MWDL)
                RECEIVE AtomicWriteFile-ACK
                    ‘File Start Position’ = Z
            }
    }
22. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        VERIFY Backup_And_Restore_State = ~~RESTORE_IN_PROGRESS~~ PERFORMING_A_RESTORE
23. TRANSMIT ReinitializeDevice-Request,
            ‘Reinitialize State Of Device’ = ENDRESTORE,
            ‘Password’ = (any valid password)
24. RECEIVE BACnet-Simple-ACK-PDU
25. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        WAIT RCT
        VERIFY Backup_And_Restore_State = IDLE
26. READ DR3 = Database_Revision
27. CHECK (DR3 <> DR1)
28. IF (Database_Revision was changed in step 16) THEN
        CHECK (DR3 <> DR2)
29. VERIFY Last_Restore_Time > LRT1
30. READ OL3 = Object_List
31. CHECK (that OL1 and OL3 contain the same set of objects)
32. REPEAT X = (1 through length of OL1) DO {
        VERIFY (OL1[X]), Object_Name = NAMES[X]
    }
```

13.8.1.6 Ending Backup and Restore Procedures via Timeout

Reason For Change: Modified how the test WAITs for Protocol_Revision < 10.

Purpose: This test case verifies that the IUT will end Backup and Restore procedures after not receiving any messages related to the backup or restore for longer than Backup_Failure_Timeout and that the Backup_Failure_Timeout property is writeable.

Test Steps:

1. WRITE Backup_Failure_Timeout = (A value T1 greater than Backup_Preparation_Timeout)

2. VERIFY Backup_Failure_Timeout = T1
3. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    READ BPT = Backup_Preparation_Time
4. TRANSMIT ReinitializeDevice-Request,
        'Reinitialized State of Device' = STARTBACKUP,
        'Property Identifier' = (any valid password)
5. RECEIVE Simple-ACK-PDU
6. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT BPT
    READ BRSTATE = Backup_And_Restore_State
    WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
        WAIT 1 second
        READ BRSTATE = Backup_And_Restore_State
    }
    CHECK (BRSTATE = PERFORMING_A_BACKUP)
7. WAIT (T1 + 10 seconds)
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    VERIFY Backup_And_Restore_State = IDLE
9. VERIFY System_Status ! = BACKUP_IN_PROGRESS
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    READ RPT = Restore_Preparation_Time
    READ RCT = Restore_Completion_Time
11. TRANSMIT ReinitializeDevice-Request,
        ‘Reinitialized State of Device’ = STARTRESTORE,
        ‘Password’ = (any valid password)
12. RECEIVE BACnet-Simple ACK-PDU
13. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    READ BRSTATE = Backup_And_Restore_State
    WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
        WAIT 1 second
        READ BRSTATE = Backup_And_Restore_State
    }
    CHECK (BRSTATE = PERFORMING_A_RESTORE)
ELSE
    WAIT (30 seconds)
14. WAIT (*T1 + 10* ~~40~~ seconds)
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    VERIFY Backup_And_Restore_State = IDLE
16. VERIFY System_Status ! = DOWNLOAD_IN_PROGRESS

Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.

13.8.1.8 Attempting a Backup Procedure with an Invalid Password

Reason for Change: Added error codes supported per Addendum 12.0g-5.
Purpose: To verify the correct execution of the Backup procedure when an invalid password is provided *and when a password is required but no password is provided*. If the IUT cannot be made to deny a ReinitializeDevice <STARTBACKUP> service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,

'Reinitialized State of Device' = STARTBACKUP,
'Password' = (any invalid password)

2. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
   RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE
   ELSE
   (RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE) |
   (RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = SERVICE_REQUEST_DENIED) |
3. *TRANSMIT ReinitializeDevice-Request,*
   'Reinitialized State of Device' = STARTBACKUP
4. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
   RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE
   ELSE
   (RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE) |
   (RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = SERVICE_REQUEST_DENIED) |

13.8.1.9 Attempting a Restore Procedure with an Invalid Password

Reason for Change: Added error codes supported per Addendum 12.0g-5.

Purpose: To verify the correct execution of the Restore procedure when an invalid password is provided *and when a password is required but no password is provided*. If the IUT cannot be made to deny a ReinitializeDevice <STARTRESTORE-> service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Password' = (any invalid password)
2. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
   RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE
   ELSE
   (RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE) |
   (RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = SERVICE_REQUEST_DENIED) |
3. *TRANSMIT ReinitializeDevice-Request,*
   'Reinitialized State of Device' = STARTRESTORE
4. *IF (Protocol_Revision is present and Protocol_Revision >= 7) THEN*
   RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,

Error Code = PASSWORD_FAILURE
ELSE
(RECEIVE BACnet-Error-PDU,
Error Class = SECURITY,
Error Code = PASSWORD_FAILURE) |
(RECEIVE BACnet-Error-PDU,
Error Class = SERVICES,
Error Code = SERVICE_REQUEST_DENIED)

13.8.1.11 Starting and Ending a Restore Procedure when a Password is not Required

Reason For Change: Corrected the 'Reinitialized State of Device' value in step 5.

Purpose: This test case verifies that the IUT ignores the password. If the IUT cannot be made to accept a ReinitializeDevice service request that contains any or no password, then this test shall be omitted.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ RPT = Restore_Preparation_Time
   READ RCT = Restore_Completion_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ~~END~~*ABORT*RESTORE,
   'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RCT
8. VERIFY System_Status ! = DOWNLOAD_IN_PROGRESS

13.8.2 Backup and Restore Initiation Tests

13.8.2.1 Initiate a Full Backup and Restore

Reason For Change: Added note about preparation time properties. Clarified test characteristics for backup file names.

Purpose: To verify that the IUT can perform a Backup and Restore on a BACnet server device.

Test Concept: The IUT is first made to initiate a Backup and then a Restore of the TD device. This test verifies that the IUT performs the Backup procedure correctly by comparing the resulting restored file with the original. The TD is made to respond appropriately such that the Backup and Restore procedures are completed normally. The final check can be accomplished using a file compare of the original files to the files restored or by comparing the network traffic during the backup to the network traffic during the restore. The number of files, the order of the files, and the file content should be the same. The test is to be executed multiple times with the TD configured with different sets of backup and restore characteristics.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. The TD is configured with some of the following characteristics:

Backup Characteristics:

1. The TD is configured to contain an APDU size that is smaller than the APDU size of the IUT. If the TD and the IUT support segmentation, the TD is configured to support a smaller window size than the IUT.
2. The TD is configured to contain a configuration file of size zero.
3. The TD is configured to contain some configuration files that are STREAM_ACCESS and some that are RECORD_ACCESS.
4. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
5. The TD is configured to require the same password for all of the reinitialize device requests.
6. The TD is configured to contain *characters in the object name of some* file ~~name~~ *objects, such as * " and *, that would *reveal weakness in the implementation process that assigns names to files where the backup is stored* ~~not be accepted~~ by the OS which the IUT is running on.
7. The TD is configured with a Protocol_Revision < 10.
8. The TD is configured with a Protocol_Revision ~~≥~~ ≥ 10. This is only used if the IUT claims Protocol_Revision ~~≥~~ ≥ 10.

Note that if IUT claims Protocol_Revision < 10, the presence of preparation time properties in a TD with Protocol_Revision ≥ 10 may be ignored and cannot be relied upon.

