



BACnet® TESTING LABORATORIES

Revision 16.1 Final
SPECIFIED TESTS

Revised December 10, 2019

Table of Contents

1. PURPOSE	14
2. Interim Data Link Layer Tests	15
2.2 MS/TP Data Link Layer Tests	15
2.2.18 Verify Tno_token w/ Serial Analyzer	15
2.2.X1 Data Not For Us Test	15
2.3 ARCNET (twisted pair bus) Data Link Layer Tests	16
2.3.1 Verify the Failsafe Biasing with an Oscilloscope	17
2.3.2 Verify the Basic Signal Duty Cycle with an Oscilloscope	17
3. DEFINITIONS	19
3.x Common language used in tests	19
4. ELECTRONIC PICS FILE FORMAT	19
4.5 Sections of the EPICS File	19
4.5.9 Timers	19
5. EPICS CONSISTENCY TESTS	19
6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS	20
6.3 Time Dependencies	20
6.3.X1 Channel Write Fail Time	20
7. OBJECT SUPPORT TESTS	21
7.1.1 Read Support Test Procedure	21
7.1.2 Non-documented Property Test	22
7.1.X3 Verifying Property_List against the EPICS	22
7.2 Write Support for Properties in Test Database	23
7.2.1 Functional Range Requirements for Property Values	23
7.2.1.3 Octetstrings and Characterstrings	23
7.2.2 Write Support Test Procedure	23
7.2.3 Read-only Property Test	24
7.2.X1 Date Pattern Properties Test	25
7.2.X2 Time Pattern Properties Test	26
7.2.X3 DateTime Pattern Properties Test	26
7.2.X4 Date Non-Pattern Properties Test	27
7.2.X5 Time Non-Pattern Properties Test	27
7.2.X6 DateTime Non-Pattern Properties Test	28
7.2.X7 BACnetDateRange Non-Pattern Properties Test	28
7.2.X8 BACnetDateRange Open-Ended Pattern Properties Test	29
7.3 Object Functionality Tests	30
7.3.1 Property Tests	30
7.3.1.6 Minimum On/Off Time Tests	30
7.3.1.6.1 Override of Minimum Time	30
7.3.1.6.2 Minimum Off Time - Writing at priorities numerically greater than 6	30
7.3.1.6.3 Minimum On Time - Writing at priorities numerically greater than 6	31
7.3.1.6.4 Minimum Off Time - Writing at priorities numerically lesser than 6	32
7.3.1.6.5 Minimum On Time - Writing at priorities numerically lesser than 6	33
7.3.1.6.6 Minimum Off Time - Clock is not affected by additional write operations	34
7.3.1.6.7 Minimum On Time - Clock is not affected by additional write operations	35
7.3.1.6.8 Ensuring Minimum Off Time starts at transition to INACTIVE	36
7.3.1.6.9 Ensuring Minimum On Time starts at transition to ACTIVE	37
7.3.1.6.10 Ensuring Minimum Times Are Not Affected By Time Changes	38
7.3.1.7 COV Tests	40
7.3.1.7.X1 COV_Resubscription_Interval Test	40
7.3.1.8 Binary Object Change of State Tests	41
7.3.1.9 Binary Object Elapsed Active Time Tests	43
7.3.1.10 Event_Enable Tests	46
7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL and TO_NORMAL	46
7.3.1.10.2 Event_Enable Tests for TO_NORMAL only Algorithms	47

BACnet Testing Laboratories - Specified Tests

7.3.1.11 Acked_Transitions Tests.....	49
7.3.1.13 Limit_Enable Tests.....	53
7.3.1.13.1 Limit_Enable Test, LowLimitEnable.....	58
7.3.1.13.2 Limit_Enable Test, HighLimitEnable	59
7.3.1.X4 Event_Message_Texts Tests.....	60
7.3.1.X5 Event_Message_Texts_Config Test.....	62
7.3.1.X6 Event_Algorithm_Inhibit Tests.....	62
7.3.1.X6.1 Event_Algorithm_Inhibit Test.....	62
7.3.1.X6.2 Event_Algorithm_Inhibit Summarization Test	64
7.3.1.X6.3 Event_Algorithm_Inhibit Acknowledgement Test.....	65
7.3.1.X7 Event_Algorithm_Inhibit_Ref Tests	65
7.3.1.X7.1 Event_Algorithm_Inhibit_Ref Test	66
7.3.1.X7.2 Event_Algorithm_Inhibit Writable Test	66
7.3.1.X8 Reliability_Evaluation_Inhibit Tests.....	66
7.3.1.X8.1 Reliability_Evaluation_Inhibit Test.....	66
7.3.1.X8.2 Reliability_Evaluation_Inhibit Summarization Test	67
7.3.1.X9 Event_Detection_Enable Tests.....	68
7.3.1.X9.1 Event_Detection_Enable Inhibits Event Generation	68
7.3.1.X9.2 Event_Detection_Enable Inhibits FAULT.....	68
7.3.1.X16 Array Resizing Test using WritePropertyMultiple Service	69
7.3.1.X18 Non-zero Writable State Count Test.....	70
7.3.1.X19 Non-zero Writable Elapsed Active Time Test.....	71
7.3.1.X20 Strike Count Tests.....	71
7.3.1.X20.1 Non-zero Writable Strike Count Test	71
7.3.1.X20.2 Strike Count Test	72
7.3.1.X41 Blink_Warn Tests	72
7.3.1.X41.Y1 Blink-Warn WARN Command Test.....	72
7.3.1.X41.Y2 Blink-Warn WARN_OFF Command Test.....	73
7.3.1.X41.Y3 Blink-Warn WARN_RELINQUISH Command Test.....	73
7.3.1.X41.Y4 Blink-Warn STOP Command Test	74
7.3.1.X41.Y5 Blink-Warn WARN Command Failure Test.....	75
7.3.1.X41.Y6 Blink-Warn WARN_OFF Command Failure Test	76
7.3.1.X41.Y7 Blink-Warn WARN_RELINQUISH Command Failure Test	77
7.3.1.X41.Y8 Blink-Warn WARN_OFF Command Halted Test.....	78
7.3.1.X41.Y9 Blink-Warn WARN_RELINQUISH Command Halted Test.....	79
7.3.2 Object Specific Tests	80
7.3.2.4 Averaging Object Tests	80
7.3.2.4.1 Reinitializing the Samples.....	80
7.3.2.4.2 Managing the Sample Window	82
7.3.2.9 Command Object Tests.....	83
7.3.2.9.7 Write While In_Process is TRUE Test.	83
7.3.2.10 Device Object Tests.....	83
7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test.....	83
7.3.2.10.6 Successful Increment of the Database_Revision Property after Changing the Object_Identifier Property of an Object.....	86
7.3.2.10.X2 Max_Segments_Accepted at least the minimum.....	87
7.3.2.13 Global Group Object Tests	87
7.3.2.13.X4 Present_Value Tracking and Reliability Test.....	89
7.3.2.13.X5 Present_Value Tracking Test	89
7.3.2.13.X6 COVU_Period and COVU_Recipient Zero Test	89
7.3.2.15 Life Safety Point Object Tests	90
7.3.2.15.X5 Writable Tracking_Value	90
7.3.2.15.X6 Supports Writable Mode Property	91
7.3.2.15.X7 Support Operation_Expected Property	91
7.3.2.15.X8 Support Writable Member_Of Property.....	92
7.3.2.15.X9 Silenced Property Test	92

7.3.2.21 Notification Class Object Tests.....	93
7.3.2.21.3.3 IssueConfirmedNotifications Test.....	95
7.3.2.21.3.5 Recipient_List Property Supports Device Identifier Recipients Test.....	99
7.3.2.21.3.6 Recipient_List Property Supports Network Address Recipients	99
7.3.2.21.3.X7 Recipient_List non-volatility Test	100
7.3.2.21.3.X8 Read-only Recipient_List with internal Notification Forwarder objects	101
7.3.2.21.3.X9 Read-only Recipient_List for external Notification Forwarder Objects	101
7.3.2.22 Program Object Tests.....	101
7.3.2.22.1 Program_Change property test	102
7.3.2.23 Schedule Object Tests.....	102
7.3.2.23.6 Weekly_Schedule Restoration Test.....	102
7.3.2.23.11.1 Internally Written Datatypes Test, non-NULL values	104
7.3.2.24 Log Object Tests	105
7.3.2.24.3 Stop_Time Test.....	105
7.3.2.24.4 Log_Interval Test.....	106
7.3.2.24.X8 Clock-Aligned Logging.....	111
7.3.2.24.X9 Logging Interval_Offset.....	112
7.3.2.25 Event Log Tests.....	112
7.3.2.25.1 Internal Logging of Notifications.....	112
7.3.2.26 Remote Logging of Notifications.....	114
7.3.2.27 Internal Logging of ACK_NOTIFICATIONs	115
7.3.2.28 Remote Logging of ACK_NOTIFICATIONs	116
7.3.2.30 Alert Enrollment Tests.....	117
7.3.2.30.6 Out_Of_Service Property Test.....	117
7.3.2.X37 Accumulator Object Tests.....	118
7.3.2.X37.1.1 Present_Value Remains In-Range Test.....	118
7.3.2.X37.1.3 Logging_Record in Accumulator Test.....	119
7.3.2.X37.1.5 Logging_Record in Accumulator STARTING Test.....	120
7.3.2.X37.1.6 Out_Of_Service Accumulator Test.....	120
7.3.2.X37.1.7 Value_Set Writing Test.....	121
7.3.2.X37.1.8 Value_Before_Change Writing Test.....	121
7.3.2.X38 Pulse Converter Object Tests	122
7.3.2.X38.1.1 Adjust_Value Write Test.....	122
7.3.2.X38.1.2 Scale_Factor Test.....	122
7.3.2.X38.1.3 Out_Of_Service Pulse Converter Test	122
7.3.2.X38.1.5 Update_Time Reflects Change to the Count and is Updated Atomically Test	123
7.3.2.X38.2.1 Adjust_Value Out-of-Range WriteProperty Test	123
7.3.2.X40 Channel Object Tests.....	124
7.3.2.X40.2 Last_Priority Test.....	124
7.3.2.X40.3 WriteGroup Service Support Test.....	124
7.3.2.X40.4 Propagation Entirety Test.....	125
7.3.2.X40.5 Write_Status Test.....	125
7.3.2.X40.6 Allow_Group_Delay_Inhibit Test	126
7.3.2.X40.7 Numeric to BOOLEAN Coercion Rule Test.....	127
7.3.2.X40.8 BOOLEAN to Numeric Coercion Rule Test.....	127
7.3.2.X40.9 Unsigned/INTEGER/REAL/Double to Numeric Coercion Rule Test	128
7.3.2.X40.10 Invalid Datatype Coercion Test	128
7.3.2.X40.11 No Coercion Test	129
7.3.2.X40.12 Write Priority Test	129
7.3.2.X40.13 Writing with a NULL Value Test	130
7.3.2.X53 Load Control Object Tests	131
7.3.2.X53.1 Requested_Shed_Level property test with LEVEL choice.....	131
7.3.2.X53.2 Shed_Levels property test	132
7.3.2.X53.3 Load Control Status_Flags and Reliability Test.....	132
7.3.2.X53.4 Requested_Shed_Level property test with PERCENT choice.....	133
7.3.2.X53.5 Requested_Shed_Level property test with AMOUNT choice	133

BACnet Testing Laboratories - Specified Tests

7.3.2.X54 Lighting Output Object Tests	133
7.3.2.X54.21 - Lighting Output Tracking Test	133
7.3.2.X54.22 - Lighting Output Present Value between 0.0 and 1.0 Test	134
7.3.2.X54.32 Lighting Command Operation FADE_TO Test	135
7.3.2.X54.33 Lighting Command Operation RAMP_TO Test	136
7.3.2.X54.34 Lighting Command Operation STEP_UP Test.....	137
7.3.2.X54.35 Lighting Command Operation STEP_DOWN Test.....	138
7.3.2.X54.36 Lighting Command Operation STEP_ON Test.....	139
7.3.2.X54.37 Lighting Command Operation STEP_OFF Test	140
7.3.2.X54.41 Transition None test	141
7.3.2.X54.42 Transition Test	141
7.3.2.X54.51 Feedback_Value Clamping Test.....	142
7.3.2.X54.61 Min_Actual_Value and Max_Actual_Value Test	143
7.3.2.X54.62 Min_Actual_Value and Max_Actual_Value Scaling Test.....	143
7.3.2.X55 Access Door Object Tests	144
7.3.2.X55.1.X1 Commandable Present_Value Test	144
7.3.2.X55.1.X2 Door_Status, Lock_Status and Door_Alarm_State Tests.....	145
7.3.2.X55.1.X3 Door_Status with Physical Door Status Tests	145
7.3.2.X55.1.X4 Lock_Status Tests	146
7.3.2.X55.1.X5 Secured_Status Tests.....	147
7.3.2.X55.1.X6 Door_Unlock_Delay_Time Test.....	148
7.3.2.X55.1.X7 Masked_Alarm_Values Tests	149
7.3.2.X55.1.X8 Door_Open_Too_Long Test.....	150
7.3.2.X56 Access Point Object Tests.....	150
7.3.2.X56.1 Authentication_Status and Access_Event Test	152
7.3.2.X56.2 Allowed Access Test.....	153
7.3.2.X56.3 Denied Access Test.....	153
7.3.2.X56.4 Authorization Mode Test.....	154
7.3.2.X56.5 Access Rights Exemptions Test.....	156
7.3.2.X56.6 Change Authentication Policy Test	157
7.3.2.X56.7 Lockout State Test	158
7.3.2.X56.8 Threat Level Test	159
7.3.2.X56.9 Denied Access Occupancy Upper Limit Test.....	160
7.3.2.X56.10 Denied Access Disabled Credential Test	161
7.3.2.X57 Access Zone Object Tests	161
7.3.2.X57.1 Occupancy State Test.....	162
7.3.2.X57.2 Occupancy Counting Test	163
7.3.2.X57.3 Keeping Track of Credentials Test	164
7.3.2.X57.4 Passback Mode Test.....	164
7.3.2.X59 Access Rights Object Tests	166
7.3.2.X59.1 Enable Test	167
7.3.2.X59.2 Negative Rules Test	168
7.3.2.X59.3 Positive Access Rules Test.....	168
7.3.2.X59.4 Accompaniment Test	168
7.3.2.X60 Access Credential Object Tests	169
7.3.2.X60.1 Credential Status, Credential Disable and Reason for Disable Test.....	170
7.3.2.X60.2 Activation Time and Expiration Time Test.....	171
7.3.2.X60.3 Disabled Access Rights Test	171
7.3.2.X60.4 Days Remaining and Uses Remaining Test	172
7.3.2.X60.5 Absentee Limit Test.....	173
7.3.2.X60.6 Last Access Point, Last Use Time and Last Access Event Test.....	173
7.3.2.X60.7 Extended Time Enable Test.....	174
7.3.2.X61 Credential Data Input Object Tests.....	174
7.3.2.X61.1 Return from Out Of Service Undefined test.....	175
7.3.2.X61.2 Read Valid Authentication Factor Test.....	175
8. APPLICATION SERVICE INITIATION TESTS.....	177

8.1	AcknowledgeAlarm Service Initiation Tests.....	177
8.1.X2	Successful Alarm Acknowledgment of Confirmed Event Notifications Using the 'Initiating Device Identifier' Parameter.....	177
8.2	ConfirmedCOVNotification Service Initiation Tests.....	179
8.2.1	Change of Value Notification from an Analog Input, Analog Output, and Analog Value a Numeric Object's Present_Value Property.....	179
8.2.2	Change of Value Notification from an Analog Input, Analog Output, and Analog Value a Numeric Object's Status_Flags Property.....	180
8.2.3	Change of Value Notification from a Binary Input, Binary Output, and Binary Value Discrete Valued Object's Present_Value Property.....	182
8.2.4	Change of Value Notification from a Binary Input, Binary Output, and Binary Value Discrete Valued Object's Status_Flags Property.....	183
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 Multi-state or other Discrete Valued Object Present_Value Property.....	184
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 Multi-state or other Discrete Valued Object Status_Flags Property.....	185
8.2.7	Change of Value Notification from Loop Object Present_Value Property	186
8.2.8	Change of Value Notification from a Loop Object Status_Flags Property.....	188
8.2.X9	ConfirmedCOVNotification Pulse Converter changing Present_Value	189
8.2.X10	ConfirmedCOVNotification Pulse Converter changing Status_Flags	191
8.2.X11	Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property.....	192
8.3	UnconfirmedCOVNotification Service Initiation Tests.....	193
8.3.1	Change of Value Notification from an Analog Input, Analog Output, and Analog Value a Numeric Object's Present_Value Property.....	193
8.3.2	Change of Value Notification from an Analog Input, Analog Output, and Analog Value a Numeric Object's Status_Flags Property.....	194
8.3.3	Change of Value Notification from a Binary Input, Binary Output, and Binary Value Discrete Valued Object's Present_Value Property.....	194
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 Multi-state or other Discrete Valued Object Present_Value Property.....	194
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 Multi-state or other Discrete Valued Object Status_Flags Property.....	195
8.3.7	Change of Value Notification from Loop Object Present_Value Property	195
8.3.10	Device Restart Notifications.....	195
8.3.X1	COVU_Recipients Notifications.....	196
8.3.X11	Unsubscribed COV Service Initiation Test	197
8.3.X12	UnconfirmedCOVNotification Pulse Converter changing Present_Value.....	198
8.3.X13	UnconfirmedCOVNotification Pulse Converter changing Status_Flags	198
8.3.X14	Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property.....	198
8.4	ConfirmedEventNotification Service Initiation Tests.....	198
8.4.4	COMMAND_FAILURE Tests	198
8.4.X1	DOUBLE_OUT_OF_RANGE Tests (ConfirmedEventNotification).....	200
8.4.X2	SIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification).....	200
8.4.X3	UNSIGNED_OUT_OF_RANGE Tests (ConfirmedEventNotification).....	200
8.4.X4	CHANGE_OF_CHARACTERSTRING Tests (ConfirmedEventNotification).....	201
8.4.X6	Extended Algorithm Tests (ConfirmedEventNotifications).....	204
8.4.X7	UNSIGNED_RANGE ConfirmedEventNotification Test.....	205
8.4.X8	CHANGE_OF_STATUS_FLAGS Test (ConfirmedEventNotification).....	207
8.4.X9	CHANGE_OF_RELIABILITY with the FAULT_OUT_OF_RANGE Algorithm	209
8.4.X10	CHANGE_OF_DISCRETE_VALUE Test (ConfirmedEventNotification).....	210

8.5 UnconfirmedEventNotification Service Initiation Tests	211
8.5.X1 DOUBLE_OUT_OF_RANGE Tests (UnconfirmedEventNotification)	211
8.5.X2 SIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification)	211
8.5.X3 UNSIGNED_OUT_OF_RANGE Tests (UnconfirmedEventNotification)	211
8.5.X4 CHANGE_OF_CHARACTERSTRING Tests (UnconfirmedEventNotification)	211
8.5.X5 Proprietary Algorithm Tests (UnconfirmedEventNotifications)	212
8.5.X6 Extended Algorithm Tests (UnconfirmedEventNotifications)	212
8.5.X8 CHANGE_OF_STATUS_FLAGS Test (UnconfirmedEventNotification)	213
8.5.X9 CHANGE_OF_RELIABILITY Tests	213
8.5.X9.1 CHANGE_OF_RELIABILITY with No Fault Algorithm	213
8.5.X9.2 CHANGE_OF_RELIABILITY with the FAULT_CHARACTERSTRING Algorithm	214
8.5.X9.3 CHANGE_OF_RELIABILITY with the FAULT_EXTENDED Algorithm	216
8.5.X9.4 CHANGE_OF_RELIABILITY with the FAULT_LIFE_SAFETY Algorithm	217
8.5.X9.5 CHANGE_OF_RELIABILITY with the FAULT_STATE Algorithm	218
8.5.X9.6 CHANGE_OF_RELIABILITY with the FAULT_STATUS_FLAGS Algorithm	219
8.5.X9.7 Event Enrollment Fault Condition Precedence Tests	220
8.5.X9.7.1 Internal Faults Take Precedence Over Monitored Object Faults	220
8.5.X9.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms	221
8.5.X9.7.3 Internal Faults Take Precedence Over Fault Algorithms	221
8.5.X9.8 CHANGE_OF_RELIABILITY of Event Enrollment Object, Monitored Object Fault	222
8.5.X9.9 CHANGE_OF_RELIABILITY of Event Enrollment Object Fault	223
8.5.X9.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL	224
8.5.X9.11 CHANGE_OF_RELIABILITY with First Stage Object Fault	226
8.5.X9.15 CHANGE_OF_RELIABILITY with the FAULT_OUT_OF_RANGE Algorithm	227
8.5.X10 CHANGE_OF_DISCRETE_VALUE Test (UnconfirmedEventNotification)	227
8.11 SubscribeCOVProperty Service Initiation Tests	228
8.11.1 Confirmed Notifications Subscription	228
8.11.2 Unconfirmed Notifications Subscription	228
8.11.3 Canceling a Subscription	229
8.11.X1 Change of Value Notification Tests	229
8.11.X1.1 Change of Value Notification	229
8.11.X1.2 Change of Value Notifications with Invalid Process Identifier	230
8.11.X1.3 Change of Value Notification Arrives after Subscription has Expired	231
8.11.X1.4 Change of Value Notifications with Invalid Monitored Object Identifier	233
8.11.X1.5 Change of Value Notifications with Invalid Monitored property	233
8.11.X4 Requests 8 Hour Lifetimes	234
8.20 ReadPropertyMultiple Service Initiation Tests	235
8.20.5 Cases In Which ReadProperty Shall Be Used After ReadPropertyMultiple Fails	235
8.20.5.1 The IUT Determines the TD does not Support the ReadPropertyMultiple Service	235
8.21 ReadRange Service Initiation Tests	235
8.21.1 Reading Values with no Specified Range	235
8.21.3 Reading a Range of Values by Position	236
8.21.9 Presents Log Records Containing a Specific Datatype	236
8.22 WriteProperty Service Initiation Tests	236
8.22.X4 Writing Array Properties as a Whole Array	236
8.24 DeviceCommunicationControl Service Initiation Tests	237
8.24.1 Indefinite Duration, Disable, No Password	237
8.24.2 Indefinite Duration, Disable, Password	237
8.24.3 Time Duration, Disable, Password	237
8.24.4 Enable, Password	238
8.24.5 Enable, No Password	238
8.24.6 Time Duration, Disable, No Password	238
8.24.7 Time Duration, Disable-Initiation, Password	239
8.27 ReinitializeDevice Service Initiation Tests	239
8.27.2 COLDSTART with a Password	239
8.27.4 WARMSTART with a Password	239