Restore Characteristics:

1. The TD is configured to support CreateObject service, and some of the configuration files exist while others do not.
2. The TD is configured such that some of the configuration file File objects exist, but the file size is different from that of the file to be restored.
3. The TD is configured to not support the CreateObject service.
4. The TD is configured to contain some configuration files that are STREAM_ACCESS and some that are RECORD_ACCESS.
5. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
6. The TD is configured to require the same password for all of the reinitialize device requests.
7. The TD is configured with a Protocol_Revision < 10.
8. The TD is configured with a Protocol_Revision ~~≥~~ ≥ 10. This is only used if the IUT claims Protocol_Revision ~~≥~~ ≥ 10.

Note that if IUT claims Protocol_Revision < 10, the presence of preparation time properties in a TD with Protocol_Revision ≥ 10 may be ignored and cannot be relied upon.

Test Steps:

1. MAKE (IUT initiate a backup on the TD device)
2. WAIT (for backup to complete)
3. MAKE (changes required in TD to meet restore characteristics for this test)
4. MAKE (IUT initiate a restore on the TD device)
5. WAIT (for restore to complete)
6. CHECK (that the file content restored is the same as the file content that was backed up)

Notes to Tester: Other items to ensure were correct during execution of the test:

1. Verify the order the IUT read the configuration files was the same as the order returned by the Configuration_Files property.
2. Verify that any file with a File_Size of zero was restored.
3. Verify that each file read is in byte order if STREAM_ACCESS and in record order if RECORD_ACCESS.

13.X12.1 Reading with maximum-segments-accepted bit pattern B'000'

Reason for Change: There is no SSPC proposal for this change.

Purpose: To verify that the IUT implements at least support for two segments, when the 'max-segments-accepted' parameter that it sends is B’000’.

Configuration Requirements: If the IUT cannot be configured to issue any BACnet-Confirmed-Request-PDU with 'segmented-response-accepted' = TRUE and the 'max-segments-accepted' parameter equal to B’000’, then this test shall be skipped.

1. RECEIVE BACnet-Confirmed-Request-PDU,
   'segmented-response-accepted' = TRUE
   'max-segments-accepted' = B‘000’
2. TRANSMIT BACnet-ComplexACK-PDU,
   'segmented-message' = TRUE,
   'more-follows' = TRUE,
   'sequence-number' = 0,
   'proposed-window-size' = (any valid value)
3. RECEIVE BACnet-SegmentACK-PDU,
   'server' = FALSE,
   'negativeACK' = FALSE
4. TRANSMIT BACnet-ComplexACK-PDU,
   'segmented-message' = TRUE,
   'more-follows' = FALSE,
   'sequence-number' = 1
5. RECEIVE BACnet-SegmentACK-PDU,
   'server' = FALSE,
   'negativeACK' = FALSE

14. BACnet/IP FUNCTIONALITY TESTS

14.1 Non-BBMD B/IP Device

14.1.7 Forwarded-NPDU (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Purpose: To verify that an IUT, not configured as a BBMD, will process a Forwarded-NPDU message.

Configuration Requirements: The IUT shall not be configured as a BBMD. The TD shall be on a different IP subnet than that of the IUT.

Test Steps:

1. TRANSMIT DA = Directed IP Broadcast to IUT’s IP Subnet, SA = TD,
   Forwarded-NPDU,
   Originating-Device = TD,
   NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE DA = TD, ~~SA = IUT,~~
   Original-Unicast-NPDU,

NPDU = I-Am
ELSE
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

14.1.8 Original-Broadcast-NPDU

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Purpose: To verify that an IUT, not configured as a BBMD, will process an Original-Broadcast-NPDU message.

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, SA = TD,
Original-Broadcast-NPDU,
NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*
RECEIVE DA = TD, ~~*SA = IUT,*~~
Original-Unicast-NPDU,
NPDU = I-Am
ELSE
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

14.1.10 Forwarded-NPDU (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Purpose: To verify that an IUT, not configured as a BBMD, will process a Forwarded-NPDU message.

Configuration Requirements: The IUT should not be configured as a BBMD. The TD shall be on the same subnet as the IUT. D1 is a device on a different IP subnet than the TD.

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, SA = TD,
Forwarded-NPDU,
Originating-Device = D1,
NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*
RECEIVE DA = D1, ~~*SA = IUT,*~~
Original-Unicast-NPDU,
NPDU = I-Am
ELSE
RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
3. CHECK (The IUT shall not issue any Forwarded-NPDUs)

14.2 BBMD B/IP Device with a Server Application

14.2.1 Execute Forwarded-NPDU

14.2.1.1 Execute Forwarded-NPDU (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask

Test Steps:

1. TRANSMIT
   DA = Directed IP Broadcast to IP Subnet 1,
   SA = BBMD1,
   Forwarded-NPDU,
   Originating-Device = BBMD1,
   NPDU = Who-Is
2. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE D~~A~~ESTINATION = BBMD1,
   Original-Unicast-NPDU,
   NPDU = I-Am
   ELSE
   *(*RECEIVE
   DA = Local IP Broadcast on IP Subnet 1,
   ~~SA = IUT,~~
   Original-Broadcast-NPDU,
   NPDU = I-Am
~~3.~~ RECEIVE
   DA = Directed IP Broadcast to IP Subnet 2,
   ~~SA = IUT~~
   Forwarded-NPDU,
   Originating-Device = IUT,
   NPDU = I-Am*)*
~~4~~*3*. CHECK (The IUT does not forward or resend the Who-Is packet out the port on which it was received)

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.2.1.2 Execute Forwarded-NPDU (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255

Test Steps:

1. TRANSMIT
~~DA = IUT,~~
S*OURCE*~~A~~ = BBMD1,
Forwarded-NPDU,
Originating-Device = BBMD1,
NPDU = Who-Is
2. RECEIVE
DA = Local IP Broadcast on IP Subnet 1,
~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = BBMD1,
NPDU = Who-Is
3. IF (the IUT responds with Unicast I-Am) THEN
RECEIVE D~~A~~*ESTINATION = BBMD1,*
Original-Unicast-NPDU,
NPDU = I-Am
ELSE
*(*RECEIVE
DA = Local IP Broadcast on IP Subnet 1,
~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
~~4.~~ RECEIVE
DA = BBMD1,
~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.2.2 Execute Original-Broadcast-NPDU

14.2.2.1 Execute Original-Broadcast-NPDU (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask

Test Steps:

1. TRANSMIT
DA = Local IP Broadcast,
SA = D1,
Original-Broadcast-NPDU,
NPDU = Who-Is
2. RECEIVE

DA = Directed IP Broadcast to IP Subnet 2,
~~SA = IUT~~
Forwarded-NPDU,
Originating-Device = D1,
NPDU = Who-Is

3. RECEIVE
DA = Local IP Broadcast,
~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am

4. IF (the IUT responds with Unicast I-Am) THEN
RECEIVE D~~A~~ESTINATION = D1,
Original-Unicast-NPDU,
NPDU = I-Am

ELSE
RECEIVE
DA = Directed IP Broadcast to IP Subnet 2,
~~SA = IUT~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.2.2.2 Execute Original-Broadcast-NPDU (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The IUT shall be configured with a BDT that contains:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255

Test Steps:

1. TRANSMIT
DA = Local IP Broadcast,
SA = D1,
Original-Broadcast-NPDU,
NPDU = Who-Is

2. RECEIVE
DA = BBMD1,
~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = D1,
NPDU = Who-Is

3. IF (the IUT responds with Unicast I-Am) THEN
RECEIVE D~~A~~ESTINATION = D1,
Original-Unicast-NPDU,
NPDU = I-Am

ELSE
RECEIVE
DA = Local IP Broadcast,

~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am

4. RECEIVE
DA=BBMD1,
~~SA=IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.3.3 Verify Broadcast Distribution Table Created from the Configuration Saved During the Previous Session

Reason for Change: Revised test to allow testing when BDT can be configured with local configuration tool only.