8.32 Who-Has Service Initiation Tests	239
8.32.1 Object Identifier Selection with no Device Instance Range.....	239
8.32.2 Object Name Selection with no Device Instance Range.....	240
8.32.3 Object Identifier Selection with a Device Instance Range.....	240
8.32.4 Object Name Selection with a Device Instance Range.....	241
8.34 Who-Is Service Initiation Tests.....	241
8.34.2 Who-Is Request with a Device Instance Range.....	241
9. APPLICATION SERVICE EXECUTION TESTS.....	242
9.1 AcknowledgeAlarm Service Execution Tests	242
9.1.1 Positive AcknowledgeAlarm Service Execution Tests.....	242
9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter.....	242
9.1.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications using the Sequence Number Form of the 'Time of Acknowledgment' Parameter	245
9.1.1.3 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter.....	245
9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter.....	246
9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter	248
9.1.1.6 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter.....	248
9.1.1.8 Successful Alarm Acknowledgment of Confirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter.....	249
9.1.1.9 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter.....	251
9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications	252
9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications	255
9.1.1.X3 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is either High-Limit or Low-Limit	257
9.1.2 Negative AcknowledgeAlarm Service Execution Tests.....	257
9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old.....	258
9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event Object Identifier' is Invalid	260
9.1.2.4 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event State Acknowledged' is Invalid.....	260
9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old.....	261
9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the Referenced Object Does Not Exist	264
9.1.2.7 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event State Acknowledged' is Invalid.....	264
9.1.X1 Unsupported Acknowledgment Source Character Set AcknowledgeAlarm Test.....	265
9.2 ConfirmedCOVNotification Service Execution Tests	266
9.2.1 Positive ConfirmedCOVNotification Service Execution Tests.....	266
9.2.1.X4 Change of Value Notification from Proprietary Objects	266
9.2.1.X5 ConfirmedCOVNotification from Access Door Object	266
9.2.2 Negative ConfirmedCOVNotification Service Execution Tests	267
9.2.2.1 Change of Value Notification Arrives after Subscription has Expired	267
9.2.2.2 Change of Value Notifications with Invalid Process Identifier	267
9.2.2.4 Change of Value Notifications with Invalid Monitored Object Identifier	268
9.3 UnconfirmedCOVNotification Service Execution Tests	269
9.3.1.X6 UnconfirmedCOVNotification from Access Door Object	269
9.3.X9 Change of Value Notification from Proprietary Objects.....	269
9.4 ConfirmedEventNotification Service Execution Tests	269

9.4.5 ConfirmedEventNotification Simple Presentation	269
9.4.6 ConfirmedEventNotification Full Presentation	270
9.4.X1 Unsupported Message Text Character Set ConfirmedEventNotificationTest.....	271
9.5 UnconfirmedEventNotification Service Execution Tests	271
9.5.X1 Unsupported Message Text Character Set UnconfirmedEventNotificationTest.....	271
9.7 GetEnrollmentSummary Service Execution Tests.....	272
9.7.1 Required GetEnrollmentSummary Filters.....	272
9.7.1.1 Enrollment Summary with Zero Summaries	272
9.7.2 User Selectable GetEnrollmentSummary Filters.....	272
9.7.2.3 Event Type Filter.....	272
9.8 GetEventInformation Service Execution Tests.....	273
9.8.6 Chaining Test.....	273
9.10 SubscribeCOV Service Execution Tests	274
9.10.1 Positive SubscribeCOV Service Execution Tests.....	274
9.10.1.7 Finite Lifetime Subscriptions	274
9.10.1.X1 Ensuring 5 Concurrent COV Subscribers	275
9.10.2 Negative SubscribeCOV Service Execution Tests	277
9.10.2.1 The Monitored Object Does Not Support COV Notification	277
9.10.2.X1 The Monitored Object Does Not Exist	277
9.10.2.X2 There Is No Space For A Subscription.....	278
9.10.2.X3 The Lifetime Parameter is Out of Range.....	278
9.10.3 Positive Unsubscribed COVNotification Execution Tests.....	279
9.10.3.X1 Unsubscribed COVNotification Execution Test	279
9.11 SubscribeCOVProperty Service Execution Tests	280
9.11.1 Positive SubscribeCOVProperty Service Execution Tests	280
9.11.1.1 Confirmed COV Notifications	280
9.11.1.2 Unconfirmed COV Notifications	280
9.11.1.5 Canceling Expired or Non-Existing Subscriptions.....	281
9.11.1.7 Finite Lifetime Subscriptions.....	281
9.11.1.9 Client-Supplied COV Increment.....	283
9.11.1.X10 Accepts SubscribeCOVProperty-Requests with 8 Hour Lifetimes.....	284
9.11.1.X11 Confirmed Change of Value Notification from Property Value	285
9.11.1.X12 Unconfirmed Change of Value Notification from Property Value	285
9.11.1.X21 Confirmed Change of Value Notification from Status_Flags Property.....	286
9.11.1.X22 Unconfirmed Change of Value Notification from Status_Flags Property.....	287
9.11.2 Negative SubscribeCOVProperty Service Execution Tests.....	287
9.11.2.1 The Monitored Object Does Not Support COV Notification	287
9.11.2.2 The Monitored Property Does Not Support COV Notification	287
9.11.2.X11 Monitored Object Does Not Exist	288
9.11.2.X12 Monitored Property Does Not Exist	288
9.11.2.X13 There Is No Space For Subscription	289
9.11.2.X14 The Lifetime Parameter is Out of Range	289
9.12 AtomicReadFile Service Execution Tests	290
9.12.1 Positive AtomicReadFile Service Execution Tests.....	290
9.12.1.2.1 Reading an Entire <i>Stream-Based</i> File.....	290
9.13 AtomicWriteFile Service Execution Tests	290
9.13.1 Positive AtomicWriteFile Service Execution Tests.....	290
9.13.1.2.1 Writing an Entire <i>Stream-Based</i> File.....	290
9.13.1.2.3 Appending Data to the End of a File.....	292
9.14 AddListElement Service Execution Tests	293
9.14.2 Negative AddListElement Service Execution Tests.....	293
9.14.2.2 Adding a List Element With an Invalid Datatype	293
9.14.2.3 An AddListElement Failure Part Way Through a List.....	293
9.15 RemoveListElement Service Execution Tests	294
9.15.2 Negative RemoveListElement Service Execution Tests.....	294
9.15.2.2 A RemoveListElement Failure Part Way Through a List.....	294

9.16 CreateObject Service Execution Tests	295
9.16.1 Positive CreateObject Service Execution Tests.....	295
9.16.1.2 Creating Objects by Specifying the Object Identifier with No Initial Values	295
9.16.1.4 Creating Objects by Specifying the Object Identifier and Providing Initial Values ...	295
9.16.2 Negative CreateObject Service Execution Tests	295
9.16.2.1 Attempting to Create an Object That Does Not Have a Unique Object Identifier.....	295
9.16.2.4 Attempting to Create an Object with an Object Type Specifier and an Error in the Initial Values.....	296
9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values.....	297
9.16.2.6 Attempting to Create an Object with an instance of 4194303	298
9.16.2.X1 Attempting to Create a non-Supported Object Type (by Object Type)	298
9.16.2.X2 Attempting to Create a non-Supported Object Type (by Object Identifier).....	299
9.17 DeleteObject Service Execution Tests	299
9.17.2 Negative DeleteObject Service Execution Tests	299
9.17.2.1 Attempting to Delete an Object That is Not Deletable	299
9.18 ReadProperty Service Execution Tests	300
9.18.1 Positive ReadProperty Service Execution Tests.....	300
9.18.1.2 Reading a Single Element of an Array	300
9.18.1.X1 Reading Properties Based on Data Type.....	300
9.18.1.X3 Respects max-segments-accepted bit pattern.....	300
9.18.1.X4 Reading Array Properties at different Array Indexes	301
9.18.1.X8 ReadProperty Service when Non-BACnet Device Offline.....	301
9.20.1.X9 ReadPropertyMultiple Service when Non-BACnet Device Offline.....	302
9.21.1.X10 ReadRange Service when Non-BACnet Device Offline	302
9.20 ReadPropertyMultiple Service Execution Tests	303
9.20.1 Positive ReadPropertyMultiple Service Execution Tests.....	303
9.20.1.1 Reading a Single Property from a Single Object	303
9.20.1.2 Reading Multiple properties from a Single Object	303
9.20.1.3 Reading a Single Property from Multiple Objects	303
9.20.1.4 Reading Multiple Properties from Multiple Objects.....	304
9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error.....	304
9.20.1.6 Reading Multiple Properties with Multiple Embedded Access Errors.....	305
9.20.1.7 Reading ALL Properties	305
9.20.1.8 Reading OPTIONAL Properties	306
9.20.1.9 Reading REQUIRED Properties	306
9.20.1.X1 Reading Properties Based on Data Type.....	307
9.20.1.X2 ReadPropertyMultiple Array Properties	307
9.21 ReadRange Service Execution Tests.....	308
9.21.1 Positive ReadRange Service Execution Tests	308
9.21.1.X1 ReadRange Support for All List Properties	308
9.21.2 Negative ReadRange Service Execution Tests	309
9.21.2.1 Attempting to Read a Property That Does not Exist	309
9.21.2.2 Attempting to Read a Property That is not a List.....	309
9.21.2.3 Attempting to Read a non-Array Property with an Array Index.....	310
9.22 WriteProperty Service Execution Tests.....	310
9.22.1 Positive WriteProperty Service Execution Tests	310
9.22.1.1 Writing a Single Element of an Array	310
9.22.1.2 Writing a Commandable Property Without a Priority.....	310
9.22.1.3 Writing a Non-Commandable Property with a Priority.....	311
9.22.1.X1 Writing an Array Size	311
9.22.1.X2 Writing to Properties Based on Data Type	312
9.22.2 Negative WriteProperty Service Execution Tests	312
9.22.2.1 Writing Non-Array Properties with an Array Index	312
9.22.2.2 Writing Array Properties with an Array Index that is Out of Range.....	313
9.22.2.3 Writing with a Property Value Having the Wrong Datatype.....	313

9.22.2.4 Writing with a Property Value that is Out of Range	314
9.22.2.X1 Writing Non-Array Read-only Property with an Array Index	314
9.22.2.X2 Resizing a writable fixed size array property.....	315
9.23 WritePropertyMultiple Service Execution Tests	316
9.23.1 Positive WritePropertyMultiple Service Execution Tests.....	316
9.23.1.1 Writing a Single Property to a Single Object.....	316
9.23.1.2 Writing Multiple properties to a Single Object.....	316
9.23.1.3 Writing a Single Property to Multiple Objects	317
9.23.1.4 Writing Multiple Properties to Multiple Objects	317
9.23.1.X4 Writing an Array Size	318
9.23.2 Negative WritePropertyMultiple Service Execution Tests	318
9.23.2.1 Writing Multiple Properties with a Property Access Error.....	318
9.23.2.2 Writing Multiple Properties with an Object Access Error.....	319
9.23.2.3 Writing Multiple Properties with a Write Access Error	319
9.23.2.4 Writing Non-Array Properties with an Array Index	320
9.23.2.5 Writing Array Properties with an Array Index that is Out of Range.....	320
9.23.2.6 Writing with a Property Value Having the Wrong Datatype.....	321
9.23.2.7 Writing with a Property Value that is Out of Range	322
9.23.2.X1 WritePropertyMultiple Reject Test.....	322
9.23.2.X2 Resizing a writable fixed size array property using WritePropertyMultiple service	323
9.23.2.X3 Writing first element of 'List of Write Access Specifications' with Object Access Error	324
9.23.2.X4 Writing first element of 'List of Write Access Specifications' with a Write Access Error	325
9.23.2.X5 WritePropertyMultiple Reject Test for first element of 'List of Write Access Specifications'	325
9.23.2.X6 Writing first element of 'List of Write Access Specifications' with a Property Access Error	326
9.23.2.X9 Date Non-Pattern Properties Test using WritePropertyMultiple Service.....	326
9.23.2.X10 Time Non-Pattern Properties Test using WritePropertyMultiple Service	327
9.23.2.X11 DateTime Non-Pattern Properties Test using WritePropertyMultiple Service.....	328
9.23.2.X12 BACnetDateRange Non-Pattern Properties Test using WritePropertyMultiple Service	328
9.24 DeviceCommunicationControl Service Execution Test.....	329
9.24.1 Positive DeviceCommunicationControl Service Execution Tests	329
9.24.1.5 Finite Time Duration Restored by ReinitializeDevice	329
9.24.2 Negative DeviceCommunicationControl Service Execution Tests	330
9.24.2.3 Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'	330
9.27 ReinitializeDevice Service Execution Tests	332
9.27.2 Negative ReinitializeDevice Service Execution Tests.....	332
9.27.2.3 COLDSTART with Missing <i>or Invalid</i> Password	332
9.27.2.4 WARMSTART with Missing <i>or Invalid</i> Password	332
9.29 UnconfirmedTextMessage Service Execution Tests.....	333
9.29.1 UnconfirmedTextMessage With No Message Class	333
9.29.2 UnconfirmedTextMessage With an Unsigned Message Class.....	334
9.29.3 UnconfirmedTextMessage With a CharacterString Message Class	334
9.30 TimeSynchronization Service Execution Tests	334
9.30.1 Positive TimeSynchronization Service Execution Tests.....	334
9.30.1.1 TimeSynchronization Local Broadcast.....	335
9.30.1.2 TimeSynchronization Directed to the IUT	336
9.31 UTCTimeSynchronization Service Execution Tests.....	336
9.31.1 Positive UTCTimeSynchronization Service Execution Tests	337
9.31.1.1 UTCTimeSynchronization Local Broadcast.....	337
9.31.1.2 UTCTimeSynchronization Directed to the IUT.....	337
9.32 Who-Has Service Execution Tests.....	337
9.32.1 Execution of Who-Has Service Requests Originating from the Local Network.....	338

9.32.1.1 Object ID Version with No Device Range	338
9.32.1.2 Object Name Version with no Device Range	338
9.32.1.3 Object ID Version with IUT Inside of the Device Range.....	338
9.32.1.4 Object ID Version with IUT Outside of the Device Range.....	339
9.32.1.5 Object Name Version with IUT Inside of the Device Range	339
9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range	340
9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range	340
9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range	341
9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range.....	341
9.32.1.11 Object Name Version, Directed to a Specific MAC Address.....	342
9.32.1.12 Who-Has After Object_Name Changed	342
9.32.1.13 Who-Has After Object_Identifier Changed	343
9.32.2 Execution of Who-Has Service Requests Originating from a Remote Network	344
9.32.2.1 Object ID Version, Global Broadcast from a Remote Network	344
9.32.2.2 Object ID Version, Remote Broadcast	344
9.32.2.X3 - Who-Has for Non-existent Object_Name.....	345
9.32.2.X5 Who-Has for Non-existent Object_Identifier.....	345
9.33 Who-Is Service Execution Tests	346
9.33.1 Execution of Who-Is Service Requests Originating from the Local Network.....	346
9.33.1.3 Local Broadcast, Specific Device Inquiry with IUT Outside of the Device Range.....	346
9.33.2 Execution of Who-Is Service Requests Originating from a Remote Network.....	346
9.33.2.3 General Inquiry, Directed to a Remote Device.....	346
9.X40 WriteGroup Tests	346
9.X40.1.X1 Channel and Group Number Test.....	346
9.X40.1.X2 Write Priority and Overriding Priority Test	348
9.X40.1.X3 Relinquish Control Test	348
9.X40.1.X4 Inhibit Delay Test with WriteGroup.....	349
10. NETWORK LAYER PROTOCOL TESTS.....	350
10.1.1 Processing Application Layer Messages Originating from Remote Networks.....	350
10.2 Router Functionality Tests.....	351
10.2.2 Processing Network Layer Messages	351
10.2.2.7.2 Unknown Network Layer Message Type	351
10.2.X1 Initiates Network-Number-Is on Startup.....	352
10.2.X2 Routers Execute What-Is-Network-Number	353
10.6 Non-Router Functionality Tests.....	353
10.6.3 Ignore Router Commands	353
10.7 Router Functionality.....	354
10.7.2 Router Binding via Application Layer Services	354
10.8 Virtual Routing Functionality Tests.....	356
10.8.3 Routing of Unicast APDUs	357
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.....	357
10.8.3.2 Route Request Message from a Virtual Device to a Local Device	358
10.8.3.5 Unicast Messages That Should Not Be Routed.....	359
10.8.3.5.1 Unknown Network	359
10.8.3.6.X1 Silently Drop Messages to a Virtual Device that is Offline	359
10.8.4 Routing of Broadcast APDUs to Virtual Devices.....	360
10.8.4.7 Route Remote Broadcast Message from a Virtual Device to a Local Network.....	360
10.8.7 Multiple Devices on a Single Virtual Network	360
10.8.7.4 Who-Is Specifying Unknown Device Ids	361
10.8.7.5 Who-Has Specifying Unknown Device Ids.....	361
12. DATA LINK LAYER PROTOCOLS TESTS.....	361

12.1 MS/TP State Machine Tests.....	361
12.1.3 MS/TP Data Link Layer Tests (Alternate).....	361
12.1.3.3 Verify T_{frame_gap}	361
12.1.3.X1 Frame Type Based on Transmitted NPDU size	362
12.1.3.X2 Executing COBS Encoded Frames.....	362
13. SPECIAL FUNCTIONALITY TESTS	363
13.1 Segmentation	363
13.1.12.1 IUT Does Not Support Segmented Response	363
13.1.12.X1 Reading with maximum-segments-accepted bit pattern B'000'	363
13.8 Backup and Restore Procedure Tests	364
13.8.1 Backup and Restore Execution Tests.....	364
13.8.1.1 Execution of Full Backup and Restore Procedure.....	364
13.8.1.2 Attempting a Backup Procedure While Already Performing a Backup Procedure.....	367
13.8.1.3 Attempting a Backup Procedure While Already Performing a Restore Procedure.....	368
13.8.1.4 Attempting a Restore Procedure While Already Performing a Backup Procedure.....	369
13.8.1.5 Attempting a Restore Procedure While Already Performing a Restore Procedure.....	369
13.8.1.6 Ending Backup and Restore Procedures via Timeout	370
13.8.1.7 Ending Backup and Restore Procedures via Abort	371
13.8.1.8 Attempting a Backup Procedure with an Invalid Password.....	372
13.8.1.9 Attempting a Restore Procedure with an Invalid Password.....	373
13.8.1.10 Starting and Ending a Backup Procedure when a Password is not Required	374
13.8.1.11 Starting and Ending a Restore Procedure when a Password is not Required	374
13.8.1.12 System_Status during a Backup Procedure.....	375
13.8.1.13 System_Status during a Restore Procedure.....	375
13.8.2 Backup and Restore Initiation Tests	376
13.8.2.1 Initiate a Full Backup and Restore	376
13.X13 General Application State Machine Tests.....	378
13.X13.1 Ignore Confirmed Broadcast Requests.....	378
14. BACnet/IP FUNCTIONALITY TESTS	378
14.1 Non-BBMD B/IP Device.....	378
14.1.7 Forwarded-NPDU (One-hop Distribution)	378
14.1.8 Original-Broadcast-NPDU	378
14.1.10 Forwarded-NPDU (Two-hop Distribution).....	379
14.1.X11 Processing Forwarded-NPDU request initiated from different port	379
14.1.X12 Processing Forwarded-NPDU request initiated from different port when registered as a Foreign Device into a BBMD.	380
14.2 BBMD B/IP Device with a Server Application	380
14.2.1 Execute Forwarded-NPDU.....	380
14.2.1.1 Execute Forwarded-NPDU (One-hop Distribution).....	380
14.2.1.2 Execute Forwarded-NPDU (Two-hop Distribution).....	381
14.2.2 Execute Original-Broadcast-NPDU	382
14.2.2.1 Execute Original-Broadcast-NPDU (One-hop Distribution).....	382
14.2.2.2 Execute Original-Broadcast-NPDU (Two-hop Distribution).....	383
14.3.3 Verify Broadcast Distribution Table Created from the Configuration Saved During the Previous Session	384
14.7 Broadcast management (BBMD, Foreign Devices, Local Application).....	384
14.7.1 Broadcast Message from Directly Connected IP Subnet	384
14.7.1.1 Broadcast Message from Directly Connected IP Subnet (One-hop Distribution)	384
14.7.1.2 Broadcast Message from Directly Connected IP Subnet (Two-hop Distribution)	386
14.7.2 Broadcast Message Forwarded by a Peer BBMD	387
14.7.2.1 Broadcast Message Forwarded by a Peer BBMD (One-hop Distribution)	387
14.7.2.2 Broadcast Message Forwarded by a Peer BBMD (Two-hop Distribution).....	388
14.7.3 Broadcast Message from a Foreign Device.....	389
14.7.3.1 Broadcast Message From a Foreign Device (One-hop Distribution).....	389
14.7.3.2 Broadcast Message From a Foreign Device (Two-hop Distribution)	390
14.8 Foreign Device Tests.....	391

BACnet Testing Laboratories - Specified Tests

14.8.1 Registering as a Foreign Device	391
14.8.X1 Register-Foreign-Device Enable and Disable Test.....	391
14.8.X2 Recurring Register-Foreign-Device Test.....	392
14.8.X3 BBMD Address Configuration Test.....	392
14.8.X4 Transmits a Broadcast at Startup preceded by Register-Foreign-Device.....	393
14.8.X5 Time-to-Live Configuration Test.....	393
14.9.1 Distribute-Broadcast-To-Network.....	393

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 than 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.