Purpose: To verify that a BBMD will update the BDT in the local configuration database and initialize it at startup.

Configuration Requirements: The IUT's BDT ~~does not consist of the same entries as are~~ *can either be* written in step 1, *or configured with a local configuration tool*.

Test Steps:

1. IF (The IUT's BDT can be written with Write-Broadcast-Distribution-Table)
TRANSMIT
DA = IUT,
SA = D1,
Write-Broadcast-Distribution-Table,
(List of BDT entries ~~consisting of three entries~~ *at least one of which is different from what it has*
~~IUT 255.255.255.255~~
~~BBMD1 255.255.255.255~~
~~BBMD2 255.255.255.255~~
)
RECEIVE
DA = D1,
SA = IUT,
BVLC-Result,
'Result Code' = Successful completion
ELSE
MAKE (the IUT's BDT different, so that values in the BDT at step 6 can be distinguished)
~~3~~*2*. WAIT (-Vendor specified period for BDT to be saved in non-volatile memory)
~~4~~*3*. MAKE (the IUT reset)
~~5~~*4*. TRANSMIT
DA = IUT,
SA = D1,
Read-Broadcast-Distribution-Table
~~6~~*5*. RECEIVE
DA = D1,
SA = IUT,
Read-Broadcast-Distribution-Table-Ack,
List of BDT Entries
~~7~~*6*. CHECK (*IUT's BDT holds the entries with which it was configured*~~List of BDT Entries consisting of three entries (order unspecified)~~
~~IUT 255.255.255.255~~
~~BBMD1 255.255.255.255~~

~~BBMD2 255.255.255.255~~
)

14.7 Broadcast management (BBMD, Foreign Devices, Local Application)

14.7.1 Broadcast Message from Directly Connected IP Subnet

14.7.1.1 Broadcast Message from Directly Connected IP Subnet (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The IUT's BDT shall contain the following three entries:

B/IP Address	Broadcast Distribution Mask
IUT	IP Subnet 1 subnet mask
BBMD1	IP Subnet 2 subnet mask
BBMD2	IP Subnet 3 subnet mask

The TD shall be on the same subnet as the IUT. D1 is a device on a different IP subnet than the TD. Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 6 is* ~~steps 6-10 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT
   DA = Local IP Broadcast,
   SA = D1,
   Original-Broadcast-NPDU,
   NPDU = Who-Is
2. RECEIVE
   DA = Directed IP Broadcast to IP Subnet 2,
   ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
3. RECEIVE
   DA = Directed IP Broadcast to IP Subnet 3,
   ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
4. RECEIVE
   DA = FD1,
   ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
5. RECEIVE
   DA = FD2,
   ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
6. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE DA = D1,~~ SA = IUT~~,
   Original-Unicast-NPDU,

NPDU = I-Am
ELSE
*(*RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
RECEIVE DA = Directed IP Broadcast to IP Subnet 2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = Directed IP Broadcast to IP Subnet 3, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.7.1.2 Broadcast Message from Directly Connected IP Subnet (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The BDT shall contain the following three entries:

B/IP Address	Broadcast Distribution Mask
IUT	255.255.255.255
BBMD1	255.255.255.255
BBMD2	255.255.255.255

Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 6 is* ~~steps 6-10 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT DA = Local IP Broadcast, SA = D1,
   Original-Broadcast-NPDU,
   NPDU = Who-Is
2. RECEIVE DA = BBMD1, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
3. RECEIVE DA = BBMD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
4. RECEIVE DA = FD1, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is

5. RECEIVE DA = FD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D1,
   NPDU = Who-Is
6. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE DA = D1, ~~SA = IUT,~~
   Original-Unicast-NPDU,
   NPDU = I-Am

ELSE
*(*RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
RECEIVE DA = BBMD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = BBMD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.7.2 Broadcast Message Forwarded by a Peer BBMD

14.7.2.1 Broadcast Message Forwarded by a Peer BBMD (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The BDT shall be configured as in test 14.7.1.1.

Steps 2-3 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 4 is* ~~steps 4-8 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT DA = Directed IP Broadcast to IP Subnet 1, SA = BBMD1,
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
2. RECEIVE DA = FD1, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
3. RECEIVE DA = FD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is

4. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE D~~A~~ESTINATION = D2,~~ SA = IUT,~~
   Original-Unicast-NPDU,
   NPDU = I-Am
   ELSE
   *(*RECEIVE DA = Local IP Broadcast,~~ SA = IUT,~~
   Original-Broadcast-NPDU,
   NPDU = I-Am
   RECEIVE DA = Directed IP Broadcast to IP Subnet 2,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = IUT,
   NPDU = I-Am
   RECEIVE DA = Directed IP Broadcast to IP Subnet 3,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = IUT,
   NPDU = I-Am
   RECEIVE DA = FD1,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = IUT,
   NPDU = I-Am
   RECEIVE DA = FD2,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = IUT,
   NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.7.2.2 Broadcast Message Forwarded by a Peer BBMD (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The BDT shall be configured as in test 14.7.1.2

Steps 2-4 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 5 is* ~~steps 5-9 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT ~~DA = IUT,~~ S*OURCE~~A~~* = BBMD1,
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
2. RECEIVE DA = Local IP Broadcast,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
3. RECEIVE DA = FD1,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
4. RECEIVE DA = FD2,~~ SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = D2,
   NPDU = Who-Is
5. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE D~~A~~ESTINATION = D2,~~ SA = IUT,~~

Original-Unicast-NPDU,
NPDU = I-Am
ELSE
*(*RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
RECEIVE DA = BBMD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = BBMD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.7.3 Broadcast Message from a Foreign Device

14.7.3.1 Broadcast Message From a Foreign Device (One-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The BDT shall be configured as in test 14.7.1.1.

Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 6 is* ~~steps 6-10 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT ~~DA = IUT,~~ SA = FD1,
   Distribute-Broadcast-To-Network,
   NPDU = Who-Is
2. RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
3. RECEIVE DA = BBMD1, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
4. RECEIVE DA = BBMD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
5. RECEIVE DA = FD2, ~~SA = IUT,~~
   Forwarded-NPDU,

Originating-Device = FD1,
NPDU = Who-Is

6. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE DA = FD1, ~~SA = IUT,~~
   Original-Unicast-NPDU,
   NPDU = I-Am

ELSE
*(*RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
RECEIVE DA = Directed IP Broadcast to IP Subnet 2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = Directed IP Broadcast to IP Subnet 3, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)*

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

14.7.3.2 Broadcast Message From a Foreign Device (Two-hop Distribution)

Reason For Change: This makes changes to all tests in sections 14.1 and 14.2 and 14.7 that use the I-Am service, to allow the unicast form of the response.

Configuration Requirements: The BDT and FDT shall be configured as in test 14.7.1.2.

Steps 2-5 are the distribution of the Who-Is request to the devices considered to be members of the BACnet network, *step 6 is* ~~steps 6-10 are~~ the distribution of the I-Am response from the local application.

Test Steps:

1. TRANSMIT ~~DA = IUT,~~ SA = FD1,
   Distribute-Broadcast-To-Network,
   NPDU = Who-Is
2. RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
3. RECEIVE DA = BBMD1, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
4. RECEIVE DA = BBMD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is

5. RECEIVE DA = FD2, ~~SA = IUT,~~
   Forwarded-NPDU,
   Originating-Device = FD1,
   NPDU = Who-Is
6. *IF (the IUT responds with Unicast I-Am) THEN*
   RECEIVE DA = FD1, ~~SA = IUT,~~
   Original-Unicast-NPDU,
   NPDU = I-Am

ELSE
*(*RECEIVE DA = Local IP Broadcast, ~~SA = IUT,~~
Original-Broadcast-NPDU,
NPDU = I-Am
RECEIVE DA = BBMD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = BBMD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD1, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am
RECEIVE DA = FD2, ~~SA = IUT,~~
Forwarded-NPDU,
Originating-Device = IUT,
NPDU = I-Am*)

Notes to Tester: The order of the messages transmitted by the IUT is not significant.

Version	Date	Author	Change
0.07	5-Aug-2004	Carl Neilson	Updates based on Nashville meeting comments on Round 3 updates.
0.08	24-Aug-2004	Carl Neilson	• Removed 9.24.4.X1, 9.24.4.X2. Now exist in 135.1. • Modified the purpose of 14.5.3. • Modified the purpose of 14.2.2 • Added 10.2.4.4
0.09		Roland Laird	• Modified all Clause 14 tests
0.10	26-Oct-2004	Roland Laird	• Continuation of BACnet/IP modifications - changes highlighted inline
0.11	27-Oct-2004	Carl Neilson	• Added 7.3.1.11, 9.1.1.1, 9.1.1.4, 9.1.2.1, 9.1.2.5. The specification of the expected Time Stamp in the ack notifications was changed. - changes still highlighted inline
0.12	29-Oct-2004	Carl Neilson	• Changes to 7.3.1.11, 9.1.1.1, 9.1.1.4, 9.1.2.1, 9.1.2.5 based on group feedback. • Changes to BACnet/IP based on group feedback
0.13	23-Nov-2004	Carl Neilson	• Added 9.24.1.X2 and 9.24.1.X3 • Added "Reason For Change" to all tests. • Added passing result text to 9.1.2.5 (missed in version 12) • A few minor typos
0.14	20-Dec-2004	Carl Neilson	• Added tests 10.2.2.3, 10.2.2.7.2, 10.2.2.7.3, 10.2.3.2, 10.2.3.5, 10.2.4.6, 10.2.4.8, 10.2.6 from P3-Routing-14. • Modified 10.2.4.4 as per P3-Routing-14. • Added tests 9.20.2.1 into RPM-B • Added 7.3.2.9.8, 7.3.2.9.9, 7.3.2.17.5, 7.3.2.18.6, 7.3.2.19.5, 7.3.2.22.9 into WP-B. • Added 9.23.1.X7, 9.23.1.X8 into WPM-B • Added 7.3.1.X2
0.15	15-Jun-2005	Carl Neilson, Roland Laird	• Modified 7.3.2.9.8's & 7.3.2.9.9's reason for change comments as 135.1a now incorporates the complete change. • Added 8.34.X1 • 13.X.1 fixed test step reference in steps 8 & 18 • 13.4.3 changed 2 < x < 254 to 2 < x <= 254 • Change 9.20.2.1 to 9.20.2.X1 as the test is new and there is already a 9.20.2.1 in 135.1 • Added scheduling tests 7.3.2.22.X2 and 7.3.2.22.X3 from ShedProtRev4Tests-9. Test numbers were changed to correspond with the equivalent pre-revision 4 tests.
0.16	19-Jul-2005	Carl Neilson	• Added WhoHas tests 9.32.1.X1, 9.32.1.X2
0.17	05-Oct-2005	Jim Butler Carl Neilson	• Added Recipient List Test 7.3.2.20.3.X1, • Added MS/TP restart tests 2.2.14...2.2.17 • Added RP fallback tests 8.20.Y1.X1, 8.20.Y1.X2 • Added AckAlarm tests 9.1.1.X1, 9.1.1.X2 • Changed 2.2.7 as per CLB-001 • Changed 2.2.6 as per CLB-002 • Changed 2.2.5 as per CLB-003 • Changed 2.2.4 as per CLB-004 • Added changes to 7.3.2.23.5
0.18	24-Oct-2005	Carl Neilson	• Added ARCNET tests & re-arranged section 2.

			• Added 7.3.1.X3 Array Sizing Test • Added 13.X2.1 APDU Retry and Timeout
0.19	27-Oct-2005	Carl Neilson	• Removed router qualification tests. • Added reason for change to 13.X2.1 & modified note to tester • Fixed incorrect numbering of BACnet/IP sections • Deleted old comment as end of 7.3.2.20.3.X1
0.20	17-Jan-2006	Carl Neilson	• Added 8.8.1 & 8.8.2 that include transmission of final BACnet-ComplexACK-PDUs
4.0.0	13-Sep-2006	Carl Neilson	• Changed revision numbering
4.0.1	04-Apr-2007	Carl Neilson	• Round 4 changes (excl SCHED)
4.0.2	02-May-2007	Carl Neilson	• Added 9.10.1.X2
4.0.3	11-Jun-2007	Lori Tribble	• Updated document per CRR-0005, CRR-0008, CRR-0009, CRR-0011, CRR-0014 • Updated document per CRR-0015, CRR-0017, CRR-0020 • Updated document per CRR-0021, CRR0022
4.0.4	23-Jul-2007	Lori Tribble	• Updated the Reason For Change. • Highlighted new items for Round 4 in Green • Highlighted items to be deleted in Yellow. Waiting on approval of round 3 documents before we delete. • Highlighted items with questions in Purple. Waiting on approval of round 3 documents before changing. • See BTL Specified Tests 3.1.4 change log for details.
4.0.5	10-Oct-2007	Lori Tribble	• Removed tests previously highlighted in yellow. These tests are now in 135.1. • Added changes to tests 7.3.2.22.X1.1,2,3,4 per CRR-0030. • Added changes to test 7.3.2.22.X2.3.12 per CRR-0035. • Added changes to tests per WSPLab suggestions.
4.0.6	25-Oct-2007	Lori Tribble	• Removed some of the highlighting. • Updated tests per mtg 10/15/2007
4.0.7	18-Dec-2007	Lori Tribble	• Added Virtual Routing tests • Added List Manipulation Tests
4.0.8	22-Feb-2008	Lori Tribble	• Fixed BTL-7.3.1.11 per TGTC-18.
4.0.9	01-Apr-2008	Lori Tribble	• Updated page header format.