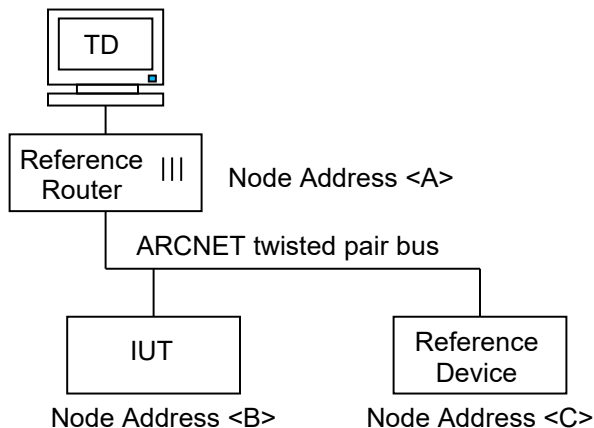
These tests require the use of a reference ARCNET twisted pair bus router and a reference ARCNET twisted pair bus device. These devices will be selected from a pool of qualified devices that are to be submitted by members of the BMA. The tester is free to select any of the qualified reference devices 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 .



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

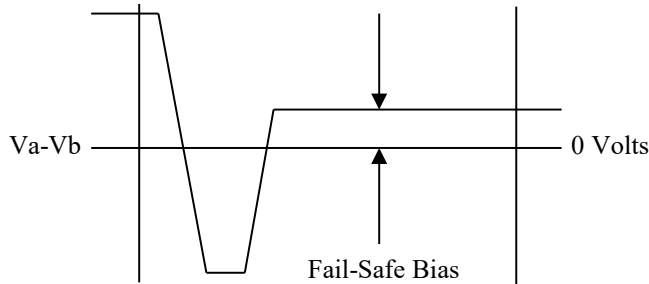
Reason for Change: No test exists for this functionality. This test is included in CLB-015.

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

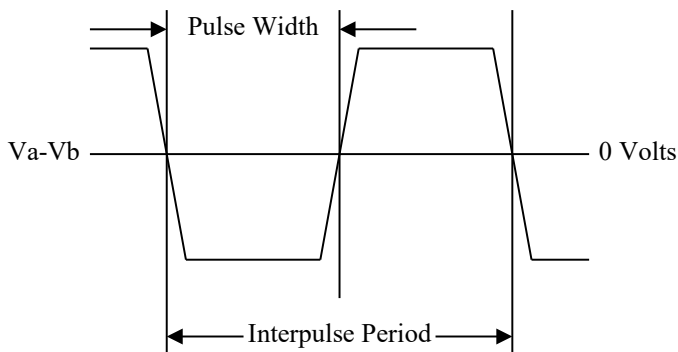
Reason for Change: No test exists for this functionality. This test is included in CLB-015.

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.

4. ELECTRONIC PICS FILE FORMAT

4.5 Sections of the EPICS File

4.5.9 Timers

This section defines timer values that are used to determine when a test has failed because an appropriate response has not been observed by the TD. A Real value in seconds must be provided for each timer. See 6.3.

Fail Times: {

{

Notification Fail Time: []

Internal Processing Fail Time: []

Minimum ON/OFF Time: []

Schedule Evaluation Fail Time: []

External Command Fail Time: []

Program Object State Change Fail Time: []

Acknowledgement Fail Time: []

Slave Proxy Confirm Interval: []

Unconfirmed Response Fail Time: []

Channel Write Fail Time: []

}

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.

6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

6.3 Time Dependencies

6.3.X1 Channel Write Fail Time

The Channel Write Fail Time is the elapsed time, in seconds, between a change to the Present_Value of a Channel object and when a test is considered to have failed because the first write operation associated with the newly written value state has not

been performed. If the Channel object has multiple target properties to write to, the time to write all of them would be less than or equal to the number of target properties times this value.

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*) THEN
 - 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,
 - 'Property 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 WritePropertyMultiple 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) THEN
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,
            'Property 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)
'Property 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,
    'Property 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~~ *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 an 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. The day-of-week field is redundant information and can be regenerated from the other fields. In order to not fail devices which always ensure this field is consistent with the other fields in the date value, the use of an unspecified day of week is always tested in conjunction with another pattern value.

Test Steps:

1. IF (Protocol_Revision is not present or Protocol_Revision < 4) THEN
 Specials = (year unspecified, month unspecified, day of month unspecified)
 ELSE IF (Protocol_Revision >= 4 and Protocol_Revision < 10) THEN
 Specials = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month)
 ELSE
 Specials = (year unspecified, month unspecified, day of month unspecified, odd months, even months, last day of month, even days, odd days)
2. REPEAT SV = (each value in Specials) DO {
 IF (SV <> day of week unspecified) THEN
 V1 = D1 updated with the value SV
 ELSE
 V1 = D1 updated with the value SV and any other value from Specials
 WRITE P1 = (V1)
 VERIFY P1 = (V1)
 }

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) *THEN*
 DateSpecials = (year unspecified, month unspecified, day of month unspecified, day of week unspecified)
 ELSE IF (Protocol_Revision >= 4 and Protocol_Revision < 10) *THEN*
 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 135-2008ac-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 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 135-2008h-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 datetime field values to ensure that the property does not accept them. A datetime DT₁ is selected which is within the range that the IUT will accept for the property. The value, V₁, written to the property is the datetime DT₁ 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' = O₁,
 - 'Property Identifier' = P₁,
 - 'Property Value' = (DT₁ 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 135-2008ac-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, P₁ is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange DR₁ is selected so it is within the range that the IUT will accept for the property. The value, V₁, written to the property is the

dateRange DR₁ with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalendarEntry, 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' = O₁,
 - 'Property Identifier' = P₁,
 - 'Property Value' = (DR₁ 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 P₁ 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 135-2008ac-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, P₁ is written with a fully unspecified date in either or both halves to ensure that the property accepts them. DR₁ 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 DR₁ 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 P₁ = (DR₁ updated with a fully unspecified date in startDate)
2. VERIFY P₁ = (the value written)
3. WRITE P₁ = (DR₁ updated with a fully unspecified date in endDate)
4. VERIFY P₁ = (the value written)
5. WRITE P₁ = (DR₁ updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P₁ = (the value written)

Notes to Tester: if P₁ 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 X 8	Null	Inactive	Inactive	Inactive
PA_Index = P Y 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~~Y~~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, 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 Affected 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 where ACTIVE is specified, ACTIVE where INACTIVE is specified, and Minimum_On_Time where Minimum_Off_Time is specified.

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 Requirements O1 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 L1LifeTime from step 1)
 - IF (the IUT uses SubscribeCOV) THEN
 - 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) THEN
 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.

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

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

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

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

Purpose: To verify that the properties of ~~binary~~ objects that collectively track state changes (changes in Present\_Value) function as required. ~~If the Change\_Of\_State\_Count, Change\_Of\_State\_Time, and Time\_Of\_State\_Count\_Reset properties are not supported this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.~~

Test Concept: The Present\_Value of the ~~binary~~ object under test is changed. The Change\_Of\_State\_Count property is checked to verify that it has been incremented and the Change\_Of\_State\_Time property is checked to verify that it has been updated. The Change\_Of\_State\_Count is reset and Time\_Of\_State\_Count\_Reset is checked to verify that it has been updated appropriately.

Configuration Requirements: The object being tested shall be configured such that the Present\_Value and Change\_Of\_State\_Count properties are writable or another means of changing these properties shall be provided.

Test Steps:

```

1. READ PV = Present_Value
2. READ N = Change_of_State_Count
3. IF (PV = ACTIVE) THEN
 IF (Present_Value is writable) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
 ELSE
 MAKE (Present_Value = INACTIVE)
ELSE
 IF (Present_Value is writable) THEN
 WRITE Present_Value = ACTIVE
 VERIFY Present_Value = ACTIVE
 ELSE
 MAKE (Present_Value = ACTIVE)
4. VERIFY (Change_of_State_Count = N+1)
5. VERIFY (Time_Of_State_Count_Time ~= the current local date and time)
6. IF (Change_of_State_Count is writable) THEN
 WRITE Change_of_State_Count = 0
ELSE
 MAKE (Change_of_State_Count = 0)
7. VERIFY Time_Of_State_Count_Reset ~= (the current local date and time)
1. TRANSMIT ReadProperty Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value
2. RECEIVE ReadProperty ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value,
 'Property Value' = ACTIVE|INACTIVE
3. TRANSMIT ReadProperty Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count
4. RECEIVE ReadProperty ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count,
 'Property Value' = (any valid value, N)
5. IF (Present_Value is writable) THEN
 IF (the value returned in step 2 was ACTIVE) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
 ELSE
 WRITE Present_Value = ACTIVE
 VERIFY Present_Value = ACTIVE
 ELSE
 MAKE (Present_Value change to the opposite state)
6. TRANSMIT ReadProperty Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Date
7. RECEIVE ReadProperty ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local Date,
 'Property Value' = (the current local date, D)
8. TRANSMIT ReadProperty Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time
9. RECEIVE ReadProperty ACK,

```

```

—— 'Object Identifier' = (the IUT's Device object);
—— 'Property Identifier' = Local_Time;
—— 'Property Value' = (the current local time, TLOC)
10. WAIT Internal Processing Fail Time
11. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Change_Of_State_Time
12. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Change_Of_State_Time;
—— 'Property Value' = (a date and time such that the date = D and the time is approximately TLOC)
13. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Change_Of_State_Count
14. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Change_Of_State_Count;
—— 'Property Value' = N + 1
15. IF (Change_Of_State_Count is writable) THEN
—— WRITE Change_Of_State_Count = 0
—— VERIFY Change_Of_State_Count = 0
—— ELSE
—— MAKE (Change_Of_State_Count = 0)
16. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = (the IUT's Device object);
—— 'Property Identifier' = Local_Time
17. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = (the IUT's Device object);
—— 'Property Identifier' = Local_Time;
—— 'Property Value' = (the current local time, TLOC)
18. TRANSMIT ReadProperty Request,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Time_Of_State_Count_Reset
19. RECEIVE ReadProperty ACK,
—— 'Object Identifier' = (the object being tested);
—— 'Property Identifier' = Time_Of_State_Count_Reset;
—— 'Property Value' = (a date and time such that the date = D and the time is approximately TLOC)

```

### 7.3.1.9 Binary Object-Elapsed Active Time Tests

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

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 and Binary Lighting Output objects.~~

Test Concept: The Present\_Value or Feedback\_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 or Feedback\_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 or Feedback\_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* or *Feedback\_Value* if that is used for the calculation, and *Elapsed\_Active\_Time* properties are writable or another means of changing these properties shall be provided. Whether *Present\_Value* or *Feedback\_Value* is used as the indicator for the calculation of the *Elapsed\_Active\_Time* is a local matter.

Notes To Tester: It was intentional to specify that the alternative use of *Feedback\_Value* tracking specified in 135-2010ad-3 is allowed regardless of the *Protocol\_Revision* claimed by the implementation.

Test Steps:

1. IF (*Present\_Value* is writable) THEN  
     WRITE *Present\_Value* = INACTIVE  
     VERIFY *Present\_Value* = INACTIVE  
   ELSE  
     MAKE (*Present\_Value* = INACTIVE)
2. IF (*Feedback\_Value* is used for *Elapsed\_Active\_Time* tracking) THEN  
     WAIT (long enough for *Feedback\_Value* to reflect the *Present\_Value*)  
     VERIFY *Feedback\_Value* = INACTIVE  
   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<sub>ELAPSED</sub> in seconds~~)
3. READ *Elapsed\_Active\_Time* = initialElapsedTime
- verify that *Elapsed\_Active\_Time* does not change when the object is INACTIVE
4. ~~WAIT (1 minute)~~ WAIT (more than Internal Processing Fail Time + at least 1 second)
5. VERIFY *Elapsed\_Active\_Time* = initialElapsedTime
- verify that *Elapsed\_Active\_Time* correctly reflects the time the object is ACTIVE
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<sub>ELAPSED</sub> as step 3)
6. IF (*Present\_Value* is writable) THEN  
     WRITE *Present\_Value* = ACTIVE  
     VERIFY *Present\_Value* = ACTIVE  
   ELSE  
     MAKE (*Present\_Value* = ACTIVE)
7. IF (*Feedback\_Value* is used for *Elapsed\_Active\_Time* tracking) THEN  
     WAIT (long enough for *Feedback\_Value* to reflect the *Present\_Value*)  
     VERIFY *Feedback\_Value* = ACTIVE
8. READ initialTime = (the IUT's Device object) Local Time
9. WAIT (more than Internal Processing Fail Time + 30 seconds)
10. IF (*Present\_Value* is writable) THEN  
     WRITE *Present\_Value* = INACTIVE  
     VERIFY *Present\_Value* = INACTIVE  
   ELSE  
     MAKE (*Present\_Value* = INACTIVE)
11. IF (*Feedback\_Value* is used for *Elapsed\_Active\_Time* tracking) THEN  
     WAIT (long enough for *Feedback\_Value* to reflect the *Present\_Value*)

*VERIFY Feedback\_Value = INACTIVE*

12. *READ currentTime = (the IUT's Device object) Local\_Time*

13. *READ totalElapsedTime = Elapsed\_Active\_Time*

14. *CHECK (totalElapsedTime  $\sim$  (currentTime - initialTime) - initialElapsedTime)*

*-- verify ability to reset Elapsed\_Active\_Time, if it is writable*

15. *IF (Elapsed\_Active\_Time is writable) THEN*

*WRITE Elapsed\_Active\_Time = 0*

*READ currentDate = (the IUT's Device object) Local\_Date*

*READ currentTime = (the IUT's Device object) Local\_Time*

*VERIFY Time\_Of\_Active\_Time\_Reset  $\sim$  { currentDate, currentTime }*

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<sub>ELAPSED</sub> + 30) ≤ T ≤ (T<sub>ELAPSED</sub> + TimeX, where TimeX is the time between the beginning of step 7 and this step30 + 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<sub>LOC</sub>)~~

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\_ValueTime\_Of\_Active\_Time\_Reset,~~

~~'Property Value' = (a date and time such that the date = D and the time is approximately T<sub>LOC</sub>)~~

### 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 different 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 equal to 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  $\geq$  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  $\geq$  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),  
 '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

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 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)
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.

### 7.3.1.13.1 Limit\_Enable Test, LowLimitEnable

Reason for Change: The 'Event Type' is checked to be an out-of-range algorithm appropriate to the object type. .

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' = *(the algorithm appropriate to the object type i.e. OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_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' = *(the algorithm appropriate to the object type i.e. OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_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.

### 7.3.1.13.2 Limit\_Enable Test, HighLimitEnable

Reason for Change: The 'Event Type' is checked to be an out-of-range algorithm appropriate to the object type.

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' = (the algorithm appropriate to the object type i.e. OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_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' = *(the algorithm appropriate to the object type i.e. OUT\_OF\_RANGE, SIGNED\_OUT\_OF\_RANGE, UNSIGNED\_OUT\_OF\_RANGE, or DOUBLE\_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<sub>1</sub> 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<sub>1</sub> is used to designate the event-triggering property linked to O<sub>1</sub>.

Test Steps:

1. READ EMT = Event\_Message\_Texts
2. IF (Event\_Enable is (TRUE, ?, ?)) THEN
3. IF (X<sub>1</sub> is writable) THEN
  - WRITE X<sub>1</sub> = (a value that is offnormal)
- ELSE

- MAKE (X<sub>1</sub> 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<sub>1</sub>),  
     '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<sub>1</sub> is writable) THEN  
     WRITE X<sub>1</sub> = (a value that will result in a TO\_NORMAL transition)  
 ELSE  
     MAKE (X<sub>1</sub> 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<sub>1</sub>),  
     '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<sub>1</sub> 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<sub>1</sub>),  
     '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\_Algorithm\_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\_Algorithm\_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 Reliability\_Evaluation\_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 disables event detection.

Test Concept: Select an event generating object, O1, that supports event reporting. Verify that Event\_State is NORMAL and Acked\_Transitions, Event\_Time\_Stamps, and Event\_Message\_Texts are equal to their respective initial conditions, as mandated in the standard. If possible, make a condition exist that would cause a transition if event reporting were enabled and observe that no notification messages are transmitted.

Configuration Requirements: O1 is configured with Event\_Detection\_Enable set to FALSE. 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 = FALSE
2. VERIFY Event\_State = NORMAL
3. VERIFY Acked\_Transitions = (T,T,T)
4. VERIFY Event\_Time\_Stamps = [NO\_TS , NO\_TS , NO\_TS ]
5. IF the Event\_Message\_Texts property exists THEN  
     VERIFY Event\_Message\_Texts = [", ", ""]
6. MAKE (a condition exist which would cause O1 to transition, if Event\_Detection\_Enable were TRUE)
7. WAIT DELAY + Notification Fail Time
8. CHECK (that the IUT did not send any event notifications for O1)
9. VERIFY Event\_State = NORMAL
10. VERIFY Acked\_Transitions = (T,T,T)
11. VERIFY Event\_Time\_Stamps = [NO\_TS, NO\_TS, NO\_TS]
12. IF the Event\_Message\_Texts property exists THEN  
     VERIFY Event\_Message\_Texts = [", ", ""]

#### 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 = [", ", ""]

### 7.3.1.X16 Array Resizing Test using WritePropertyMultiple Service

Reason For Change: The existing test plan has a test case using WriteProperty service. We have added a new test case using WritePropertyMultiple service.

Purpose: To verify that resizable arrays are resized in accordance with the rules added in Protocol\_Revision 4.