4.0.10	16-Apr-2008	Lori Tribble	• Applied the following: TGTC-05 - Adds UTCTimeSync to all schedule tests. TGTC-08 - Adds 9.1.2.3 and 9.1.2.6 to this document. TGTC-13 - changes already existed in this document. TGTC-14 - Modified 7.3.2.23.9, added 7.3.2.23.10, modified 7.3.2.23.X2. TGTC-16 - Modified 7.3.1.13 TGTC-17 - 7.3.2.23.10 has already been modified by TGTC-14 TGTC-18 - Added 7.3.1.10.X1, TGTC-19 - Modified 9.21.1.4 TGTC-21 - Modified 7.3.2.22.X1.2 and 7.3.2.22.X1.4 TGTC-35 - Added 7.3.1.3. TGTC-36 - Added 7.3.1.10 TGTC-37 - Modified 7.3.1.11 TGTC-40 - Added 9.22.2.4 TGTC-41 - Modified 7.3.2.23.6.1

			TGTC-43 - Added 8.22.1, 8.22.2. Modified 8.22.X2. BTL-CRR-0018 - Modified 9.10.2.1 BTL-CRR-0050 - Modified 7.2.1.10 BTL-CRR-0051 - Modified 13.X1.3, 13.X1.6, 13.X1.7 • Updated reason for change on several tests. • Updated tests for Rev 5 and 6 Added 9.1.1.X3 Added 9.1.2.3, 9.1.2.4, 9.1.2.7 • Added 13.X5.1, 13.X5.2, 13.X5.3, 13.X5.5, 13.X5.6 for Backup and Restore Initiation testing.
4.0.11	April 16, 2008	Lori Tribble	• Accepted Changes made above • Updated table of contents • Added Reason for Change to tests that did not have it. • Marked test 9.10.2.1 for further review
4.0.13	May 21, 2008	Lori Tribble	• Added test correction for 9.33.2.3 per BTL-CRR-0055. • Added test corrections for 9.14.2.3 and 9.15.2.2 per WS-038-4. • Added test corrections for 8.4.1, 8.4.2, 8.4.3.1, 8.4.3.2, 8.4.4, 8.4.5, 8.4.6 per BTL-CRR-0017. • Corrected reason for change on several tests. • Removed : from test numbers
4.0.14	June 20, 2008	Lori Tribble	• Updated tests 7.3.2.22.X1 and 7.3.2.22.X2.3.1 to match recent changes made in TI-WG on SED-004 and SED-006. • Updated all tests which use the UTCTimeSynchronization service to indicate using a UTC date. • Updated non-router tests per CN-092-04 • Question about test 8.4.6 correction to be answered. See comment.
4.0.15	September 9, 2008	Lori Tribble	• Applied BTL-CRR-0056 Time Master changes • Applied BTL-CRR-0064 NonRouterNetworkCommands • Updated document to reference 135.1-2007 section numbers. • Added 7.3.2.8.1 and 7.3.2.8.3. These required updates for UTCTimeSynchronization. • Added 7.3.2.21.3.1 and 7.3.2.21.3.2. These required updates to include UTCTimeSynchronization. • Added 7.3.2.23.1 and 7.3.2.23.2. These required updates to include UTCTimeSynchronization. • Added 7.3.2.23.3.1 - 9 and 7.3.2.23.4 - 8. These tests required updates to include UTCTimeSynchronization.
4.0.16	September 17, 2008	Lori Tribble	• Accepted all changes made previously. • Made format changes.
4.0.18	Oct 21, 2008	Lori Tribble	• Updated Reason For Change for all tests that now have SSPC proposals. • Added client side schedule tests • Removed test 9.23.1.X • Changed COV test from 24 hour lifetime to 8 hour lifetime.
5.0.1	Oct 21, 2008	Lori Tribble	• Accepted all changes made above. Changed version to

			5.0.1 • Updated Reason For Change for tests that now have SSPC proposals. •
5.0.2	Feb 24, 2009	Lori Tribble	• Added place holder for new test BTL-8.22.X4 Writing Array properties as a Whole array. • Renumbered test steps for 7.3.2.21.3.2 • TGTC-57: Updated Configuration Requirements for test 7.3.2.23.X2.3.12 Revision 4 Lower Event Priority Change Test. • TGTC-58: Updated test 7.3.2.23.X2.3.10 Revision 4 Calendar Entry WeekNDay Odd-Numbered Month Test. • TGTC-59: Updated test 7.3.2.24.1 Log_Enable Test. • TGTC-60: Updated test 8.4.2 CHANGE_OF_STATE Tests • TGTC-79: Updated tests 8.2.1 through 8.2.8 to include BEFORE Notification Fail Time before each notification. • TGTC-80: added tests 9.10.1.1 through 9.10.1.3 to wait Notification Fail Time before each notification. • TGTC-81: Added test 7.2.2. • TGTC-84: Updated test 7.3.2.24.10 Notification_Threshold Test • TGTC-85: Updated test 8.4.7 BUFFER_READY Tests • BDS-001: Updated tests 7.3.2.23.5 Exception_Schedule Restoration Test and 7.3.2.23.6 Weekly_Schedule Restoration Test • BTL-CRR-0069: Updated test 10.X.1 Static Router Binding, 10.X.2 Router Binding via Application Layer Services, 10.X.3 Router Binding via Who-Is-Router-To-Network, and 10.X.4 Router Binding via Broadcast. • BTL-CRR-JN3: Updated test 2.2.7. • Added database_revision tests from 135.1-2007f.
5.0.4	27-Mar-2009	Lori Tribble	• Changed test 9.2.1.X8 to be 9.3.X9. The test doesn't exist yet but is supposed to be the unconfirmed version of the 9.2.1.X4 test which is also not written. • Added place holder for 8.4.X2 Extended Algorithm Tests (ConfirmedEventNotification) and 8.5.X3 Extended Algorithm Tests (UnconfirmedEventNotification). • Renumbered 9.23.2.X7 to 9.23.2.6 (as defined in 135.1-2007) • Renumbered 9.23.1.X8 to 9.23.2.7 (as defined in 135.1-2007) • Updated tests 9.14.2.3 and 9.23.2.6 per BTL-CRR-0072 • Updated tests 9.1.1.1 and 9.1.1.4 per TGTC-111 • Updated test 7.3.2.23.X2.3.9 per TGTC-127 • Updated test 7.3.2.21.3.X per TGTC-128 • Updated test 9.22.1.X2 per TGTC-133 • Added test 7.3.2.24.8 per BTL-CRR-0070 • Added Chapter 6 sections that are changed or new to

			135.1 and whose contents are being used within some of the tests (i.e. READ) (Described in CN-093) • Added section 7.2.1.3 to document to show proposed change to text. (FR-??)
5.0.5	6-Apr-2009	Lori Tribble	• Added reason for change to chapter 6 section and for test 7.2.2. • Modified text for 7.2.1.3 and added reason for change. • Added reason for change for the Record_Count test (7.3.2.24.8)
5.0.6	9-Apr-2009	Lori Tribble	• Added test 9.2.20.1 Reading a Single, Unsupported Property from a Single Object. Per CRR-0039. • Fixed spelling error in Configuration Requirements of Stop_When_Full TRUE Test (7.3.2.24.6.1). • Updated Create and Delete Tests per proposal provided to BTL-WG and approved on 4/9/2009. Tests modified are: 8.16.2, 8.16.3, 8.16.4, 9.16.1.1, 9.16.1.2, 9.16.1.3, 9.16.1.4, 9.16.2.1, 9.16.2.2, 9.16.2.3, 9.16.2.4, 9.16.2.5, 9.16.2.6, 9.17.1.1. Also removed test 9.16.1.X1 per this document.
5.0.7	8-Jun-2009	Lori Tribble	• Added to 9.20.2.1 that this change is included in CN-121. • Changed test 7.3.1.1 per BTL-CRR-0074 and DJH-001-3.