Test Concept: The resizable array property P1 of object O1 is written with WritePropertyMultiple as a whole to set it to a non-zero size. It is then resized smaller and larger by writing the entire array. It is then resized smaller and larger by writing to element number zero. An attempt is made to increase it with an invalid write. After each operation, the array size and array contents are checked. Finally, if it can be resized to have zero elements, it is then written to size zero. If possible, all elements in the arrays should be distinguishable from each other and across WritePropertyMultiple operations.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = (array A1 of non-zero size N1)
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY P1 = (array A1), ARRAY INDEX = 0, (array size i.e. N1)

--Resize the array to make it smaller in size

4. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'Property Value' = (array A2 of non-zero size N2, where  $N2 \leq N1$ )
5. RECEIVE BACnet-SimpleACK-PDU
6. VERIFY P1 = (array A2), ARRAY INDEX = 0, (array size N2)

--Resize the array to make it larger in size

7. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'Property Value' = (array A3 of non-zero size N3, where  $N3 \geq N1$ ),
8. RECEIVE BACnet-SimpleACK-PDU
9. VERIFY P1 = (array A3), ARRAY INDEX = 0, (array size N3)

--Modify the existing content of element

10. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (array A4 of non-zero unsigned value N4, where  $N4 \leq N1$ ),
11. RECEIVE BACnet-SimpleACK-PDU
12. VERIFY P= (array A4), ARRAY INDEX = 0, (array size N4)

--Resize the array by writing the size of the array

13. TRANSMIT WritePropertyMultiple-Request  
     'Object Identifier' = O1,  
     'Property Identifier' = P1  
     'Property Value' = (N5, where  $N5 \geq N4$ ),  
     'Property Array Index' = 0,
14. RECEIVE BACnet-SimpleACK-PDU
15. VERIFY (array contains unchanged first N4 elements of the array written in step 10, plus N5-N4 additional elements, initialized to particular values for the array property being tested)
16. VERIFY P1, ARRAY INDEX = 0, (array size N5)

--Try to add the array element at Array Index which is greater than the size of the array

17. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (one array element),  
     'Property Array Index' = (N6, where  $N6 \geq N5$ ),
18. RECEIVE WritePropertyMultiple-Error  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Array Index' = N6
19. VERIFY (array is unchanged from step 15)

--Resize the array to size zero

20. IF (the array can be resized to have zero elements) THEN  
     TRANSMIT WritePropertyMultiple-Request,  
         'Object Identifier' = O1,  
         'Property Identifier' = P1,  
         'Property Value' = (empty array),  
     BACnet-SimpleACK-PDU
21. VERIFY P1 = (array is empty), ARRAY INDEX = 0, (array size is zero)

### 7.3.1.X18 Non-zero Writable State Count Test

Reason for Change: Additional behavior was specified in 135-2012az-1 and 135-2012az-2 when the Change\_of\_State\_Count property accepts writes of non-zero values.

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

Test Concept: The Change\_of\_State\_Count property is set with a non-zero value. The Time\_Of\_State\_Count\_Reset property is checked to verify that it has not been updated. The Time\_Of\_State\_Count\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall contain Change\_of\_State\_Count and Time\_Of\_State\_Count\_Reset properties, and its Change\_of\_State\_Count property must accept writes of non-zero values.

Test Steps:

1. READ TSCR = Time\_Of\_State\_Count\_Reset
2. WRITE Change\_of\_State\_Count = (a value > 0)
3. VERIFY Time\_Of\_State\_Count\_Reset = TSCR
4. WRITE Time\_Of\_State\_Count\_Reset = (T1: any valid value)
5. VERIFY Time\_Of\_State\_Count\_Reset = T1

### 7.3.1.X19 Non-zero Writable Elapsed Active Time Test

Reason for Change: Additional behavior was specified in 135-2012az-1 and 135-2012az-2 when the Elapsed\_Active\_Time property accepts writes of non-zero values.

Purpose: To verify that Time\_Of\_Active\_Time\_Reset is writable when Elapsed\_Active\_Time accepts writes of non-zero values and does not automatically change when Elapsed\_Active\_Time is written to a non-zero value.

Test Concept: The Present\_Value or Feedback\_Value is made INACTIVE and the Elapsed\_Active\_Time is set with a non-zero value. The Time\_Of\_Active\_Time\_Reset property is checked to verify that it has not been updated. The Time\_Of\_Active\_Time\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Elapsed\_Active\_Time and Time\_Of\_Active\_Time\_Reset properties are present, and in which the Elapsed\_Active\_Time property accepts writes of non-zero values.

Test Steps:

1. IF (Present\_Value is writable) THEN
  - WRITE Present\_Value = INACTIVE
  - VERIFY Present\_Value = INACTIVE
- ELSE
  - MAKE (Present\_Value = INACTIVE)
2. READ TATR = Time\_Of\_Active\_Time\_Reset
3. WRITE Elapsed\_Active\_Time = a supported non-zero value
4. VERIFY Time\_Of\_Active\_Time\_Reset = TATR
5. WRITE Time\_Of\_Active\_Time\_Reset = (T1: any valid value)
6. VERIFY Time\_Of\_Active\_Time\_Reset = T1

### 7.3.1.X20 Strike Count Tests

#### 7.3.1.X20.1 Non-zero Writable Strike Count Test

Reason for Change: Additional behavior was specified in 135-2012az-1 when the Strike\_Count property accepts writes of non-zero values.

Purpose: To verify that Time\_Of\_Count\_Reset is writable when Strike\_Count accepts writes of non-zero values and does not automatically change when Strike\_Count is written to a non-zero value.

Test Concept: The Strike\_Count property is set with a non-zero value. The Time\_Of\_Strike\_Count\_Reset property is checked to verify that it has not been updated. The Time\_Of\_Strike\_Count\_Reset property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which Strike\_Count and Time\_Of\_Strike\_Count\_Reset properties are present, and in which the Strike\_Count property accepts writes of non-zero values.

Test Steps:



1. READ TSCR = Time\_Of\_Strike\_Count\_Reset
2. WRITE Strike\_Count = (a value > 0)
3. VERIFY Time\_Of\_Strike\_Count\_Reset = TSCR
4. WRITE Time\_Of\_Strike\_Count\_Reset = (T1: any valid value)
5. VERIFY Time\_Of\_Strike\_Count\_Reset = T1

### 7.3.1.X20.2 Strike Count Test

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

Test Concept: The Present\_Value or Feedback\_Value of O1 can be used as the source S1 to increment Strike\_Count. S1 is transitioned from OFF to ON. The Strike\_Count property is checked to verify that it has been incremented. The Strike\_Count is reset and Time\_Of\_Strike\_Count\_Reset is checked to verify that it has been updated appropriately. Strike\_Count is set to a non-zero value and the Time\_Of\_Strike\_Count\_Reset is unchanged.

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

Test Steps:

1. C1 = Strike\_Count
2. MAKE (S1 transition OFF to ON)
3. VERIFY (Strike\_Count = C1 + 1)
4. IF (Strike\_Count is writable) THEN  
MAKE (Strike\_Count = 0)  
VERIFY (Time\_Of\_Strike\_Count\_Reset = current local time)
5. IF (Strike\_Count is writable to a non-zero value) THEN  
MAKE (Strike\_Count > 0)  
VERIFY (Time\_Of\_Strike\_Count\_Reset is unchanged)

### 7.3.1.X41 Blink Warn Tests

#### 7.3.1.X41.Y1 Blink-Warn WARN Command Test

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

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Active is FALSE.

|          | Binary Lighting Output object | Lighting Output object                           |
|----------|-------------------------------|--------------------------------------------------|
| PROP_REF | Present Value                 | Present Value or Lighting Command                |
| C1       | WARN                          | -1.0 if PROP_REF = Present Value, otherwise WARN |
| V1       | ON                            | >1.0                                             |

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Active = FALSE

4. WRITE PROP\_REF = C1, PRIORITY = PTY1
5. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
6. VERIFY Egress\_Active = FALSE
7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

### 7.3.1.X41.Y2 Blink-Warn WARN\_OFF Command Test

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

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN\_OFF command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority\_Array value at priority PTY1 after Egress\_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Active is FALSE, and Egress\_Time is a non-zero value.

|          | <b>Binary Lighting Output object</b> | <b>Lighting Output object</b>                        |
|----------|--------------------------------------|------------------------------------------------------|
| PROP_REF | Present Value                        | Present Value or Lighting Command                    |
| C1       | WARN_OFF                             | -3.0 if PROP_REF = Present Value, otherwise WARN_OFF |
| V1       | ON                                   | >1.0                                                 |
| V2       | OFF                                  | 0.0                                                  |

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. WHILE (Egress\_Active = TRUE)  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY Egress\_Time  $\approx$  (T1 – T2) +/- **Internal Processing Fail Time**
11. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1

### 7.3.1.X41.Y3 Blink-Warn WARN\_RELINQUISH Command Test

Purpose: To verify the correct operation of the blink-warn WARN\_RELINQUISH commands.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN\_RELINQUISH command by writing C1 to PROP\_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority\_Array value at priority PTY1 after Egress\_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, and Egress\_Active is FALSE.

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

Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. WHILE (Egress\_Active = TRUE)  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. T2 = current local time
10. VERIFY Egress\_Time  $\approx$  (T1 - T2) +/- **Internal Processing Fail Time**
11. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1

### 7.3.1.X41.Y4 Blink-Warn STOP Command Test

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

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority\_Array at PTY1 has a value V1, Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, and Egress\_Active is FALSE.

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

## Test Steps:

1. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. **BEFORE Internal Processing Fail Time**  
CHECK (blink-warn occurred)
8. VERIFY Egress\_Active = TRUE
9. WAIT less than Egress\_Time  
WRITE PROP\_REF = STOP, PRIORITY = PTY1
10. T2 = current local time
11. **WAIT Internal Processing Fail Time**
12. VERIFY Egress\_Active = FALSE
13. WAIT Egress\_Time – (T2 – T1) + **Internal Processing Fail Time**
14. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

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

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

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

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

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

## Test Steps:

-- Test for the specified priority is not the highest active priority

1. VERIFY Blink\_Warn\_Enable = TRUE
2. WRITE Present\_Value = V1, PRIORITY = PTY1
3. VERIFY Egress\_Active = FALSE
4. WRITE Present\_Value = V2, PRIORITY = PTY2
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. **WAIT Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
7. VERIFY Egress\_Active = FALSE
8. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
9. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is either OFF or 0.0

10. WRITE Present\_Value = V3, PRIORITY = PTY1
  11. WRITE PROP\_REF = C1, PRIORITY = PTY1
  12. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
  13. VERIFY Egress\_Active = FALSE
  14. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
  15. WRITE Present\_Value = V1, PRIORITY = PTY1
- Test for Blink\_Warn\_Enable is FALSE
16. IF (Blink\_Warn\_Enable is writable) THEN  
WRITE Blink\_Warn\_Enable = FALSE  
WRITE PROP\_REF = C1, PRIORITY = PTY1  
WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)  
VERIFY Egress\_Active = FALSE  
VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1

### 7.3.1.X41.Y6 Blink-Warn WARN\_OFF Command Failure Test

Purpose: To verify blink-warn WARN\_OFF command does not occur when the specified priority is not the highest active priority, the Present\_Value is either 0.0 or OFF, or Blink\_Warn\_Enable is FALSE.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value and Egress\_Active is FALSE.

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

Test Steps:

- Test for the specified priority is not the highest active priority
1. VERIFY Blink\_Warn\_Enable = TRUE
  2. VERIFY Egress\_Time > 0
  3. WRITE Present\_Value = V1, PRIORITY = PTY1
  4. VERIFY Egress\_Active = FALSE
  5. WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
  6. WRITE PROP\_REF = C1, PRIORITY = PTY1
  7. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
  8. VERIFY Egress\_Active = FALSE
  9. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
  10. WRITE Present\_Value = V1, PRIORITY = PTY1
- Test for the Present\_Value is OFF or 0.0
11. WRITE Present\_Value = V3, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
  12. WRITE PROP\_REF = C1, PRIORITY = PTY1
  13. WAIT **Internal Processing Fail Time**

CHECK (blink-warn did not occur)

14. VERIFY Egress\_Active = FALSE
15. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1
16. WRITE Present\_Value = NULL, PRIORITY = PTY2
17. WRITE Present\_Value = V1, PRIORITY = PTY1

-- Test for Blink\_Warn\_Enable is FALSE

18. IF (Blink\_Warn\_Enable is writable) THEN
  - WRITE Blink\_Warn\_Enable = FALSE
  - WRITE PROP\_REF = C1, PRIORITY = PTY1
  - WAIT **Internal Processing Fail Time**
  - CHECK (blink-warn did not occur)
  - VERIFY Egress\_Active = FALSE
  - VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1

### 7.3.1.X41.Y7 Blink-Warn WARN\_RELINQUISH Command Failure Test

Purpose: To verify blink-warn WARN\_RELINQUISH command does not occur when the specified priority is not the highest active priority, the value at the specified priority is V0, the value of the next highest non-NULL priority, including Relinquish\_Default, is V1, or Blink\_Warn\_Enable is FALSE.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, Egress\_Active is FALSE.

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

Test Steps:

-- Test for the specified priority is not the highest active priority

1. VERIFY Blink\_Warn\_Enable = TRUE
2. VERIFY Egress\_Time > 0
3. WRITE Present\_Value = V1, PRIORITY = PTY1
4. VERIFY Egress\_Active = FALSE
5. WRITE Present\_Value = V1, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP\_REF = C1, PRIORITY = PTY1
7. WAIT **Internal Processing Fail Time**
  - CHECK (blink-warn did not occur)
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the value at the specified priority is OFF or 0.0

11. WRITE Present\_Value = V2 PRIORITY = PTY1
12. WRITE PROP\_REF = C1, PRIORITY = PTY1
13. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
14. VERIFY Egress\_Active = FALSE
15. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
  
- Test for the value at the specified priority is NULL
16. WRITE Present\_Value = NULL, PRIORITY = PTY1
17. WRITE PROP\_REF = C1, PRIORITY = PTY1
18. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
19. VERIFY Egress\_Active = FALSE
20. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
  
- Test for the value of the next highest non-NULL priority is ON or > 1.0
21. WRITE Present\_Value = V1 PRIORITY = PTY1
22. WRITE Present\_Value = V1, PRIORITY = PTY3, a value numerically greater than PTY1
23. WRITE PROP\_REF = C1, PRIORITY = PTY1
24. WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)
25. VERIFY Egress\_Active = FALSE
26. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
27. WRITE Present\_Value = NULL, PRIORITY = PTY3
  
- Test for the value of Relinquish\_Default is ON or > 1.0
28. IF (Relinquish\_Default is writable) THEN  
WRITE Present\_Value = V1, PRIORITY = PTY1  
WRITE Relinquish\_Default = V1  
WRITE PROP\_REF = C1, PRIORITY = PTY1  
WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)  
VERIFY Egress\_Active = FALSE  
VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1  
WRITE Relinquish\_Default = V2
  
- Test for Blink\_Warn\_Enable is FALSE
29. IF (Blink\_Warn\_Enable is writable) THEN  
WRITE Present\_Value = V1, PRIORITY = PTY1  
WRITE Blink\_Warn\_Enable = FALSE  
WRITE PROP\_REF = C1, PRIORITY = PTY1  
WAIT **Internal Processing Fail Time**  
CHECK (blink-warn did not occur)  
VERIFY Egress\_Active = FALSE  
VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1

### 7.3.1.X41.Y8 Blink-Warn WARN\_OFF Command Halted Test

Purpose: To verify blink-warn WARN\_OFF execution is halted when a higher priority entry is written or the Present\_Value at the specified priority is changed.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value and Egress\_Active is FALSE.

|          | Binary Lighting Output object | Lighting Output object                               |
|----------|-------------------------------|------------------------------------------------------|
| PROP_REF | Present_Value                 | Present_Value or Lighting_Command                    |
| C1       | WARN_OFF                      | -3.0 if PROP_REF = Present_Value, otherwise WARN_OFF |
| V1 to V3 | ON                            | >1.0                                                 |
| V4       | OFF                           | 0.0                                                  |

Test Steps:

-- Test for a higher priority entry is written to a non NULL value

1. WRITE Present\_Value = V1, PRIORITY = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
7. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = V4, ARRAY INDEX = PTY1
10. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the Present\_Value at the specified property is changed

11. WRITE Present\_Value = V1, PRIORITY = PTY1
12. VERIFY Blink\_Warn\_Enable = TRUE
13. VERIFY Egress\_Time > 0
14. VERIFY Egress\_Active = FALSE
15. WRITE PROP\_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
17. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V3, PRIORITY = PTY1
18. VERIFY Egress\_Active = FALSE
19. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1

### 7.3.1.X41.Y9 Blink-Warn WARN\_RELINQUISH Command Halted Test

Purpose: To verify blink-warn WARN\_RELINQUISH execution is halted when a higher priority entry is written or the Present\_Value at the specified priority is changed.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority\_Array numerically less than PTY1 have a value of NULL and at least one slot numerically greater than PTY1 or Relinquish\_Default shall have a value of V1 and all



other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink\_Warn\_Enable is TRUE, Egress\_Time is a non-zero value, Egress\_Active is FALSE.

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

#### Test Steps:

-- Test for a higher priority entry is written to a non NULL value

1. WRITE Present\_Value = V2, PRIORITY = PTY1
2. VERIFY Blink\_Warn\_Enable = TRUE
3. VERIFY Egress\_Time > 0
4. VERIFY Egress\_Active = FALSE
5. WRITE PROP\_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
7. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress\_Active = FALSE
9. VERIFY Priority\_Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present\_Value = NULL, PRIORITY = PTY2

-- Test for the Present\_Value at the specified property is changed

11. WRITE Present\_Value = V2, PRIORITY = PTY1
12. VERIFY Blink\_Warn\_Enable = TRUE
13. VERIFY Egress\_Time > 0
14. VERIFY Egress\_Active = FALSE
15. WRITE PROP\_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**  
CHECK (blink-warn occurred)
17. BEFORE Egress\_Active = FALSE  
WRITE Present\_Value = V3, PRIORITY = PTY1
18. VERIFY Egress\_Active = FALSE
19. VERIFY Priority\_Array = V3, ARRAY INDEX = PTY1

### 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~~ verifies 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
73. REPEAT X = (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  $\geq 10$ .

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  $\geq 10$ ) 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: IC135-2012-18 ruled that the increment shall not be in the Active\_COV\_Subscriptions property value if the property is not numeric; present if a valid Increment was provided in the subscription; and optionally present otherwise

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.

*Test Concept: INC<sub>1</sub>, INC<sub>2</sub>, and INC<sub>3</sub> are each not present if the property is not numeric; present if a valid Increment was provided in the subscription; and optionally present otherwise.*

Configuration Requirements: In this test, the tester shall choose three standard objects, O<sub>1</sub>, O<sub>2</sub>, and O<sub>3</sub>, for which the device supports SubscribeCOV. O<sub>1</sub>, O<sub>2</sub>, and O<sub>3</sub> are not required to refer to different objects. The tester shall also choose three non-zero unique process identifiers, P<sub>1</sub>, P<sub>2</sub>, and P<sub>3</sub>, and three non-zero lifetimes L<sub>1</sub>, L<sub>2</sub>, and L<sub>3</sub>. Lifetime L<sub>1</sub> shall be long enough to allow the initial part of the test to run through to step 14. Lifetimes L<sub>2</sub> and L<sub>3</sub> 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<sub>1</sub>,  
     'Monitored Object Identifier' = O<sub>1</sub>,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L<sub>1</sub>
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = P<sub>1</sub>,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O<sub>1</sub>,  
     'Time Remaining' = (a value approximately equal to L<sub>1</sub>),  
     'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF P<sub>1</sub> is numeric THEN  
     VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value }, TRUE, (a value less than L<sub>1</sub>),  
     (INC<sub>1</sub> : not present or a valid Increment if the property is REAL) } }  
     ELSE  
     VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>, { O<sub>1</sub>, Present\_Value }, TRUE, (a value less than L<sub>1</sub>), (INC<sub>1</sub>: not present) } } }
6. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = P<sub>2</sub>,  
     'Monitored Object Identifier' = O<sub>2</sub>,  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = L<sub>2</sub>
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = P<sub>2</sub>,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O<sub>2</sub>,  
     'Time Remaining' = (a value approximately equal to L<sub>2</sub>),  
     'List of Values' = (values appropriate to the object type of the monitored object)
9. VERIFY Active\_COV\_Subscriptions = { { {TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than L<sub>1</sub>),  
     INC<sub>1</sub>(a valid Increment if the property is REAL) },  
     { {TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than L<sub>2</sub>),  
     (INC<sub>2</sub>: not present if the property is not numeric; present  
     if a valid Increment was provided in the subscription;  
     optionally present otherwise if the property is REAL) } } }
10. TRANSMIT SubscribeCOV-Request,

- 'Subscriber Process Identifier' = P<sub>3</sub>,  
 'Monitored Object Identifier' = O<sub>3</sub>,  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = L<sub>3</sub>
11. RECEIVE BACnet-SimpleACK-PDU
12. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = P<sub>3</sub>,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O<sub>3</sub>,  
 'Time Remaining' = (a value approximately equal to L<sub>3</sub>),  
 'List of Values' = (values appropriate to the object type of the monitored object)
13. IF P<sub>3</sub> is numeric THEN  
 VERIFY Active\_COV\_Subscriptions = { {{TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than L<sub>1</sub>),  
 INC<sub>1</sub>(a valid Increment if the property is REAL)},  
 {{TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than L<sub>2</sub>),  
 INC<sub>2</sub>(a valid Increment if the property is REAL)},  
 {{TD, P<sub>3</sub>}, {O<sub>3</sub>, Present\_Value}, FALSE, (a value less than L<sub>3</sub>),  
 INC<sub>3</sub>: not present or (a valid Increment if the property is REAL)}}  
 ELSE  
 VERIFY Active\_COV\_Subscriptions = { {{TD, P<sub>1</sub>}, {O<sub>1</sub>, Present\_Value}, TRUE, (a value less than  
 L<sub>1</sub>), INC<sub>1</sub>},  
 {{TD, P<sub>2</sub>}, {O<sub>2</sub>, Present\_Value}, FALSE, (a value less than  
 L<sub>2</sub>), INC<sub>2</sub>},  
 {{TD, P<sub>3</sub>}, {O<sub>3</sub>, Present\_Value}, FALSE, (a value less Than  
 L<sub>3</sub>), (INC<sub>3</sub>: not present)}} }
14. WAIT L<sub>1</sub> + the IUT's timer granularity
15. VERIFY Active\_COV\_Subscriptions = { {{TD, P 2 }, {O 2 , Present\_Value}, FALSE, (a value less than L 2 ),  
 INC<sub>2</sub> (a valid Increment if the property is REAL)},  
 {{TD, P 3 }, {O 3 , Present\_Value}, FALSE, (a value less than L 3 ),  
 INC<sub>3</sub>(a valid Increment if the property is REAL)}} }
16. TRANSMIT SubscribeCOV-Request,  
 'Subscriber Process Identifier' = P<sub>3</sub> ,  
 'Monitored Object Identifier' = O<sub>3</sub>
17. RECEIVE BACnet-SimpleACK-PDU
18. VERIFY Active\_COV\_Subscriptions = { {{TD, P 2 }, {O 2 , Present\_Value}, FALSE, (a value less than L 2 ),  
 INC<sub>2</sub>(a valid Increment if the property is REAL) } }
19. TRANSMIT SubscribeCOV-Request,  
 'Subscriber Process Identifier' = P<sub>2</sub> ,  
 'Monitored Object Identifier' = O<sub>2</sub>
20. RECEIVE BACnet-SimpleACK-PDU
21. VERIFY Active\_COV\_Subscriptions = { }

### 7.3.2.23.10.3.7 Revision 4 Calendar Entry WeekNDay Day Of Week Test

Reason for Change: Added clarifying text to table 7-16.1.