5.0.8	22-Jun-2009	Lori Tribble	• Removed test 9.23.1.7 Writing Maximum Multiple Properties test. • Updated test 7.3.1.11 to update the configuration requirements to include initial configuration of the ACK_Required property. • Test 7.3.2.23.X1.1 - updated configuration requirements • Updated tests 7.3.2.23.7, 7.3.2.23.8, 7.3.23.X2.8, 7.3.23.X2.7 step 1 to correctly reference D_t not D_1 • Updated test 7.3.1.10 configuration requirements. Changed 'read-only' to 'not configurable'. • Removed all highlighting • Updated TOC.
5.0.final	26-Jun-2009	Lori Tribble	• Accepted all changes per acceptance by BTL-WG 6/18/2008.
6.0.1	26-Jan-2011	Duffy O'Craven	• Corrected: 'inside' for 'outside' in step 4 of test 7.3.2.8.2, based upon BTL-CRR-0172_7.3.2.8.2_inside_outside.doc • Put section 7.3.2.10 in order, before 7.3.2.21 • Revised test 7.3.2.23.X2.4 Revision 4 Weekly_Schedule and Exception_Schedule Interaction Test , based upon KV-001-03_7.3.2.23.X2.4.doc • Adjusted heading on test instead of section, so that test 7.3.2.24.6.1 appears in Table of Contents. • Added tests 9.1.1.X4 and 9.1.1.X5 to ACK-B, based upon BTL Specified Tests-Add135-2004m-4-ReAckAlarms-3.doc • Removed test 9.1.2.6, as the correct version is now in

			135.1-2009, per BTL-CRR-0125_9.1.2.6.doc • Added test 9.21.1.X5 Reading Items with Negative Count and MOREITEMS • Derived tests from 135.1-2009 in DCC-A and RD-A, adding proper password treatment based upon BTL-CRR-0078 DeviceCommunicationControl_Password.doc • Revised tests 9.24.21, 9.24.2.2, 9.27.1.1 and 9.27.1.3, and added tests 9.24.2.X3, 9.27.2.X3 and 9.27.2.X4 in DCC-B and RD-B, based upon 135-2004m-8 r2 Clarify DeviceCommunicationControll and ReinitializeDevice interactions.doc
9.0.3	7-Apr-2011	Duffy O'Craven	• Incorporated BTL - 7.3-MO_V9.doc including new test 7.3.2.24.X7 • Derived BTL – 7.3.1.12 with modifications in consequence of BTL-CRR-0171_7.3.1.12_TO-NORMAL.doc • Incorporated changes to genericize tests for logging objects in BTL - 8.21-MO V8.doc and BTL - 9.21-MO V7.doc • Incorporated DO-016-08_Verify_Notification_Logging.doc as tests 7.3.2.26.X1, 7.3.2.26.X2, 7.3.2.26.X3, and 7.3.2.26.X4 • Incorporated 135-2004b-5 - Restart Parameters_v2.doc in test 8.3.X1
9.0.4	26-May-2011	Duffy O'Craven	• Incorporated BTL-CRR-0082_ReadOnlyTest.doc in test 7.2.2.1 • Incorporated BTL-CRR-0083_nonDocumentedProperty.doc in test 7.2.2.X2 • Added test 2.2.18 Verify Tno_token w/ Serial Analyzer in consequence of BTL-CRR-0085-NewMSTPTest.doc • Specified Protocol_Revision ≥ 7 in test 13.2 in consequence of BTL-CRR-0087_TimeMasterTest.doc • Added modified test 9.7.1.1, in consequence of BTL-CRR-0089_9.7.1.1.doc
9.0.5	31-May-2011	Duffy O'Craven	• Modified tests 8.2.2, 8.2.4, 8.2.6, and 8.2.8 in consequence of BTL-CRR-0095_changeable_Status_Flags.doc • Modified tests 8.4.4, 8.4.5, 8.4.6, and 8.4.7 in consequence of BTL-CRR-0096_object_referenced_by_EE.doc • Specified Protocol_Revision < 10, in consequence of BTL-CRR-0104_correcting_9.10.2.1.doc • Corrected Test Concept: from TD to IUT in 10.X.3 in consequence of BTL-CRR-0106_10.X.3_TestingHints.doc • Corrected Test Configuration: of test 7.3.2.24.4 in consequence of BTL-CRR-0116_Log_Interval_read-only.doc • Corrected expected result in test 9.14.2.2 in consequence of BTL-CRR-

			0117_9.14.2.2_First_Failed_Element.doc • Corrected the name of test 9.30.1.1, and added BTL Specified Tests versions of 9.30.1.2, 9.31.1.1 and 9.31.1.2 derived from 135.1 - 2007 as specified in BTL-CRR-0113_9.31.1.1_diverge_dissimilar_tests.doc
9.0.6	09-Jun-2011	Duffy O'Craven	• Added tests 7.3.2.29.X1 and 7.3.2.29.X2 for Structured View in consequence of Structured View Test Plan v6.doc
9.0.7	12-Jun-2011	Duffy O'Craven	• Revised test 7.2.2.X2 to restrict the test to standard object types, in consequence of BTL-CRR-0130_7.2.2.X2.doc and making it identical to the revision in 135.1-2009n-1 • Further revised test 7.2.2.X2, in consequence of BTL-CRR-0180_P_C_C.doc • Further revised test 7.2.2, in consequence of BTL-CRR-0178_allowed-values_REAL.doc • Modified test 10.X.5 to ensure that the packet actually reaches the IUT, and that the test uses an address which resembles the actual address of IUT, in consequence of BTL-CRR-0138_10.X.5_same_DADR.doc • Removed test 7.3.2.21.3.X, as the version in 135.1-2009g-6 replaces it, in consequence of BTL-CRR-0141_7.3.2.21.3.X_DDB_without_range.doc. • Removed tests 8.22.1 and 8.22.2, as 135.1-2009i-7 ratified the Notes to Tester: addition that had caused these revised tests to supercede the 135.1 - 2003 - 8.22.1 and 135.1 - 2003 - 8.22.2 versions. • Removed test 8.22.X1, as the version in 135.1-2009i-8 replaces it. • Added qualifying language in Notes to Tester: of tests 7.3.2.23.X1.3, 7.3.2.23.X1.4, and 7.3.2.23.X2.8 in consequence of BTL-CRR-0158_Sch_Object_Writes.doc • Revised test 10.X.2 in consequence of BTL-CRR-0149_non-BROADCAST.doc • Removed test 14.1.7 in consequence of BTL-CRR-0152_eliminating_14.1.7.doc • Added to the Configuration Requirements: of test 7.3.2.24.X3, in consequence of BTL-CRR-0151_7.3.2.24.X3.doc • Adds a Notes to Tester: to test 7.3.2.24.X1, in consequence of BTL-CRR-0160_Log-interrupted.doc • Further revised test 7.3.2.24.8 in consequence of BTL-CRR-0169_7.3.2.24.7_Not_all_at_once.doc • Derives with modifications, and adds a Notes to Tester: to create test BTL - 7.3.2.24.12, in consequence of BTL-CRR-0165_7.3.2.24.12.doc • Added 'Server' = TRUE, to test 7.1 and derived a BTL Specified Test 13.1.12.1 with that change, in consequence of BTL-CRR-0177_server_in_Abort-PDU.doc • Further revised test 9.1.2.3, and derived test BTL - 9.1.2.6 in consequence of BTL-CRR-