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date matching a WeekNDay's DayOfWeek field in an Exception\_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-10. The value of the Present\_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception\_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the day of the week. The criteria for the dates used in the test are given in Table 7-10. The local date and time shall be set such that the Present\_Value property has a value other than V<sub>1</sub>.

**Table 7-16.1.** Criteria for Calendar Entry WeekNDay Day of Week Test Dates and Values

| Date           | Criteria                                                                                                                                                                                                                                                                                      | Value                                                      |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| D <sub>1</sub> | 1. Date occurs during Effective_Period,<br>2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay,<br>2B. calendarEntry: WeekNDay specifies only DayOfWeek,<br>2C. Date falls on the specified day of the week, and<br>2D. Higher eventPriority than any coincident BACnetSpecialEvents. | V <sub>1</sub>                                             |
| D <sub>2</sub> | 1. Date occurs during Effective_Period,<br>2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay,<br>2B. calendarEntry: WeekNDay specifies only DayOfWeek, and<br>2C. Date does not fall on the specified day of the week.                                                              | V <sub>2</sub><br><i>(different from any other values)</i> |

Test Steps:

1. VERIFY Present\_Value = (any value other than V<sub>1</sub>)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>1</sub>) |  
MAKE (the local date and time = D<sub>1</sub>)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present\_Value = V<sub>1</sub>
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
MAKE (the local date and time = D<sub>2</sub>)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present\_Value = (any value other than V<sub>2</sub>)

### 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 correctly 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. IF Segmentation\_Supported == SEGMENTED\_TRANSMIT or Segmentation\_Supported == SEGMENTED\_NONE  
THEN  
VERIFY Max\_Segments\_Accepted = 1  
ELSE  
VERIFY (Max\_Segments\_Accepted > 1)

#### 7.3.2.13 Global Group Object Tests

##### 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.

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. IF (Reliability is present) THEN  
    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.

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 N1 that 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. IF (Reliability is present) THEN  
    VERIFY Reliability = COMMUNICATION\_FAILURE
5. 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  $\neq$  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 attempt to produce unsubscribed UnconfirmedCOVNotifications. Repeat with COVU\_Recipients containing an empty list.

Configuration Requirements: At the start of the test, O1 shall be configured with a non-zero COVU\_Period and a non-empty COV\_Recipient property.

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. WRITE (COVU\_Period = 0)
4. MAKE (O1 a condition that would normally cause the IUT to issue an unsubscribed UnconfirmedCOVNotification)
5. **WAIT Notification Fail Time** times 2
6. CHECK (that O1 has not transmitted an UnconfirmedCOVNotification-Request)
7. WRITE (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. WRITE (COVU\_Recipient an empty list)
11. MAKE (O1 a condition that would normally cause the IUT to issue an unsubscribed UnconfirmedCOVNotification)
12. **WAIT Notification Fail Time** times 2
13. CHECK (that O1 has not transmitted an UnconfirmedCOVNotification-Request)

### 7.3.2.15 Life Safety Point Object Tests

#### 7.3.2.15.X5 Writable Tracking\_Value

BACnet Reference Clauses: 12.15.4, 12.15.5, 12.16.4 and 12.16.5

Purpose: This test case verifies that Tracking\_Value is writable when Out\_Of\_Service is TRUE.

Test Concept: It verifies the interrelationship between the Tracking\_Value, Status Flags and Present\_Value 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. This test applies to Life Safety Zone and Life Safety point object.

The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. MAKE (Out\_Of\_Service TRUE)

2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, FALSE, ?, TRUE)
4. MAKE (Event\_State = Normal)
5. VERIFY Event\_State = Normal
6. WRITE Tracking\_Value = (X: any value that corresponds to an Event\_State of NORMAL)
7. VERIFY Tracking\_Value = X
8. VERIFY Present\_Value = X

#### 7.3.2.15.X6 Supports Writable Mode Property

BACnet Reference Clauses: 12.15.12, 12.15.13, 12.16.12 and 12.16.13

Purpose: To verify that the Mode property takes one of the values found in the Accepted\_Modes property.

Test Concept: It verifies the interrelationship between the Mode, and Accepted\_Modes properties. This test applies to Life Safety Zone and Life Safety point object. The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. READ AM = Accepted\_Modes
2. TRANSMIT WriteProperty-Request  
     'Object Identifier' = (the object being tested),  
     'Property Identifier' = Mode,  
     'Property Value' = (X: Any valid value from list of AM)
3. RECEIVE SimpleACK-PDU
4. VERIFY Mode = X
5. TRANSMIT WriteProperty-Request  
     'Object Identifier' = (the object being tested),  
     'Property Identifier' = Mode,  
     'Property Value' = (X: Any invalid value, which is not present in AM)
6. RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 7.3.2.15.X7 Support Operation\_Expected Property

BACnet Reference Clauses: 12.15.24, and 12.16.24

Purpose: To verify that the Operation\_Expected property takes on the value of ConfirmedEventNotification-Request.

Test Concept: It verifies the interrelationship between the Operation\_Expected property, and ConfirmedEventNotification-Request. This test applies to Life Safety Zone and Life Safety point object. The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. MAKE (the IUT send an ConfirmedEventNotification)
2. RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = TD,

'Event Object Identifier' = (any Life-Safety object),  
 'Time Stamp' = (the current local time),  
 'Notification Class' = (any valid notification class),  
 'Priority' = (any valid priority),  
 'Event Type' = CHANGE-OF-LIFE-SAFETY,  
 'Message Text' = (any character string),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = (any non-normal state appropriate to the event type),  
 'Event Values' = (New State: (Any Valid State), New-Mode: (Any Valid Mode),  
 Status-Flag: (TRUE, FALSE, ?, ?),  
 Operation\_Expected: (X: Any Valid operation))

3. VERIFY Operation\_Expected = X ( operation expected in the step 2)

### 7.3.2.15.X8 Support Writable Member\_Of Property

BACnet Reference Clauses: 12.15.29, and 12.16.29

Purpose: To verify that the Member\_Of property takes only supported values of the life safety objects within the IUT.

Test Concept: If the property is writable and is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result (-), if the property is not writable and if the value of the property cannot be changed by other means, then this test shall be omitted. The IUT will select one instance of each appropriate object type, O1 and test it as described.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (O1),  
     'Property Identifier' = Member\_Of  
     'Property Value' = (X: any valid life safety zone object reference)
2. RECEIVE Simple-ACK-PDU,
3. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (O1),  
     'Property Identifier' = Member\_Of
4. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (the object being tested),  
     'Property Identifier' = Member\_Of  
     'Property Value' = X

### 7.3.2.15.X9 Silenced Property Test

Purpose: This test verifies the behavior of Silenced property.

Test Concept: Verify the interrelationship between the Silenced property and any audible or visual indication that has been silenced by the receipt of a LifeSafetyOperation service request or a local process. If the Silenced property of the object under test is unchanging by means of a LifeSafetyOperation service requests, because none of the silencing operations are supported, then this test shall be omitted. This test applies to Life Safety Zone and Life Safety Point object.

The IUT will select one instance of each appropriate object type and test it as described.

Test Steps:

1. InitialSilencedState = READ Silenced
2. TRANSMIT LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid identifier),  
     'Requesting Source' = (any valid character string),  
     'Request' = (any supported LifeSafetyOperation request transmitted to silence the sounder/strobe),  
     'Object Identifier' = (the selected object)
3. RECEIVE BACnet-SimpleACK-PDU
4. CHECK (Sounder/Strobe inactive)
5. ResultingSilencedState = READ Silenced
6. CHECK (the ResultingSilencedState is equal to the InitialSilencedState, modified by the LifeSafetyOperation request transmitted)

### 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<sub>1</sub>, shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, N<sub>2</sub>, 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<sub>1</sub>" and "X<sub>2</sub>" are used to designate the event-triggering property linked to Notification objects "N<sub>1</sub>" and "N<sub>2</sub>" respectively.

Test Steps:

1. VERIFY (the event-generating object linked to N<sub>1</sub>), Event\_State = NORMAL
2. VERIFY (the event-generating object linked to N<sub>2</sub>), Event\_State = NORMAL
3. WAIT (Time\_Delay + **Notification Fail Time**)
4. IF (X<sub>1</sub> is writable) THEN  
     WRITE X<sub>1</sub> = (a value that is OFFNORMAL)  
   ELSE  
     MAKE (X<sub>1</sub> 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<sub>1</sub>),  
     '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<sub>2</sub> is writable) THEN  
     WRITE X<sub>2</sub> = (a value that is OFFNORMAL)  
   ELSE  
     MAKE (X<sub>2</sub> 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<sub>2</sub>),  
     '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<sub>last</sub> 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<sub>last</sub> 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<sub>1</sub> represents a time between 00:00 and the time of the first scheduled write operation in the BACnetDailySchedule. V<sub>last</sub> 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  $\geq 4$ ) 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  $\geq 4$ ) 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:  $D_1, D_2, D_3, \dots, D_{N-1}, D_N$ . 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  | -     | -     |



## BACnet Testing Laboratories - Specified Tests

|                |                |                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 3              | -              | -              | V <sub>3</sub> | -              | -              | -              | -              | NULL           | -              |
| 4              | -              | V <sub>2</sub> | -              | -              | -              | -              | -              | -              | NULL           |
| 5              | V <sub>1</sub> | -              | -              | -              | -              | -              | -              | -              | -              |
| Present Value: | V <sub>1</sub> | V <sub>2</sub> | V <sub>3</sub> | V <sub>4</sub> | V <sub>5</sub> | V <sub>4</sub> | V <sub>3</sub> | V <sub>2</sub> | V <sub>1</sub> |

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.*

### 7.3.2.23.11.1 Internally Written Datatypes Test, non-NULL values

Reason for Change: To ensure that datatype-specific testing is conducted even when there is no Schedule that can be made to reference another property. The effect should be observed in the Present\_Value property of the Schedule Object instead.

BACnet Reference Clauses: 12.24, 12.24.10

Purpose: This test verifies that the Schedule object within the IUT writes to properties in the same device for the non-NULL datatype being tested.

Test Concept: Two Date/Time, values, D1 and D2, are chosen by the TD based on the criteria in Table 7-17 such that D1 is sufficiently different from, and later than, the current time to cause a Schedule evaluation when the time is changed to D1, and such that setting the time to D2 (later than D1) from D1 will cause a Schedule evaluation that will cause it to write value V2. These values may be chosen based on the Schedule object's existing configuration, or the Schedule object, S, may be configured with such values.

Configuration Requirements: The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-17 can be configured with uniquely scheduled values. The Schedule object shall be configured with a List\_Of\_Object\_Property\_References, including at least one reference to a writable property within the device, *if possible*. *Step 4 and step 8 would REPEAT zero times, if the referenced property is empty or not present. If the IUT cannot be configured to these requirements, then this test shall be omitted. Other properties in the Schedule object shall be consistent in both datatypes and values in a manner permitting this test to be executed.*

**Table 7-17.** Criteria for Test Date and Times

| Date and Time: | Value:                                         |
|----------------|------------------------------------------------|
| D <sub>1</sub> | V <sub>1</sub>                                 |
| D <sub>2</sub> | V <sub>2</sub> different from V <sub>1</sub> . |

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>1</sub>)  
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D<sub>1</sub>)  
| MAKE (the local date and time = D<sub>1</sub>)
2. WAIT (**Schedule Evaluation Fail Time**)
3. VERIFY S, Present\_Value = V<sub>1</sub>
4. REPEAT P = (writable property in List\_Of\_Property\_References)  
VERIFY P = V<sub>1</sub>
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>)  
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D<sub>2</sub>)  
| MAKE (the local date and time = D<sub>2</sub>)
6. WAIT(**Schedule Evaluation Fail Time**)
7. VERIFY S, Present\_Value = V<sub>2</sub>
8. REPEAT P = (writable property in List\_Of\_Property\_References)  
VERIFY P = V<sub>2</sub>

Note to Tester: In the context of this test definition, writable means that the Schedule object is capable of modifying the property. It does not necessarily indicate that the property is modifiable via BACnet services.

### 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.

#### 7.3.2.24.3 Stop\_Time Test

Reason For Change: Errata change to correct the Test Concept to reference correct property.

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

BACnet Reference Clause: 12.25.7.

Purpose: To verify that logging is disabled at the time specified by Stop\_Time.

Test Concept: The logging object is configured to acquire data by each means available to the implementation. The test is begun at some time prior to the time specified in *Stop\_Time*~~Start\_Time~~ and collection of records is confirmed. Non-collection of records after the time specified by Stop\_Time is then confirmed.

Configuration Requirements: Stop\_Time shall be configured with a date and time such that steps 1 through 9 will be concluded before that time. Start\_Time, if present shall be configured with date and time preceding the initiation of the test. Stop\_When\_Full, if configurable, shall be set to FALSE.

Test Steps:

1. WRITE Enable = FALSE
2. WAIT **Internal Processing Fail Time**
3. WRITE Record\_Count = 0

4. WRITE Enable = TRUE
5. READ X = Total\_Record\_Count
6. WAIT **Internal Processing Fail Time**
7. MAKE (IUT collect another record)
8. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
9. VERIFY Total\_Record\_Count > X
10. WHILE (IUT clock is earlier than Stop\_Time) DO {}
11. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
12. READ X = Total\_Record\_Count
13. MAKE (IUT collect another record)
14. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
15. VERIFY Total\_Record\_Count = X

Note to Tester: For each MAKE (IUT collect another record), perform the following actions:

```

IF (Event Log Object) THEN
 MAKE (Event Log Object collect another record)
ELSE
 IF (COV subscription in use) THEN
 MAKE (monitored value change sufficient to generate another record)
 ELSE IF (interval or period logging is in use) THEN
 WAIT (Log_Interval)
 ELSE
 MAKE (Trend Log or Trend Log Multiple Object collect another record)

```

#### 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' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Index' = 1,~~  
~~'Count' = 1~~
5. ~~RECEIVE ReadRange-Ack~~  
~~'Object Identifier' = 01,~~  
~~'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' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Index' = 1,~~  
~~'Count' = 2~~
9. ~~RECEIVE ReadRangeAck~~  
~~'Object Identifier' = 01,~~  
~~'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' = 01,~~  
~~'Property Identifier' = Log\_Buffer,~~  
~~'Reference Time' = (2154 12 31, 23:59:59.99),~~  
~~'Count' = 1~~
11. ~~RECEIVE ReadRangeAck~~  
~~'Object Identifier' = 01,~~  
~~'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,  $O_I$  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 34)*
  - TRANSMIT ReadRange-Request*
    - 'Object Identifier' =  $O_I$ ,
    - 'Property Identifier' = Log\_Buffer,
    - 'Reference Index' =  $N-4+X$ ,
    - 'Count' = 1
  - RECEIVE ~~ReadRangeAck~~ReadRange-ACK*
    - 'Object Identifier' =  $O_I$ ,

'Property Identifier' = Log\_Buffer,  
 'Result Flags' = (~~True~~, ~~True~~, False),  
 'Item Count' = 1,  
 'Item Data' =  $\{$ (one data record storing the timestamp in TS[X]) $\}$

14. TRANSMIT ReadRange-Request

'Object Identifier' = O1,  
 'Property Identifier' = Log\_Buffer,  
 'Reference Index' = N,  
 'Count' = 1

RECEIVE ReadRange-ACK

'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]  $\sim$  10 seconds )~~

~~15. CHECK( TS[4] - TS[3]  $\sim$  20 seconds )~~

~~16. CHECK( TS[5] - TS[4]  $\sim$  40 seconds )~~

~~17. CHECK( TS[6] - TS[5]  $\sim$  30 seconds )~~

15. CHECK( TS[2] - TS[1]  $\sim$  20 seconds )

16. CHECK( TS[3] - TS[2]  $\sim$  40 seconds )

17. CHECK( TS[4] - TS[3]  $\sim$  30 seconds )

### 7.3.2.24.X8 Clock-Aligned Logging

Purpose: To verify that logged trend records have timestamps aligned to that interval, when Align\_Intervals is TRUE and Log\_Interval is a factor of (divides without remainder) a day.

Test Concept: For this test, select two evenly divisible factors. Write each to Log\_Interval in the test. Trend records are logged, and checked that those are aligned to the Log\_Interval. This is done twice to ensure that different interval frequency behavior is verified. This test does not employ Log\_Interval values which are not one of the evenly divisible factors.

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 in order that it occurs after the end of the test. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Interval\_Offset is set to zero. Align\_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. Logging\_Mode is POLLED. X1 and X2 are each a value which the IUT supports for which the standard mandates the alignment behavior.

Test Steps:

1. CHECK (Log\_Buffer contains 1 entry, and it is the buffer-purged event)
2. WRITE Log\_Interval = X1
3. WRITE Enable = TRUE
4. MAKE (logging object collect at least 2 records)
5. WRITE Enable = FALSE
6. CHECK (Log\_Buffer contains at least 5 entries, and at least two data records)
7. CHECK (that the timestamp of each data record, since the Log\_Interval was written, is a multiple of X1)
8. WRITE Log\_Interval = (X2, any value which requires alignment behavior, that was not already chosen)
9. WRITE Enable = TRUE
10. MAKE (logging object collect at least 2 more records)
11. WRITE Enable = FALSE
12. CHECK (Log\_Buffer has collected two or more additional data records and two or more log-status records)
13. CHECK (that the timestamp are multiples of X2 for all data records collected, since the write with X2)

Notes to Tester: The values for Log\_Interval which require alignment are those for which the standard mandates the alignment behavior, where 8,640,000 modulo Log\_Interval is zero.



### 7.3.2.24.X9 Logging Interval\_Offset

Purpose: To verify that timestamps abide by the Interval\_Offset.

Test Concept: Log\_Interval is set to a value which the IUT supports which is a factor of (divides without remainder) a day and which is greater than 3 seconds.

Interval\_Offset is first set to a non-zero value less than Log\_Interval. After logging some records, their timestamps are checked. The logging is stopped. Interval\_Offset is set to a value which the IUT supports greater than Log\_Interval, logging is re-enabled, and the timestamps again are checked.

Configuration Requirements: Align\_Intervals is set to TRUE. The Log\_DeviceObjectProperty property in a Trend Log or in a Trend Log Multiple, is configured to the property or properties monitored. 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 in order that it occur after the end of the test. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Align\_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. If the Interval\_Offset cannot be set to a value which the IUT supports greater than Log\_Interval, then steps 11 through the end of this test are skipped. Logging\_Mode is POLLED. An evenly divisible value is a value for which the standard mandates the alignment behavior.

Test Steps:

1. WRITE Record\_Count = 0
2. CHECK (Log\_Buffer contains 1 entry, and it is the buffer-purged event)
3. WRITE Log\_Interval = (any evenly divisible value greater than 3 seconds)
4. WRITE Interval\_Offset = (any value, between 2 seconds and Log\_Interval - 1 seconds)
5. WRITE Enable = TRUE
6. MAKE (logging object collect at least 2 records)
7. WRITE Enable = FALSE
8. CHECK (Log\_Buffer contains two or more data records and at least three log-status)
9. CHECK (the timestamp for the data records have a fixed offset, determined by Log\_Interval and Interval\_Offset)
10. WRITE Interval\_Offset = (any value greater than Log\_Interval)
11. WRITE Enable = TRUE
12. MAKE (logging object collect at least 2 records)
13. WRITE Enable = FALSE
14. CHECK (Log\_Buffer has collected two or more additional data records and two or more log-status entries)
15. CHECK (the timestamp for data records collected since the Interval\_Offset was last written, have Log\_Interval between records, at a fixed offset of Interval\_Offset modulo Log\_Interval)

Note to tester: Interval\_Offset in logging objects, and Log\_Interval are each an Unsigned number of hundredths of seconds. Excellent choices are 400, 500, 600, 1000, or 1200. When Interval\_Offset is larger than Log\_Interval, then Interval\_Offset modulo Log\_Interval, is smaller than Log\_Interval.

[Revise the references in BTL Test Plan to these two tests being added in BTL Specified Tests, in two different BIBBs, T-VMT-I-B, and T-VMMV-I-B, as shown]

### 7.3.2.25 Event Log Tests

The tests in this section verify that Event Log objects correctly record event notifications.

Some of the general logging object tests in Clause 7.3.2.24 are also applicable to the Event Log object type.

#### 7.3.2.25.1 Internal Logging of Notifications

Purpose: To verify the IUT correctly collects and represents the Notifications which it initiates.

Test Concept: Make the IUT generate two event notification messages which the IUT logs. Use ReadRange to retrieve them from an Event Log and compare the two representations.

Configuration Requirements: The tester shall choose two events which are configured to be sent to the TD and to be placed into one of the IUT's Event Logs, LO1.

Test Steps:

1. WRITE Enable = TRUE
2. MAKE (IUT generate an EventNotification)
3. RECEIVE ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = IUT,
  - 'Event Object Identifier' = (any valid object),
  - 'Time Stamp' = (T1, any valid timestamp),
  - '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)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (IUT generate an EventNotification)
6. RECEIVE ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = IUT,
  - 'Event Object Identifier' = (any valid object),
  - 'Time Stamp' = (T2, any valid timestamp),
  - '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 S3, any valid state for this event type),
  - 'To State' = (state S4, any valid state for this event type that can follow S3),
  - 'Event Values' = (any values appropriate to the event type)
7. TRANSMIT BACnet-SimpleACK-PDU
8. READ RC = LO1, Record\_Count
9. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = RC,
  - 'Count' = -2
10. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE, ?, FALSE, TRUE},
  - 'Item Count' = 2,
  - 'Item Data' = (logged data that matches the information received in steps 3 and 6, except that Process\_Identifier may be any value and is not required to match)
11. CHECK (T2 > T1, and that the notifications were logged in order)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the TD shall skip the steps in which a SimpleACK-PDU is sent.

### 7.3.2.26 Remote Logging of Notifications

Purpose: To verify that the IUT correctly collects and represents the Notifications which it receives.

Test Concept: Make TD send multiple event notification messages. Use ReadRange to retrieve the events from an Event Log or perhaps from multiple Event Logs in the IUT, and compare the two representations.

Configuration Requirements: LO1 is an Event Log object in IUT which logs the event types which are sent. Stop\_When\_Full in LO1 shall be FALSE or absent.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid object identifier),
  - 'Time Stamp' = (T1, any valid timestamp),
  - '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)
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = IUT,
  - 'Event Object Identifier' = (any valid object identifier),
  - 'Time Stamp' = (T2, any valid timestamp),
  - '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 S3, any valid state for this event type),
  - 'To State' = (state S4, any valid state for this event type that can follow S3),
  - 'Event Values' = (any values appropriate to the event type)
5. RECEIVE BACnet-SimpleACK-PDU
6. READ RC = LO1, Record\_Count
7. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = RC,
  - 'Count' = -2
8. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = LO1,
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE, ?, FALSE, TRUE},
  - 'Item Count' = 2,
  - 'Item Data' = (logged data that matches the information received in steps 2 and 4, except that Process\_Identifier can be any value and is not required to match)

9. CHECK (that the events were logged in the order in which they were received)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the test shall skip the steps in which a *BACnet-SimpleACK-PDU* is expected.

### 7.3.2.27 Internal Logging of ACK\_NOTIFICATIONs

Purpose: To verify the IUT correctly collects and represents an ACK\_NOTIFICATION which it initiates.

Test Concept: Make the IUT generate an ACK\_NOTIFICATION message. Use ReadRange to retrieve that same event from an Event Log and compare the two representations. If the IUT does not support logging of the ACK\_NOTIFICATIONs which it initiates, this test shall be skipped.

Configuration Requirements: O1 is an event initiating object in the IUT, which is configured to send event notifications to TD. LO1 is an Event Log object in IUT which logs ACK\_NOTIFICATIONs.

Test Steps:

1. WRITE Enable = TRUE
- ~~2. READ RC = LO1, Record\_Count~~
23. MAKE (the IUT generate a notification)
34. RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (PI1, any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (T1, any valid timestamp),  
    'Notification Class' = (N1, any valid notification class),  
    'Priority' = (P1, any valid priority),  
    'Event Type' = (ET1, any standard event type),  
    'Message Text' = (any character string),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = (S1, any valid state for this event type),  
    'To State' = (S2, any valid state for this event type),  
    'Event Values' = (any values appropriate to the event type)
45. TRANSMIT BACnet-SimpleACK-PDU
56. TRANSMIT AcknowledgeAlarm-Request,  
    'Acknowledging Process Identifier' = (any valid value),  
    'Event Object Identifier' = O1,  
    'Event State Acknowledged' = S2,  
    'Time Stamp' = T1,  
    'Time of Acknowledgment' = (the current time)
67. RECEIVE BACnet-SimpleACK-PDU
78. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = PI1,  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = O1,  
        'Time Stamp' = (T2, any valid timestamp > T1),  
        'Notification Class' = N1,  
        'Priority' = P1,  
        'Event Type' = ET1,  
        'Message Text' = (any character string),  
        'Notify Type' = ACK\_NOTIFICATION,  
        'From State' = S1
89. TRANSMIT BACnet-SimpleACK-PDU

9. *READ RC = LO1, Record\_Count*
10. TRANSMIT ReadRange-Request,  
     'Object Identifier' = LO1,  
     'Property Identifier' = Log\_Buffer,  
     'Reference Index' = RC,  
     'Count' = -1
11. RECEIVE ReadRange-ACK,  
     'Object Identifier' = LO1,  
     'Property Identifier' = Log\_Buffer,  
     'Result Flags' = {FALSE, ?, ~~FALSE~~TRUE},  
     'Item Count' = 1,  
     'Item Data' = (logged data that matches the information received in step 74,  
         except that Process\_Identifier can be any value and is not  
         required to match)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the TD shall skip the steps in which *BACnet-SimpleACK-PDUs* are sent in response to ConfirmedEventNotifications.

### 7.3.2.28 Remote Logging of ACK\_NOTIFICATIONS

Purpose: To verify that the IUT correctly collects and represents ACK\_NOTIFICATIONs which it receives.

Test Concept: Send an ACK\_NOTIFICATION to the IUT. Use ReadRange to retrieve that same event from an Event Log, and compare the two representations.

Configuration Requirements: LO1 is an Event Log object in IUT which logs ACK\_NOTIFICATIONs. Stop\_When\_Full in LO1 shall be FALSE or absent.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (any valid object identifier),  
     'Time Stamp' = (T1, any valid timestamp),  
     'Notification Class' = (any valid notification class),  
     'Priority' = (any valid priority),  
     'Event Type' = (any standard event type),  
     'Message Text' = (any character string),  
     'Notify Type' = ACK\_NOTIFICATION,  
     'From State' = (state S1, any valid state for this event type)
3. RECEIVE BACnet-SimpleACK-PDU
4. READ RC = LO1, Record\_Count
5. TRANSMIT ReadRange-Request,  
     'Object Identifier' = LO1,  
     'Property Identifier' = Log\_Buffer,  
     'Reference Index' = RC,  
     'Count' = -1
6. RECEIVE ReadRange-ACK,  
     'Object Identifier' = LO1,  
     'Property Identifier' = Log\_Buffer,  
     'Result Flags' = {FALSE, ?, ~~FALSE~~TRUE},  
     'Item Count' = 1,  
     'Item Data' = (logged data that matches the information received in step 2,  
         except that Process\_Identifier can be any value and is not required to match)

Notes to Tester: When the UnconfirmedEventNotification service is used instead of the ConfirmedEventNotification service, the test shall skip the step in which a *BACnet-SimpleACK-PDU* is expected.

### 7.3.2.30 Alert Enrollment Tests

#### 7.3.2.30.6 Out\_Of\_Service Property Test

BACnet Reference Clauses: 12.51.7

Purpose: This test case verifies that event forwarding is not done while Out\_Of\_Service is TRUE.

Test Concept: Set up both Recipient\_List and Subscribed\_Recipient recipient entries with no filters specified and then send event notifications to the Notification Forwarder while the value of the Out\_Of\_Service property is TRUE. Subscribed\_Recipients are configured as part of base setup 2 for Notification Forwarder object tests. Verify that forwarding of the event notifications is not performed.

If 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.

Configuration Requirements: Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Recipient\_List = { (all), --Valid Days  
(all), --From Time, To Time  
DEST\_OBJ\_ID2, --Recipient D2  
DEST\_PROCESS\_ID, --Process Identifier  
FALSE, --Issue Confirmed Notifications  
{T, T, T} --Transitions  
}) --One list element
2. MAKE (Out\_Of\_Service = TRUE)
3. VERIFY Out\_Of\_Service = TRUE
4. VERIFY Status\_Flags = (?FALSE, FALSE, ?FALSE, TRUE)
5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 

|                                |                                       |                                                |
|--------------------------------|---------------------------------------|------------------------------------------------|
| 'Process Identifier'           | = SRC_PROCESS_ID,                     |                                                |
| 'Initiating Device Identifier' | = SRC_NOTIF_DEV,                      |                                                |
| 'Event Object Identifier'      | = SRC_NOTIF_OBJ,                      |                                                |
| 'Time Stamp'                   | = (any valid time stamp),             |                                                |
| 'Notification Class'           | = SRC_NOTIF_CLS,                      |                                                |
| 'Priority'                     | = (any valid priority),               |                                                |
| 'Event Type'                   | = (any valid event type),             |                                                |
| 'Message Text'                 | = (optional, any valid message text), |                                                |
| 'Notify Type'                  | = SRC_NOTIF_TYP,                      |                                                |
| 'AckRequired'                  | = (any valid value),                  | -- absent if 'Notify Type' is ACK_NOTIFICATION |
| 'From State'                   | = (any valid From_State),             | -- absent if 'Notify Type' is ACK_NOTIFICATION |
| 'To State'                     | = (any valid To_State),               |                                                |
| 'Event Values'                 | = (any valid event values)            | -- absent if 'Notify Type' is ACK_NOTIFICATION |
6. WAIT Notification Fail Time
7. CHECK (the IUT did not transmit an event notification)
8. MAKE (Out\_Of\_Service = FALSE)
9. VERIFY Out\_Of\_Service = FALSE
10. VERIFY Status\_Flags = (?FALSE, ?, ?FALSE, FALSE)
11. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 

|                      |                   |  |
|----------------------|-------------------|--|
| 'Process Identifier' | = SRC_PROCESS_ID, |  |
|----------------------|-------------------|--|

'Initiating Device Identifier' = SRC\_NOTIF\_DEV,  
 'Event Object Identifier' = SRC\_NOTIF\_OBJ,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = SRC\_NOTIF\_CLS,  
 'Priority' = (any valid priority),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = SRC\_NOTIF\_TYP,  
 'AckRequired' = (any valid value), -- absent if 'Notify Type' is ACK\_NOTIFICATION  
 'From State' = (any valid From\_State), -- absent if 'Notify Type' is ACK\_NOTIFICATION  
 'To State' = (any valid To\_State),  
 'Event Values' = (any valid event values) -- absent if 'Notify Type' is ACK\_NOTIFICATION

12. BEFORE **Notification Fail Time** --The following can be in any order

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request

RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request

### 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<sub>1</sub>)  
 ELSE  
     MAKE (Present\_Value equal any valid value V<sub>1</sub>)
2. MAKE (the Accumulator increase its Present\_Value)
3. READ V<sub>2</sub> = Present\_Value)

4. READ  $V_3 = \text{Present\_Value}$ )
5. IF (the Accumulator is stopped) THEN
  - CHECK ( $V_3 = V_2 = ((\text{Prescale-multiplier}) * \text{pulse-count of signals generated by the measuring instrument}) / (\text{Prescale-moduloDivide}) + V_1$ )
  - 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,  $Pv_{\text{prior}}$  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.  $Pv_{\text{prior}}$  = 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:  $\text{Present\_Value} - Pv_{\text{prior}}$ ,
  - 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.  $Pv_{\text{prior}}$  = 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:  $(\text{Present\_Value} - \text{Value\_Set}) + (\text{Value\_Before\_Change} - Pv_{\text{prior}})$ ,



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 = (?, ?, ?, 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  $\neq$  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

### 7.3.2.X40 Channel Object Tests

#### 7.3.2.X40.2 Last\_Priority Test

Purpose: To verify that the initial value of Last\_Priority is 16. To verify that a Channel object correctly retains the priority of written values. To verify that a Last\_Priority will have a default priority of 16 if the last attempt to write to the Present\_Value was done without specifying the priority.

Test Concept: First, confirm that the default value of Last\_Priority is 16. Next, write a valid value to a Channel object using a valid priority level other than 16 and then check the value of Last\_Priority to make sure it shows the specified priority level. Finally, write a valid value to a Channel object again but without specifying priority and then check the value of Last\_Priority to make sure that it now shows 16 again.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable Present\_Value property which is either on local device or remote device.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. VERIFY Last\_Priority = 16
3. WRITE Present\_Value = (Any valid value), PRIORITY = (Y: Any valid value < 16)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY Last\_Priority = Y
6. WRITE Present\_Value = (Any valid value)
7. WAIT (**Channel Write Fail Time** \* LEN)
8. VERIFY Last\_Priority = 16

#### 7.3.2.X40.3 WriteGroup Service Support Test

Purpose: To verify that the Present\_Value of the Channel object can be written by WriteGroup Service

Test Concept: The Channel object, O1, is written to via WriteGroup and it is verified that the Present\_Value of the object is updated correctly.

Test Steps:

1. READ X = O1, Present\_Value
2. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = (any valid value),

'Change List' = (O1's channel number, no overriding priority, Y: a value different than X)

3. VERIFY Present\_Value = Y

### 7.3.2.X40.4 Propagation Entirety Test

Purpose: To verify that the Channel object keeps propagating the value to each local/remote object property reference in its List\_Of\_Object\_Property\_References after propagating the value fails for one or more target.

Test Concept: The Channel object, O1, is configured with at least 1 referenced target which the Channel object won't send the write to due to an invalid datatype coercion error. The Channel object's Present\_Value is written, and the writes to remote targets are checked. Once all writes complete, the Write\_Status is verified to be FAILED and all targets the Channel object sent the write to are verified to have accepted the written value.

Configuration Requirements: Configure the Channel object's List\_Of\_Object\_Property\_References so that at least one of the target references (contained in entry X of List\_Of\_Object\_Property\_References) will result in an invalid datatype coercion when the value WrittenValue is written to the Channel object. The rest of the target reference(s) shall be selected such that they will accept the written value as is without coercion. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the invalid datatypes.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N1 = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = WRITTEN\_VALUE
4. WAIT (**Channel Write Fail Time** \* LEN)
5. REPEAT REF = (each reference in O1.List\_Of\_Object\_Property\_References) {
  - IF (REF is not contained in the IUT) THEN
    - RECEIVE WriteProperty-Request
    - 'Object Identifier' = (Object Identifier of N1),
    - 'Property Identifier' = (Property Identifier of N1),
    - 'Property Value' = WrittenValue
  - IF (REF  $\diamond$  N1) THEN
    - TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Write\_Status = FAILED
7. REPEAT REF = (each reference in O1.List\_Of\_Object\_Property\_References) {
  - IF (REF  $\diamond$  N1) THEN
    - VERIFY REF = WRITTEN\_VALUE

### 7.3.2.X40.5 Write\_Status Test

Purpose: To verify that the Write\_Status of the Channel object is IDLE when it's List\_Of\_Object\_Property\_References is empty, and is IN\_PROGRESS while the Channel object's Present\_Value is being propagated, FAILED when propagation failed, and SUCCESSFUL when propagation succeeds.

Test Concept: The Channel object's List\_Of\_Object\_Property\_References is read and verified to be empty. Then, the Channel object's Write\_Status is verified to be IDLE. Next, any valid value is written to the Channel object's Present\_Value and Write\_Status is verified to be IDLE still. An object property reference, R1, that the IUT cannot reach is set into the List\_Of\_Object\_Property\_References and then any valid value is written to the Present\_Value and the Write\_Status is verified to be IN\_PROGRESS. After the IUT determines that the referenced device is offline, the Write\_Status is verified to be FAILED.

Finally, any valid reference, R2, that will cause successful value propagation is set to the List\_Of\_Object\_Property\_References and the test is repeated to verify that Write\_Status becomes SUCCESSFUL.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0

-- IDLE test

2. READ L = List\_Of\_Object\_Property\_References
3. VERIFY L = (empty)
4. VERIFY Write\_Status = IDLE
5. WRITE Present\_Value = (any valid value)
6. VERIFY Write\_Status = IDLE

-- IN\_PROGRESS and FAIL test

7. WRITE List\_Of\_Object\_Property\_References = (R1)-- write the whole array
8. WRITE Present\_Value = (any valid value)
9. VERIFY Write\_Status = IN\_PROGRESS
10. WAIT (**Channel Write Fail Time** \* LEN)
11. VERIFY Write\_Status = FAILED

-- SUCCESSFUL test

12. WRITE List\_Of\_Object\_Property\_References = (R2)-- write the whole array
13. WRITE Present\_Value = (any valid value)
14. WAIT (**Channel Write Fail Time** \* LEN)
15. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.6 Allow\_Group\_Delay\_Inhibit Test

Purpose: To verify that no Execution\_Delay will be applied to any of the writes if Allow\_Group\_Delay\_Inhibit is TRUE.

Test Concept: Setup List\_Of\_Object\_Property\_References to contain 2 valid entries PR1, PR2 and provide each with an execution delay (ED1 and ED2). Set Allow\_Group\_Delay\_Inhibit to TRUE so that no delays will occur between writes to referenced properties. Write to the Channel object's Present\_Value and verify that no delay occurs between writes to the referenced properties.

Set Allow\_Group\_Delay\_Inhibit to FALSE so that delays will occur between writes to referenced properties. Write to the Channel object's Present\_Value and verify that delays occur between writes to the referenced properties.

Configuration Requirements: PR1 and PR2 shall be references to writable properties and shall be the same datatype. ED1 and ED2 shall be values which are large enough that the delay between writes is sufficient for the test. V1 and V2 shall be of the expected datatype for PR1 and PR2 so that no coercion occur, and shall be different values. This test shall be skipped if the Channel object does not support at least 2 entries in the List\_Of\_Object\_Property\_References.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0

-- Setup the Channel object

2. WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)
3. WRITE Execution\_Delay = (ED1, ED2)

-- Test that delays are inhibited

4. WRITE Allow\_Group\_Delay\_Inhibit = TRUE
5. WRITE Present\_Value = V1
6. WAIT (**Channel Write Fail Time** \* LEN)
7. VERIFY PR1 = V1
8. VERIFY PR2 = V1
9. VERIFY Write\_Status = SUCCESSFUL

-- Test that delays are not inhibited

10. WRITE Allow\_Group\_Delay\_Inhibit = FALSE
11. WRITE Present\_Value = V2
12. WAIT (**Channel Write Fail Time** \* LEN)
13. VERIFY PR1 = V1
14. VERIFY PR2 = V1
15. VERIFY Write\_Status = IN\_PROGRESS
16. WAIT (ED1)
17. VERIFY PR1 = V2
18. VERIFY PR2 = V1
19. VERIFY Write\_Status = IN\_PROGRESS
20. WAIT (ED2 – ED1)
21. VERIFY PR2 = V2
22. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.7 Numeric to BOOLEAN Coercion Rule Test

Purpose: To verify that the Channel object correctly propagates all numeric datatype values to a BOOLEAN target based on Coercion Rule 1 – Numeric to BOOLEAN.

Test Concept: Write a value of 0 to the Present Value of the Channel object with a BOOLEAN target object property reference, verify that a target object property has a value of FALSE, and that Write\_Status of the Channel object shows SUCCESSFUL. When any non-zero numeric value is written to the Present\_Value of the same Channel object, verify that a target object property has a value of TRUE, and a Write Status shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable BOOLEAN object property.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ B = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = 0
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY B = FALSE
6. VERIFY Write\_Status = SUCCESSFUL
7. WRITE Present\_Value = (Any non-zero numeric value)
8. WAIT (**Channel Write Fail Time** \* LEN)
9. VERIFY B = TRUE
10. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.8 BOOLEAN to Numeric Coercion Rule Test

Purpose: To verify that the Channel object can correctly propagate BOOLEAN values to a numeric target object property reference based on Coercion Rule 2 – BOOLEAN to Numeric defined in ASHRAE 135.