			0195_9.1.2.3_and_9.1.2.6.doc • Further revised tests 9.10.1.1 and 9.10.1.2, in consequence of BTL-CRR-0182_9.10.1.2.doc • Further revised test 9.10.1.1, and derived test BTL - 9.10.1.7, in consequence of BTL-CRR-0194_ACK_in_9.10.1.1_and_9.10.1.7.doc • Further derived 9.10.1.7, in consequence of BTL-CRR-0184-9.10.1.7.doc and BTL-CRR-0200-9.10.1.7.doc • Removed test 9.21.1.4 as 135.1-2009g-16 replaced it, in consequence of BTL-CRR-0201_9.21.1.4.doc
9.0.8	22-Jun-2011	Duffy O'Craven	• Referred from BTL 7.2.2.X2 to test 7.1.x and from BTL 7.2.2.1 to test 7.2.x in 135.1-2009i-22, which is the first occurrence of this test in a ratified addendum., but with slightly different content. • Added qualifying language in Notes to Tester: of tests7.3.2.23.8 in consequence of BTL-CRR-0158_Sch_Object_Writes.doc • Changed test numbers for tests 7.3.2.23.X2.1 through 7.3.2.23.X2.8 (now 7.3.2.23.X.1 through 7.3.2.23.X.8), 7.3.2.23.X2.3.1 through 7.3.2.23.X2.3.13 (now 7.3.2.23.X.3.1 through 7.3.2.23.X.3.13), and 7.3.2.23.X1 through 7.3.2.23.X4 (now 7.3.2.23.Y.1 through 7.3.2.23.Y.4) and 7.3.2.23.X3 (now 7.3.2.23.Y) to the 135.1-2009j-17 test number used for BUFFER_READY tests. • Changed test number for 8.5.X1 to the 8.5.7 used in 135.1-2009l. • Added more qualifying language in Notes to Tester: of tests 7.3.2.23.X1.3, 7.3.2.23.X1.4, and 7.3.2.23.8 incorporating from the versions in 135.1-2009g-17, 135.1-2009g-21 and 135.1-2009i-7 • Added mention of the version in 135.1-2009j-14, in test 7.3.2.24.4 The BTL Specified Test takes precedence. • Added references to the version in 135.1-2009i-14, in tests 7.3.2.24.X1, 7.3.2.24.X2, and 7.3.2.24.X3 The BTL Specified Tests take precedence. • Added references to the version in 135.1-2009i-14, in tests 7.3.2.24.X4, 7.3.2.24.X6, and 7.3.2.24.X7 The versions of these tests are identical with those in BTL Specified Tests. • Replaced test 7.3.2.24.5 with exactly the 135.1-2009g-16 version, adjusting only some capitalization typos, with no semantic difference. • Added mention of the version in 135.1-2009j-10, in tests 7.3.2.24.6.1, and 7.3.2.24.6.2 The versions of these tests are quite different. • Replaced tests 7.3.2.24.7, and 7.3.2.24.8 with exactly the 135.1-2009j-13 and 135.1-209j-14 PPR1_DRAFT versions, with no semantic difference. • Added references to the versions in 135.1-2009h-3, in tests 8.2.1 through 8.2.8 • Added reference to the version in 135.1-2009i-3, in test 8.2.x1 which is identical with the version in BTL

			Specified Tests. • Replaced tests 8.18.1, 8.18.2, 8.18.X1, and 8.18.X2 with exactly the 135.1-2009i-4 versions, which compared with the prior BTL Specified version means adjusting a syntactically incorrect VERIFY to CHECK, with no semantic difference. • Specified (BACnetDeviceObjectPropertyReference– referring to the buffer property of the log object) as the first part of Event_Values in every ConfirmedEventNotification of BUFFER_READY event type.
9.0.9	6-Jul-2011	Duffy O'Craven	• Removed tests 7.3.2.10.X3, 7.3.2.10.X4, 7.3.2.10.X5 as these are identical to the versions in 135.1-2009f-2. • Renumbered test 7.3.2.10.X6 to 7.3.2.10.X4 to match the , number of the corresponding test which is in 135.1-2009f-2, and which is identical, except for an errata, with the version in BTL Specified Tests.
9.0.10	1-Aug-2011	Duffy O'Craven	• Fixed a type "end" for "and", in test 13.1.X6 • Fixed step number reference in step 14 of tests 9.1.1.X4 and 9.1.1.X5 per BTL-CRR-0214_9.1.1.X4_and_9.1.1.X5.doc
9.0.11	28-Sep-2011	Duffy O'Craven	• Incorporated DO-014-01_TimeMaster.doc as tests 13.2.1 through 13.2.7 • Eliminated Chapter 6 Conventions for Specifying BACnet Conformance Tests, since that content is now completely expressed in 135.1-2009 • Corrected the missing underscore typo in Record_Count in test 7.3.2.24.X7, and renumbered the steps to be consecutive.
9.0.12	30-Sep-2011	Duffy O'Craven	• Deleted tests 7.3.1.11, 9.1.1.1, 9.1.1.4, 9.1.2.1 and 9.1.2.5, as the versions from 135.1-2009f-1take precedence. • Deleted tests 10.2.2.3, 10.2.2.7.2, 10.2.3.2, 10.2.3.5, 10.2.4.4, 10.2.4.6, 10.2.4.8, and 10.2.6 as the versions from 135.1-2009g-3 take precedence. • Deleted tests 9.1.1.X1 and 9.1.1.X2 as the versions in 135.1-2009g-4 take precedence. • Deleted test 12.1.1.9.X1 because it is identical to test 12.1.1.9.X in 135.1-2009g-5 • Deleted tests 9.24.1.X2 and 9.24.1.X3 as the versions in 135.1-2009g-8 with numbers 9.24.1.X1 and 9.24.1.X2 take precedence. • Deleted test 7.3.1.1 as the version in 135.1-2009g-9 takes precedence. • Deleted tests 10.X.1, 10.X.2, 10.X.3, and 10.X.4 as the versions in 135.1-2009g-10 - 10.Y.1, 10.Y.2, 10.Y.3, and 10.Y.4 take precedence. • Deleted tests 10.X.5, 10.X.6, and 10.X.7 as the versions in 135.1-2009g-10 - 10.X.1, 10.X.2, and 10.X.3 take precedence. • Added tests 7.3.2.21.1, 7.3.2.21.3.4, and 8.4.8.14 as the versions in 135.1-2009g-11 only portrayed the intended revision with a context-diff, so the entirety of the revised tests is rendered here.

			• Modified test 8.4.2 with the change in 135.1-2009g-11 • Deleted tests 8.18.X3, 8.22.X2, and 8.22.X3 as the versions in 135.1-2009g-14 - 8.18.3, 8.22.4, and 8.22.5 take precedence. • Deleted tests 9.4.X1, 9.4.X2, 9.5.X1, and 9.5.X2 as the versions in 135.1-2009g-15 - 9.4.5, 9.4.6, 9.5.1, and 9.5.2 take precedence.