Test Concept: When a value of FALSE is written to the Present\_Value of the Channel object with a numeric target object property reference, verify that the target object property has a value of 0, and a Write\_Status of the Channel object shows SUCCESSFUL. When a value of TRUE is written to the present value of the same Channel object, verify that a target object property has a value of 1, and a Write\_Status shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable numeric object property on the IUT. The referenced property shall not be 0 at the start of the test.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. VERIFY  $N \neq 0$  -- non-zero so that coercion is verified in the following write
4. WRITE Present\_Value = FALSE
5. WAIT (Channel Write Fail Time \* LEN)
6. VERIFY N = 0
7. VERIFY Write\_Status = SUCCESSFUL
8. WRITE Present\_Value = TRUE
9. WAIT (Channel Write Fail Time \* LEN)
10. VERIFY N = 1
11. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.9 Unsigned/INTEGER/REAL/Double to Numeric Coercion Rule Test

Purpose: To verify that the Channel object correctly propagates Unsigned, INTEGER, REAL or Double datatype values to a numeric target object property reference.

Test Concept: Select a Channel object with a numeric target property, N. Select an Unsigned, INTEGER, REAL or Double value, V1, which is in the acceptable range for N, and which coerces to value V2 based on N's datatype. Write V1 to the Present\_Value of the Channel object. Verify that the N changes to V2 and that Write\_Status of the Channel object is SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to the selected numeric property N. Configure the Channel object with no execution delays.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. VERIFY List\_Of\_Object\_Property\_References = N, ARRAY INDEX = X
3. WRITE Present\_Value = V1
4. WAIT (Channel Write Fail Time \* LEN)
5. VERIFY N = V2
6. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.10 Invalid Datatype Coercion Test

Purpose: To check that the Channel object does not write to a target object property reference and the Write\_Status indicates FAILED when invalid datatype coercion occur.

Test Concept: When an invalid data type value is written to a Present\_Value of the Channel object, verify that a target object reference value does not change.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that the Channel object will fail to propagate InvalidDataTypeValue. Refer to the Table 12-63 - Datatype Coercion Rules from ASHRAE 135 for the invalid datatypes.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = (InvalidDataTypeValue: Any invalid data type value)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY N  $\neq$  InvalidDataTypeValue
6. VERIFY Write\_Status = FAILED

### 7.3.2.X40.11 No Coercion Test

Purpose: To check that the Channel object can successfully write to a target object property reference without any value conversions and Write\_Status indicates SUCCESSFUL when no coercion occurs.

Test Concept: When a valid data type value is written to a Present\_Value of the Channel object using a value that require no coercion, verify that a written value is directly mapped to a target object reference and a Write\_Status of the Channel object shows SUCCESSFUL.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that no coercion will occur. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the data types that require no coercion.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. WRITE Present\_Value = (ValidDataTypeValue: Any value of a datatype that requires no coercion)
4. WAIT (**Channel Write Fail Time** \* LEN)
5. VERIFY N = ValidDataTypeValue
6. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.X40.12 Write Priority Test

Purpose: To check that the Channel object uses a priority level specified by a write service when the Channel object propagate its Present\_Value to (a) target object property reference(s). If no priority level is specified, check that 16 is used by default.

Test Concept: When a valid data type value is written to a Present\_Value of the Channel object by a WriteProperty request and a 'Priority' is provided in the write, the Channel object will use this same priority to command the referenced properties. When another value is written to a Present\_Value of the Channel object by a WriteProperty request with no 'Priority' specified, the Channel object will use a priority 16 by default to command the referenced properties.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a writable object property of a specific data type on the IUT such that no coercion will occur. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the data types that require no coercion. The referenced property must contain a Priority\_Array property.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
2. READ N = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
3. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (Object Identifier of the Channel object)

- 'Property Identifier' = Present\_Value  
 'Property Value' = (V1: Any value of a datatype that requires no coercion)  
 'Priority' = (P1: Any valid value but 16)
4. WAIT (**Channel Write Fail Time** \* LEN)
  5. RECEIVE BACnet-SimpleACK-PDU
  6. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Object Identifier of N)  
     'Property Identifier' = Priority\_Array  
     'Property Array Index' = P1
  7. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (Object Identifier of N)  
     'Property Identifier' = Priority\_Array  
     'Property Array Index' = P1  
     'Property Value' = V1
  8. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (Object Identifier of the Channel object)  
     'Property Identifier' = Present\_Value  
     'Property Value' = (V2: Any value of a datatype that requires no coercion)
  9. WAIT (**Channel Write Fail Time** \* LEN)
  10. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Object Identifier of N)  
     'Property Identifier' = Priority\_Array  
     'Property Array Index' = 16
  11. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (Object Identifier of N)  
     'Property Identifier' = Priority\_Array  
     'Property Array Index' = 16  
     'Property Value' = V2

### 7.3.2.X40.13 Writing with a NULL Value Test

Purpose: To check that the Channel object ignores datatype errors when writing a NULL value to a non-commandable target.

Test Concept: This test is to check a special exception of Write\_Status reporting SUCCESSFUL after propagating a NULL value to both a commandable and non-commandable property. Writing a NULL value to a commandable property will result in a property relinquishing a value to a Relinquish\_Default value. However, writing a NULL value to a non-commandable property will result in a property remaining a current value. If a non-commandable target property is on a remote device, the IUT will receive either an ERROR\_INVALID\_DATATYPE or REJECT\_INVALID\_PARAMETER\_DATA\_TYPE. The Write\_Status after such events should report SUCCESSFUL instead of FAILED.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commandable property that accepts NULL value and configure entry Y of the List\_Of\_Object\_Property\_References to a non-commandable property that rejects a NULL value with either ERROR\_INVALID\_DATATYPE or REJECT\_INVALID\_PARAMETER\_DATA\_TYPE. For a commandable property, all prioritized commands has to be relinquished and any minimum on/off time has to be accounted for prior to the test. An initial value of a commandable property, N1 must be different from a value of its Relinquish\_Default. N1's Priority\_Array has only one non-Null value in it and it is in the priority array level that the Channel object is targeting.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0  
 -- Read an initial value of target properties
  2. READ P1 = List\_Of\_Object\_Property\_References: ARRAY INDEX = X, which is a commandable property
  3. READ P2 = List\_Of\_Object\_Property\_References: ARRAY INDEX = Y, which is a non-commandable property
- Find expected values of the target properties after a NULL value is written to them

```

4. READ V1 = P1.Relinquish_Default
5. READ V2 = P2

-- Make the Channel object to propagate NULL value to targets
6. WRITE Present_Value = NULL
7. WAIT (Channel Write Fail Time * LEN)
8. IF (P2 is on external) THEN
 RECEIVE WritePropertyMultiple-Error
 'Error Class' = PROPERTY,
 'Error Code' = INVALID_DATATYPE |
 RECEIVE BACnet-Reject-PDU
 'Reject Reason' = INVALID_PARAMETER_DATATYPE

--Check that P1 has Relinquish_Default value and P2 remains the same
9. VERIFY P1 = V1
10. VERIFY P2 = V2

--Check that the Channel object ignores the datatype error and Write_Status is SUCCESSFUL
11. VERIFY Write_Status = SUCCESSFUL

```

### 7.3.2.X53 Load Control Object Tests

The Load Control object defines a standardized object whose properties represent the externally visible characteristics of a mechanism for controlling load requirements. A BACnet device can use a Load Control object to allow external control over the shedding of a load that it controls. The mechanisms by which the loads are shed are not visible to the BACnet client. The Load Control Object utilizes parameter control through its writable Requested\_Shed\_Level, Start\_Time, Shed\_Duration, Duty\_Window, Enable and Shed\_Levels properties.

#### 7.3.2.X53.1 Requested\_Shed\_Level property test with LEVEL choice

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

Purpose: To verify the performance of a shed request with LEVEL choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to a LEVEL choice and it is verified that the series of required actions which that sets into operation occur correctly.

Configuration Requirements: The IUT shall be configured so that Present\_Value is equal to SHED\_INACTIVE, preceding the beginning of this test. Writing Start\_Time and/or Shed\_Duration with values such that current time is after ST+SD forces Present\_Value to become equal to SHED\_INACTIVE.

Test Steps:

4. VERIFY Requested\_Shed\_Level = (one of the default Requested\_Shed\_Level values for a previous shed request, not necessarily the LEVEL default of 0)
5. VERIFY Expected\_Shed\_Level = (that same default Requested\_Shed\_Level value)
6. VERIFY Actual\_Shed\_Level = (that same default Requested\_Shed\_Level value)
7. VERIFY Present\_Value = SHED\_INACTIVE
8. VERIFY Shed\_Duration = 0
9. VERIFY Start\_Time = (the fully unspecified datetime value)
10. VERIFY Duty\_Window = (PAV, the pre-agreed upon value)
11. WRITE Enable = TRUE
12. WRITE Shed\_Duration = (SD, any value appropriate to the object)
13. WRITE Start\_Time = (ST, any value preceding the beginning of this test)
14. WRITE Duty\_Window = (DW, any value appropriate to the object)

15. WRITE Requested\_Shed\_Level = (a value appropriate to the object with a LEVEL choice, that is not equal to the default value: 0)
16. VERIFY Present\_Value = SHED\_REQUEST\_PENDING
17. WAIT (until Start\_Time)
18. VERIFY Present\_Value = (SHED\_REQUEST\_PENDING or SHED\_COMPLIANT or SHED\_NONCOMPLIANT)
19. IF (current time is before ST, but the shed request has started) THEN  
    VERIFY Present\_Value = SHED\_NONCOMPLIANT
20. IF (current time is at or after ST) THEN  
    VERIFY Present\_Value = (SHED\_COMPLIANT or SHED\_NONCOMPLIANT)
21. IF (current time is after ST+DW and Actual\_Shed\_Level does not comply with Requested\_Shed\_Value) THEN  
    VERIFY Present\_Value = SHED\_NONCOMPLIANT
22. VERIFY Shed\_Duration = SD
23. VERIFY Start\_Time = ST
24. VERIFY Duty\_Window = DW
25. VERIFY Expected\_Shed\_Level = (any value appropriate to the choice, that is not equal to the default value)
26. VERIFY Actual\_Shed\_Level = (any value appropriate to the choice, that is not equal to the default value)
- the above VERIFY statements apply all through the time that there is a pending or active shed event
27. WAIT (until the shed request has completed, at ST+SD)
28. VERIFY Requested\_Shed\_Level = 0
29. VERIFY Expected\_Shed\_Level = (0, that same Default LEVEL value)
30. VERIFY Actual\_Shed\_Level = (0, that same Default LEVEL value)
31. VERIFY Shed\_Duration = 0
32. VERIFY Start\_Time = (the fully unspecified datetime value)
33. VERIFY Duty\_Window = PAV

Notes to Tester: The writing of Duty\_Window can be skipped, for the tester to see that the VERIFY Duty\_Window = DW during a pending or active shed event, that property takes on PAV, the pre-agreed upon value.

### 7.3.2.X53.2 Shed\_Levels property test

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

Purpose: To verify writability of Shed\_Levels property and verify that when commanded with the LEVEL choice, the Load Control object shall take a shedding action described by the corresponding element in the Shed\_Level\_Descriptions array.

Test Concept: The Shed\_Levels property of the Load Control object being tested is written to BACnetARRAY of unsigned integers representing the shed levels for the LEVEL choice of BACnetShedLevel that have meaning for this particular Load Control object. Verify that is updating correctly. The array shall be ordered by increasing shed amount.

Test Steps:

1. READ N1 = Shed\_Levels, ARRAY INDEX = 0
2. VERIFY (Shed\_Level\_Descriptions = N1, ARRAY INDEX = 0)
3. WRITE Shed\_Levels = (any content that is different from the current value, but nonetheless still ordered by increasing shed amount)
4. READ N2 = Shed\_Levels, ARRAY INDEX = 0 -- obtaining the length of the new value
5. VERIFY (Shed\_Level\_Descriptions = N2, ARRAY INDEX = 0)

### 7.3.2.X53.3 Load Control Status\_Flags and Reliability Test

Purpose: To ensure Status\_Flags reflects the Reliability property value.

Test Concept: Write to Reliability and verify the interrelationship between the Status\_Flags and Reliability.

Configuration Requirements: The selected object is configured such that its Reliability is NO\_FAULT\_DETECTED before execution of this test. If the Reliability property is not present or not writable, then this test shall be skipped.

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. VERIFY Status\_Flags = (?, FALSE, ?, FALSE)
3. 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, ?, FALSE)
  - WRITE Reliability = NO\_FAULT\_DETECTED
  - VERIFY Reliability = NO\_FAULT\_DETECTED
  - VERIFY Status\_Flags = (? FALSE, ?, FALSE)

#### **7.3.2.X53.4 Requested\_Shed\_Level property test with PERCENT choice**

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

Purpose: To verify the performance of a shed request with PERCENT choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to a PERCENT choice and it is verified that the series of required actions which that sets into operation occur correctly.

Test Steps: The test steps defined in test **7.3.2.X53.1** shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to a PERCENT choice, and the default value for a shed request with PERCENT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 100

#### **7.3.2.X53.5 Requested\_Shed\_Level property test with AMOUNT choice**

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

Purpose: To verify the performance of a shed request with AMOUNT choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to an AMOUNT choice and it is verified that the series of required actions which that sets into operation occur correctly.

Test Steps: The test steps defined in test **7.3.2.X53.1** shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to an AMOUNT choice, and the default value for a shed request with AMOUNT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 0.0

#### **7.3.2.X54 Lighting Output Object Tests**

##### **7.3.2.X54.21 - Lighting Output Tracking Test**

Purpose: To verify that the Tracking\_Value property follows the Present\_Value property.

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

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

Test Steps:

1. WRITE Present\_Value = 100, PRIORITY = PTY1
2. VERIFY Present\_Value = 100
3. WHILE (In\_Progress <> IDLE) DO {  
    }  
4. VERIFY Tracking\_Value = 100
5. WRITE Present\_Value = 1, PRIORITY = PTY1
6. VERIFY Present\_Value = 1
7. WHILE (In\_Progress <> IDLE) DO {  
    }  
8. VERIFY Tracking\_Value = 1
9. WRITE Present\_Value = 0, PRIORITY = PTY1
10. VERIFY Present\_Value = 0
11. WHILE (In\_Progress <> IDLE) DO {  
    }  
12. VERIFY Tracking\_Value = 0

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

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

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

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

Test Steps:

1. VERIFY Present\_Value <> 1.0
2. WRITE Present\_Value = a value numerically greater than 0.0 but less than 1.0
3. VERIFY Present\_Value = 1.0

### 7.3.2.X54.31 Lighting Command Operation NONE Test

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

Test Concept: The TD writes the Lighting Command Operation NONE to the IUT, and expects Error Class of PROPERTY and an Error Code of VALUE\_OUT\_OF\_RANGE

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS);
2. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = O1  
    'Property Identifier' = Lighting\_Command  
    'Property Value' = NONE

3. RECEIVE BACnet-Error PDU,  
     Error Class =     PROPERTY,  
     Error Code =     VALUE\_OUT\_OF\_RANGE
4. VERIFY (Object1), Lighting\_Command = (the value defined for this property in the EPICS)

### 7.3.2.X54.32 Lighting Command Operation FADE\_TO Test

Purpose: To verify the correct operation of FADE\_TO lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

Test Concept: The TD writes to the Present\_Value at each end of the range (i.e. 0% or 100%), and then writes to the Lighting Command Operation with FADE\_TO with a long enough fade-time to allow In\_Progress and Tracking\_Value to be observed while set to FADE\_ACTIVE. The Tracking\_Value will be checked at the end of the fade to verify that it tracked the target level. The IUT shall be tested for fade up (0% to 100%) and fade down (100% to 0%).

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

Test Steps:

- Start with 0% Present\_Value to test fade up
  1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
  2. VERIFY Present\_Value = 0
  3. WAIT **Internal Processing Fail Time**
  4. VERIFY Tracking\_Value = 0
- Write a FADE\_TO command (operation, target-level, priority, fade-time)
  5. WRITE Lighting\_Command = (FADE\_TO, V1, PTY1, FT)
  6. WAIT **Internal Processing Fail Time**
  7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
  8. VERIFY Present\_Value = V1
- In a half way of fading up, check In\_Progress and Tracking\_Value
  9. WAIT FT/2
  10. VERIFY In\_Progress = FADE\_ACTIVE,
  11. VERIFY Tracking\_Value  $\approx$  V1 / 2
  12. WAIT FT/2
- When fading up is completed, check In\_Progress and Tracking\_Value
  13. VERIFY In\_Progress = IDLE
  14. VERIFY Tracking\_Value = V1
- Now repeat the test with 100% Present\_Value to test fade down
  15. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
  16. VERIFY Present\_Value = 100
  17. WAIT **Internal Processing Fail Time**
  18. VERIFY Tracking\_Value = 100
- Write a FADE\_TO command (operation, target-level, priority, fade-time)
  19. WRITE Lighting\_Command = (FADE\_TO, V2, PTY1, FT)
  20. WAIT **Internal Processing Fail Time**
  21. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1
  22. VERIFY Present\_Value = V2



-- In a half way of fading down, check In\_Progress and Tracking\_Value

23. WAIT FT/2
24. VERIFY In\_Progress = FADE\_ACTIVE,
25. VERIFY Tracking\_Value  $\approx$  V1 / 2
26. WAIT FT/2

-- When fading down is completed, check In\_Progress and Tracking\_Value

27. VERIFY In\_Progress = IDLE
28. VERIFY Tracking\_Value = V2

### 7.3.2.X54.33 Lighting Command Operation RAMP\_TO Test

Purpose: To verify the correct operation of RAMP\_TO lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

Test Concept: The TD writes to Present\_Value at each end of the range (i.e. 0% or 100%), and then writes to the Lighting Command Operation with RAMP\_TO with a slow enough ramp rate to allow In\_Progress and Tracking\_Value to be observed while set to RAMP\_ACTIVE. The Tracking\_Value will be checked at the end of the ramp to verify that it tracked the target level. The IUT shall be tested for ramp up (0% to 100%) and ramp down (100% to 0%).

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

Test Steps:

-- Start with 0% Present\_Value to test ramp up

1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present\_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking\_Value = 0

-- Write a RAMP\_TO command (operation, target-value, priority, ramp-rate)

5. WRITE Lighting\_Command = (RAMP\_TO, V1, PTY1, any valid rate)
6. WAIT **Internal Processing Fail Time**
7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
8. VERIFY Present\_Value = V1

-- Check In\_Progress while ramping up

9. VERIFY In\_Progress = RAMP\_ACTIVE

-- Make sure that Tracking\_Value increases with the ramp-rate

10. WHILE (In\_Progress  $\neq$  IDLE) DO {
11. VERIFY Tracking\_Value > 0 < V1
12. CHECK (Tracking\_Value is increasing with the ramp-rate)}

-- When ramping up is completed, check In\_Progress and Tracking\_Value

13. VERIFY In\_Progress = IDLE
14. VERIFY Tracking\_Value = V1

-- Now repeat the test with 100% Present\_Value to test ramp down

15. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
16. VERIFY Present\_Value = 100
17. WAIT **Internal Processing Fail Time**
18. VERIFY Tracking\_Value = 100

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

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

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

-- Check In_Progress and Tracking_Value
28. VERIFY In_Progress = IDLE
29. VERIFY Tracking_Value = V2
```

### 7.3.2.X54.34 Lighting Command Operation STEP\_UP Test

Purpose: To verify the correct operation of STEP\_UP lighting command by observing the value of Present\_Value, In\_Progress and Tracking\_Value.

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

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

Test Steps:

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

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

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

-- Now test with Tracking_Value >0
10. WRITE Present_Value = 1, ARRAY INDEX = PTY1
11. VERIFY Present_Value = 1
```

12. **WAIT Internal Processing Fail Time**

13. **VERIFY** Tracking\_Value = 1

-- Keep stepping up while continuously checking Priority\_Array, Present\_Value and Tracking\_Value

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

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

15. **WRITE** Lighting\_Command = (STEP\_UP, PTY1, any valid value)

16. **WAIT Internal Processing Fail Time**

17. **VERIFY** Priority\_Array = 100, ARRAY INDEX = PTY1

18. **VERIFY** Present\_Value = 100

19. **VERIFY** Tracking\_Value = 100

### 7.3.2.X54.35 Lighting Command Operation STEP\_DOWN Test

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

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_DOWN and any step increment. The Tracking\_Value shall remain at 0%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment greater than 99%, the Tracking\_Value shall be 1%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment less than 99%, the Tracking\_Value shall be 100% minus the step increment.