9.0.13	10-Oct-2011	Duffy O'Craven	• Deleted test 7.3.2.24.9 as the version in 135.1-2009g-16 takes precedence. • Deleted tests 7.3.2.23.3.1, 7.3.2.23.X.3.1, 7.3.2.23.X.3.2, 7.3.2.23.X.3.3, 7.3.2.23.X.3.4, 7.3.2.23.X.3.5, 7.3.2.23.X.3.6 as the versions in 135.1-2009g-17 take precedence. Note that the test numbers used in 135.1-2009g-17 each specify X rather than the X2 used in Test Plan-5.0.final and BTL Specified Test-5.0.final. • Deleted tests 13.X3 and 13.X4 as the 13.X1 and 13.X2 versions in 135.1-2009g-19 take precedence. • Deleted tests 8.3.X1 and 9.3.X8 as the versions 8.3.X and 9.3.1 in 135.1-2009g-20 take precedence. • Deleted tests 7.3.2.23.Y.1, 7.3.2.23.Y.2, 7.3.2.23.Y.3, and 7.3.2.23.Y.4 as the versions in 135.1-2009g-21 take precedence. Note that the test numbers used in 135,1-2009g-21 each specify Y rather than the X1 used in Test Plan-5.0.final and BTL Specified Test-5.0.final. • Corrected COLDSTART to WARMSTART in test 7.3.2.23.5 in accordance with 135.1-2009i-1 • Deleted tests 8.8.1 and 8.8.2 as the versions in 135.1-2009i-5 take precedence. • Deleted tests 8.20.Y1.1 and 8.20.Y1.2 as the versions in 135.1-2009i-6 take precedence.
9.0.14	14-Nov-2011	Duffy O'Craven	• Fixed the number on test 9.16.1.2 (was inadvertently 16.1.1.2 in BTL Specified Tests-5.0.final.) • Put test 13.X6.5.1 in the Table of contents, by giving it Header 4 style. • Removed the Notes to tester: section of test 7.3.1.11 which had had the rest of the test removed in revision 9.0.12. • Separated the Purpose and Test Concept of test 7.3.1.13. • Fixed the indentation of step 14. In test 7.3.1.13 • Removed test 7.3.1.X1 as it is identical to the version in 135.1- 2009d-2 - 7.3.2.10.1 • Added Reason for change (to correct a cut&paste&forgot-to-revise typo in the Test Concept) to test 7.3.2.10.X4 • Added Reason for change (the version in 135.1-2009g-11 only portrays the intended revision with a context-diff, so the entirety of the revised test is rendered here) to test 7.3.2.21.3.4 • Removed test 7.3.1.X2 as it is identical to the version in 135.1- 2009i-15 - 7.3.2.11.X

			• Removed test 7.3.2.21.X1 as it is identical to the version in 135.1- 2009g-7 - 7.3.2.20.X (note that is the second test in that addenda with that same number, there is another in g-6). • Removed tests 9.1.1.X1 and 9.1.1.X2 as the versions in 135.1-2009g-4 take precedence.
9.0.15	23-Nov-2011	Duffy O'Craven	• Removed test 8.34.X1 as it is identical to the version in 135.1- 2009i-12. • Removed tests 9.1.1.X4 and 9.1.1.X5 as the versions in 135.1-2009i-17 take precedence. • Removed test 9.10.1.X2 as the version in 135.1-2009d-1 - 9.10.X takes precedence. • Added Notes to tester: to tests 9.14.2.2 and 9.14.2.3 in consequence of BTL-CRR-0232_9.14.2.2_addl_error_codes.doc, and also applied Protocol_Revision conditional from the version in 135.1-2009i-10 to test 9.14.2.3. • Removed test 8.16.2 because the correction has already been applied in 135.1-2007. • Removed tests 8.16.3, 8.16.4, 9.16.1.1, 9.16.1.3, 9.16.2.2, 9.16.2.3, 9.16.2.4, and 9.16.2.5 because the versions in 135.1-2009f-3 take precedence. Note that BTL - 9.16.1.4 is preserved for it contains a more accurate restriction of "...any unique object identifier of a type that is creatable *and an instance number that is creatable"* . • Removed tests 9.21.1.1, 9.21.1.2, 9.21.1.3, 9.21.1.4.X1, 9.21.1.6.X1, 9.21.1.6.X2, 9.21.1.X1, 9.21.1.X2, and 9.21.2.X4 because the versions in 135.1-2009i-14 take precedence. Note that BTL - 9.21.1.X3 is preserved for it contains a more accurate list: "Qualifying tests are: 9.21.1.1, 9.21.1.2, 9.21.1.3, 9.21.1.4, 9.21.1.4.X1, 9.21.1.X1 or 9.21.1.X2." • Removed test 9.23.2.6 as the version in 135.1-2009i-10 takes precedence. • Removed test 9.20.2.1 as the version in 135.1-2009i-11 takes precedence. • Removed tests 13.X3 and 13.X4 as the versions in 135.1-2009g-19 take precedence. • Test WARMSTART with no Password is made 9.27.1.3, in correspondence with 135.1-2007 numbering. • Removed entire Chapter 14, replicated in 135-2009e-1
9.0.final	01-Dec-2011	Duffy O'Craven	• Updated from 9.0.15 to 9.0.final, accepting all change tracking
12.0.1	25-Jul-2012	Lori Tribble	• Applied Errata 9.0 7/19/2012 • Applied Addendum 9.0-a • Applied Addendum 9.0-b • Applied Addendum 9.0-c • Applied Errata 12.0 7/23/2012
12.0.2	02-Aug-2012	Lori Tribble	• Applied Errata-BTL Test Package 9.0 plus addenda 8/02/2012 (includes above Errata which was not published)
12.0.final	02-Aug-2012	Lori Tribble	• Accepted all changes and Changed Name

12.1.1	27-Sept-2013	Lori Tribble	• Applied Addendum 12.0b
12.1.2	27-Sept-2013	Lori Tribble	• Applied Addendum 12.0c
12.1.3	30-Sept-2013	Lori Tribble	• Applied Addendum 12.0d
12.1.4	30-Sept-2013	Lori Tribble	• Applied Addendum 12.0e
12.1.5	1-Oct-2013	Lori Tribble	• Applied Addendum 12.0f
12.1.6	1-Oct-2013	Lori Tribble	• Applied Addendum 12.0g
12.1.7	1-Oct-2013	Lori Tribble	• Applied Errata 9/30/2013
14.0.a	1-Nov-2014	Lori Tribble	• Applied Addendum 12.1a
14.0.b	1-Nov-2014	Lori Tribble	• Applied Addendum 12.1b
14.0.c	1-Nov-2014	Lori Tribble	• Applied Addendum 12.1c
14.0.d	1-Nov-2014	Lori Tribble	• Applied Addendum 12.1d
14.0.e	1-Nov-2014	Lori Tribble	• Applied Addendum 12.1e
14.0.plus_errata	3-Nov-2014	Lori Tribble	• Updated Reason for Change on all remaining tests. • Removed some tests which existed in 135.1-2013.
14.0.final	19-Nov-2014	Duffy O'Craven	• Removed comments, and pdated to 14.0.final without change.
15.0.05	24-Aug-2017	Lori Tribble	• Applied Addenda 14.0b-j plus errata
15.0.08	25-Sep-2017	Lori Tribble	• Removed test 8.4.X9.
15.0.11	11-Oct-2017	Lori Tribble	• Applied errata from voting members
15.0.final	11-Oct-2017	Lori Tribble	• Accepted all changes
15.1.1	30-Mar-2018	Lori Tribble	• Applied addenda a, b , c, d, and errata
15.1.2	6-Apr-2018	Lori Tribble	• Accepted all changes
15.1.3	26-Apr-2018	Lori Tribble	• Reformatted almost all tests to meet 135.1 formats, applied errata
15.1.4	1-May-2018	Lori Tribble	• Applied Errata
15.1.5	3-May-2018	Lori Tribble	• Fixed formatting and numbering issues.
15.1.final	1-June-2018	Lori Tribble	• Renamed to final