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

Test Steps:

-- Start with 0% Present\_Value

1. **WRITE** Present\_Value = 0, ARRAY INDEX = PTY1

2. **VERIFY** Present\_Value = 0

3. **WAIT Internal Processing Fail Time**

4. **VERIFY** Tracking\_Value = 0

-- Write a STEP\_DOWN command (operation, priority, step-increment)

5. **WRITE** Lighting\_Command = (STEP\_DOWN, PTY1, any valid value)

6. **WAIT Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking\_Value was 0

7. **VERIFY** Priority\_Array = 0, ARRAY INDEX = PTY1

8. **VERIFY** Present\_Value = 0

9. **VERIFY** Tracking\_Value = 0

-- Now test with Tracking\_Value = 100

10. **WRITE** Present\_Value = 100, ARRAY INDEX = PTY1

11. **VERIFY** Present\_Value = 100

12. **WAIT Internal Processing Fail Time**

13. **VERIFY** Tracking\_Value = 100

```

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

```

### 7.3.2.X54.36 Lighting Command Operation STEP\_ON Test

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

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_UP and any step increment. The Tracking\_Value shall be 1%. The TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment greater than 99%, the Tracking\_Value shall be 100%. The TD writes to Present\_Value at 1%, and then writes to the Lighting Command Operation with STEP\_UP and a step increment less than 99%, the Tracking\_Value shall be 1% plus the step increment.

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

#### Test Steps:

```

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

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

-- Confirm that the Present_Value and Tracking_Value became 1
7. VERIFY Priority_Array = 1, ARRAY INDEX = PTY1
8. VERIFY Present_Value = 1
9. VERIFY Tracking_Value = 1

-- Keep stepping on while continuously checking Priority_Array, Present_Value and Tracking_Value
10. REPEAT X = (1 through (100 - step-increment)) DO {
 WRITE Lighting_Command = (STEP_ON, PTY1, any valid values)
 WAIT Internal Processing Fail Time
 VERIFY Priority_Array = X + step-increment, ARRAY INDEX = PTY1
 VERIFY Present_Value = X + step-increment
 VERIFY Tracking_Value = X + step-increment
}

```

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

11. WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
12. WAIT **Internal Processing Fail Time**
13. VERIFY Priority\_Array = 100, ARRAY INDEX = PTY1
14. VERIFY Present\_Value = 100
15. VERIFY Tracking\_Value = 100

### 7.3.2.X54.37 Lighting Command Operation STEP\_OFF Test

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

Test Concept: The TD writes to Present\_Value at 0%, and then writes to the Lighting Command Operation with STEP\_DOWN and any step increment. The Tracking\_Value shall remain at 0%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment greater than 99%, the Tracking\_Value shall be 1%. The TD writes to Present\_Value at 100%, and then writes to the Lighting Command Operation with STEP\_DOWN and a step increment less than 99%, the Tracking\_Value shall be 100% minus the step increment.

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

Test Steps:

-- Start with 0% Present\_Value

1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
2. VERIFY Present\_Value = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Tracking\_Value = 0

-- Write a STEP\_OFF command (operation, priority, step-increment)

5. WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the command was ignored since Tracking\_Value was 0

7. VERIFY Priority\_Array = 0, ARRAY INDEX = PTY1
8. VERIFY Present\_Value = 0
9. VERIFY Tracking\_Value = 0

-- Now test with Tracking\_Value = 100

10. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
11. VERIFY Present\_Value = 100
12. WAIT **Internal Processing Fail Time**
13. VERIFY Tracking\_Value = 100

-- Keep stepping off while continuously checking Priority\_Array, Present\_Value and Tracking\_Value

14. REPEAT X = (100 through (1 + step-increment)) DO {
  - WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
  - WAIT **Internal Processing Fail Time**
  - VERIFY Priority\_Array = X - step-increment, ARRAY INDEX = PTY1
  - VERIFY Present\_Value = X - step-increment
  - VERIFY Tracking\_Value = X - step-increment

-- Confirm that the Present\_Value and Tracking\_Value become 0 when STEP OFF command is executed while Tracking\_Value is 1

15. WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)
16. WAIT **Internal Processing Fail Time**
17. VERIFY Priority\_Array = 0, ARRAY INDEX = PTY1
18. VERIFY Present\_Value = 0
19. VERIFY Tracking\_Value = 0

#### 7.3.2.X54.41 Transition None test

Purpose: To verify that the Tracking\_Value property immediately follows the Present\_Value property if Transition is NONE.

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

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

Test Steps:

1. VERIFY Transition = NONE
2. VERIFY In\_Progress = IDLE
3. WRITE Present\_Value = 100, ARRAY INDEX = PTY1
4. VERIFY In\_Progress = IDLE
5. VERIFY Tracking\_Value = 100
6. WRITE Present\_Value = 1, ARRAY INDEX = PTY1
7. VERIFY In\_Progress = IDLE
8. VERIFY Tracking\_Value = 1

#### 7.3.2.X54.42 Transition Test

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

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

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

To test FADE functionality, T is FADE, A is FADE\_ACTIVE, W1 and W2 are (Default\_Fade\_Time / 2), and Default\_Fade\_Time is sufficiently large so as to allow the intermediate progress checks.

To Test RAMP functionality, T is RAMP, A is RAMP\_ACTIVE, W1 is ((V1 - V0) / Default\_Ramp\_Rate) / 2, W2 is ((V1 - V2) / Default\_Ramp\_Rate) / 2, and Default\_Ramp\_Rate is sufficiently small so as to allow the intermediate progress checks.

Test Steps:

1. VERIFY Transition = T
2. VERIFY In\_Progress = IDLE
3. V0 = READ Present\_Value
4. WRITE Present\_Value = V1, ARRAY INDEX = PTY1
5. VERIFY Present\_Value = V1
6. WAIT W1
7. VERIFY Tracking\_Value  $\approx (V1 + V0) / 2$
8. VERIFY In\_Progress = A
9. CHECK (the physical output is fading from V0 to V1 )
10. WAIT W1
11. VERIFY In\_Progress = IDLE
12. VERIFY Tracking\_Value = V1
13. WRITE Present\_Value = V2, ARRAY INDEX = PTY1
14. VERIFY Present\_Value = V2
15. WAIT W2
16. VERIFY Tracking\_Value  $\approx (V2 + V1) / 2$
17. VERIFY In\_Progress = A
18. CHECK (the physical output is fading V1 to V2 )
19. WAIT W2
20. VERIFY In\_Progress = IDLE
21. VERIFY Tracking\_Value = V2

### 7.3.2.X54.51 Feedback\_Value Clamping Test

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

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

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

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

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

Test Steps:

-- Verify Feedback\_Value when output is above Max\_Actual\_Value

1. WRITE Max\_Actual\_Value = 100
2. WRITE Min\_Actual\_Value = 1
3. WRITE Present\_Value = 100, PRIORITY = PTY1
4. WHILE In\_Progress  $\neq$  IDLE {}
5. WRITE Max\_Actual\_Value = (Lowest supported Max\_Actual\_Value)
6. VERIFY Feedback\_Value = 100

-- Verify Feedback\_Value when output is below Min\_Actual\_Value

7. WRITE Max\_Actual\_Value = 100
8. WRITE Min\_Actual\_Value = 1

9. WRITE Present\_Value = 1, PRIORITY = PTY1
10. WHILE In\_Progress <> IDLE {}
11. WRITE Min\_Actual\_Value = (Highest supported Min\_Actual\_Value)
12. VERIFY Feedback\_Value = 1

### 7.3.2.X54.61 Min\_Actual\_Value and Max\_Actual\_Value Test

Purpose: To verify that Min\_Actual\_Value remains less than Max\_Actual\_Value and within the allowable range when either is written to a value that would violate these conditions.

Test Concept: Write a value to Min\_Actual\_Value which is larger than Max\_Actual\_Value. Verify that Max\_Actual\_Value became equal to Min\_Actual\_Value. Next, write a value to Max\_Actual\_Value which is less than Min\_Actual\_Value. Verify that Min\_Actual\_Value became equal to Max\_Actual\_Value.

Verify that neither Min\_Actual\_Value nor Max\_Actual\_Value will accept a value outside the range 1.0 to 100.0.

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

Test Steps:

1. V1 = READ Max\_Actual\_Value
2. WRITE Min\_Actual\_Value = V2, a value greater than V1
3. VERIFY Max\_Actual\_Value = V2
4. WRITE Max\_Actual\_Value = V3, a value less than V2
5. VERIFY Min\_Actual\_Value = V3
6. TRANSMIT WritePropertyRequest  
     'Object Identifier' = O1,  
     'Property Identifier' = Min\_Actual\_Value,  
     'Property Value' = (any value outside the range 1.0 to 100.0)
7. RECEIVE BACnet-Error-PDU,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE
8. TRANSMIT WritePropertyRequest  
     'Object Identifier' = O1,  
     'Property Identifier' = Max\_Actual\_Value,  
     'Property Value' = (any value outside the range 1.0 to 100.0)
9. RECEIVE BACnet-Error-PDU,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE

### 7.3.2.X54.62 Min\_Actual\_Value and Max\_Actual\_Value Scaling Test

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

Test Concept: Set Min\_Actual\_Value to a value other than the lowest supported minimum value, and set Max\_Actual\_Value to a value other than the highest support value but larger than Min\_Actual\_Value.

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



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

Test Steps:

1. WRITE Min\_Actual\_Value = (a supported value that is not the lowest supported value)
2. WRITE Max\_Actual\_Value = (a supported value which is not the highest support value)
3. WRITE Present\_Value = 1.0, ARRAY INDEX = PTY1
4. CHECK(the value of the physical output is Min\_Actual\_Value)
5. WRITE Present\_Value = 100.0, ARRAY INDEX = PTY1
6. CHECK(the value of the physical output is Max\_Actual\_Value)
7. WRITE Present\_Value = (V1, a value between 1.0 and 100.0 exclusive), ARRAY INDEX = PTY1
8. MAKE(measure the value of the physical output and record in MV)
9. CHECK ( $MV \approx \text{Min\_Actual\_Value} + (V1 / 100) * (\text{Max\_Actual\_Value} - \text{Min\_Actual\_Value})$ )

### 7.3.2.X55 Access Door Object Tests

#### 7.3.2.X55.1.X1 Commandable Present\_Value Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that writing to the Present\_Value will cause a corresponding change to the physical output.

Test Concept: The IUT shall be configured with a door control output that can be observed during the test. The Present\_Value property is written with each of the following values: UNLOCK, LOCK, PULSE\_UNLOCK, EXTENDED\_PULSE\_UNLOCK and the Access Door object is monitored to ensure that the door locks and unlocks appropriately.

Configuration Requirements: The Relinquish\_Default shall have the value LOCK. All writes are at a priority higher than any internal algorithms writing to this property. Out\_Of\_Service shall be set to FALSE. Prior to the test the Present\_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

-- Test UNLOCK value

1. WRITE Present\_Value = UNLOCK
2. WAIT (**Internal Processing Fail Time**)
3. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
4. CHECK (that the door control output is in a state that would cause the door to be unlocked)

-- Test LOCK value

5. WRITE Present\_Value = LOCK
6. WAIT (**Internal Processing Fail Time**)
7. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
8. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test PULSE\_UNLOCK value

9. WRITE Present\_Value = PULSE\_UNLOCK
10. WAIT (**Internal Processing Fail Time** + Door\_Unlock\_Delay\_Time if present)
11. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED

12. CHECK (that the IUT is in a state that would cause the door to be unlocked)
13. WAIT (Door\_Pulse\_Time)
14. VERIFY Present\_Value = LOCK
15. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
16. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED\_PULSE\_UNLOCK value

17. WRITE Present\_Value = EXTENDED\_PULSE\_UNLOCK
18. WAIT (**Internal Processing Fail Time** + Door\_Unlock\_Delay\_Time if present)

### 7.3.2.X55.1.X2 Door\_Status, Lock\_Status and Door\_Alarm\_State Tests

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: This test case verifies that Door\_Status, Lock\_Status and Door\_Alarm\_State properties are writable when Out\_Of\_Service is TRUE.

Test Concept: Set Out\_Of\_Service to TRUE and then make sure one at a time that Door\_Status, Lock\_Status and Door\_Alarm\_State, if present, are writable.

Configuration Requirements: If the Out\_Of\_Service property of this object is not writable, and if the Out\_Of\_Service property cannot be changed by other means, then this test shall be omitted. All writes to the Present\_Value shall be performed at a priority higher (numerically smaller) than any internal algorithms writing to this property. For testing Door\_Alarm\_State, test only values listed in either the Alarm\_Values or Fault\_Values.

Test Steps:

1. MAKE (Out\_Of\_Service TRUE)
2. VERIFY Status\_Flags = (?, ?, TRUE)
3. IF (Door\_Status is present) THEN  
    REPEAT X = (all values of the Door\_Status enumeration values supported by the property)  
    DO {  
        WRITE Door\_Status = X  
        VERIFY Door\_Status = X  
    }  
4. IF (Lock\_Status is present) THEN  
    REPEAT X = (all values of the Lock\_Status enumeration values supported by the property)  
    DO {  
        WRITE Lock\_Status = X  
        VERIFY Lock\_Status = X  
    }  
5. IF (Door\_Alarm\_State is present) THEN  
    REPEAT X = (all values of the Door\_Alarm\_State enumeration values supported by the property)  
    DO {  
        WRITE Door\_Alarm\_State = X  
        VERIFY Door\_Alarm\_State = X  
    }

### 7.3.2.X55.1.X3 Door\_Status with Physical Door Status Tests

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Door\_Status property reflects the state of the physical door (CLOSED, OPENED, UNUSED and DOOR\_FAULT if the object supports detecting door faults).

Test Concept: The IUT is configured to monitor the state of a physical door. The physical door may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can determine the state of a door. The Access\_Door object associated with this physical door shall be configured with Out\_Of\_Service = FALSE.

Test Steps:

1. MAKE (set physical door to the closed state)
2. VERIFY Door\_Status = CLOSED
3. MAKE (set physical door to the opened state)
4. VERIFY Door\_Status = OPENED
5. IF (the object supports detecting door faults) THEN  
    MAKE (set the physical door to a state that would cause the Door\_Status to take on a value of DOOR\_FAULT)  
    VERIFY Door\_Status = DOOR\_FAULT
6. IF (possible to remove a door status input associated with the door) THEN  
    MAKE (remove a door status input associated with the door)
7. VERIFY Door\_Status = UNUSED | UNKNOWN

#### **7.3.2.X55.1.X4 Lock\_Status Tests**

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Lock\_Status property reflects the state of the physical lock. (LOCKED, UNLOCKED and LOCK\_FAULT if the object supports detecting lock faults).

Test Concept: The IUT monitors the state of a physical lock. The state of the physical lock may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can monitor the state of the physical lock. The Access\_Door object associated with this physical door shall be configured with Out\_Of\_Service = FALSE. The physical lock shall be manipulated other than through the Access Door object.

Note to tester: The physical lock shall be manipulated other than through the Access Door object.

Test Steps:

1. MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of LOCKED)
2. VERIFY Lock\_Status = LOCKED
3. MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of UNLOCKED)
4. VERIFY Lock\_Status = UNLOCKED
5. IF (the object and the lock support detecting lock faults) THEN  
    MAKE (set the physical lock to a state that would cause the Lock\_Status to take on a value of LOCK\_FAULT)  
    VERIFY Lock\_Status = LOCK\_FAULT

### 7.3.2.X55.1.X5 Secured\_Status Tests

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Secured\_Status property reflects the state of the physical lock, the physical door and the state of the Access Door object.

Test Concept: Start the test by creating a condition where the Secured\_Status = SECURED. Then create various conditions one at a time to verify that the Secured\_Status becomes UNSECURED when it should.

Configuration Requirements: All writes to the Present\_Value shall be performed at a priority higher than any internal algorithms writing to this property. If this object supports intrinsic reporting then the Alarm\_Values property shall be empty. If this object supports the Masked\_Alarm\_Values property then it shall be empty. Out\_Of\_Service is FALSE.

Test Steps:

-- Create a condition where the Secured\_Status becomes SECURED

1. WRITE Present\_Value = LOCK
2. WAIT (**Internal Processing Fail Time**)
3. VERIFY Status\_Flags = (FALSE ?, ?, ?)
4. IF (Lock\_Status property is present) THEN  
    MAKE (Lock\_Status = LOCKED or UNUSED)
5. MAKE (Door\_Status = CLOSED or UNUSED)

-- Verify that the Secured\_Status is SECURED when it should

6. VERIFY Secured\_Status = SECURED

-- Verify that Secured\_Status is UNSECURED when Present\_Value is anything other than LOCKED

7. REPEAT X = (UNLOCK, PULSE\_UNLOCK, EXTENDED\_PULSE\_UNLOCK) DO {  
    WRITE Present\_Value = X  
    WAIT (**Internal Processing Fail Time**)  
    VERIFY Secured\_Status = UNSECURED  
}

-- Recreate a condition where the Secured\_Status becomes SECURED again

8. WRITE Present\_Value = LOCK
9. WAIT (**Internal Processing Fail Time**)
10. VERIFY Secured\_Status = SECURED

-- Verify that Secured\_Status is UNSECURED when Masked\_Alarm\_Value, if exist, is NOT empty

11. IF (Masked\_Alarm\_Values is present) THEN  
    MAKE (Masked\_Alarm\_Values = (any valid BACnetDoorAlarmState enumeration))  
    WAIT(**Internal Processing Fail Time**)  
    VERIFY Secured\_Status = UNSECURED

-- Recreate a condition where the Secured\_Status becomes SECURED again

MAKE ( Masked\_Alarm\_Values = {})  
WAIT (**Internal Processing Fail Time**)  
VERIFY Secured\_Status = SECURED

-- Verify that Secured\_Status is UNSECURED when Lock\_Status, if present, is anything other than LOCKED or UNUSED

12. IF (Lock\_Status property is present) THEN  
    REPEAT X = (UNLOCKED, UNKNOWN, LOCK\_FAULT) DO {  
        MAKE (Lock\_Status = X)  
        WAIT (**Internal Processing Fail Time**)  
        VERIFY Secured\_Status = UNSECURED  
    }

```

REPEAT X = (LOCKED, UNUSED) DO {
 MAKE (Lock_Status = X)
 VERIFY Secured_Status = SECURED
}

```

-- Verify that Secured\_Status is UNSECURED when Door\_Status, is anything other than CLOSED or UNUSED

```

13. REPEAT X = (OPEN, UNKNOWN, DOOR_FAULT) DO {
 MAKE (Door_Status = X)
 WAIT (Internal Processing Fail Time)
 VERIFY Secured_Status = UNSECURED
}
REPEAT X = (CLOSED, UNUSED) DO {
 MAKE (Door_Status = X)
 WAIT (Internal Processing Fail Time)
 VERIFY Secured_Status = SECURED
}

```

-- Verify that Secured\_Status is UNSECURED when In\_Alarm bit of Status\_Flag is True

```

14. IF (Alarming is supported) THEN
 IF (Alarm_Values is writable) THEN
 WRITE Alarm_Values = { AV: any valid value}
 MAKE (trigger an alarm by using a physical door/lock to create the door alarm state AV)
 WAIT (Internal Processing Fail Time + Time_Delay)
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY Secured_Status = UNSECURED

```

### 7.3.2.X55.1.X6 Door\_Unlock\_Delay\_Time Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that when the Door\_Unlock\_Delay\_Time property has a non-zero value, the output is delayed in unlocking when a PULSE\_UNLOCK or EXTENDED\_PULSE\_UNLOCK is written to the Present\_Value and not when UNLOCK is written.

Test Concept: When unlocking the door by writing PULSE\_UNLOCK to the Present\_Value of the Access Door object, it is verified that the door is still locked for the specified Door\_Pulse\_Time then the door is unlocked. The same test is done for EXTENDED\_PULSE\_UNLOCK, but this time it is verified that the door is still locked for the specified Door\_Extended\_Pulse\_Time then the door is unlocked.

Configuration Requirements: The IUT shall be configured with a door control output that can be observed during the test. The Relinquish\_Default shall have the value LOCK. All writes to the Present\_Value shall be performed at a priority higher than any internal algorithms writing to this property. Door\_Unlock\_Delay\_Time shall be set to a non-zero value which is sufficient to observe the delay and check the status of the lock. Out\_Of\_Service shall be set to FALSE. Prior to the test the Present\_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

```

-- Test PULSE_UNLOCK
1. WRITE Present_Value = PULSE_UNLOCK
2. WAIT (Internal Processing Fail Time)
3. BEFORE Door_Unlock_Delay_Time
 IF (Lock_Status is present) THEN
 VERIFY Lock_Status = LOCKED
 CHECK (that the door control output is in a state that would cause the door to be locked)

```

4. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
5. CHECK (that the door control output is in a state that would cause the door to be unlocked)
6. WAIT (Door\_Pulse\_Time)
7. VERIFY Present\_Value = LOCK
8. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
9. CHECK (that the door control output is in a state that would cause the door to be locked)
- Test EXTENDED\_PULSE\_UNLOCK
10. WRITE Present\_Value = EXTENDED\_PULSE\_UNLOCK
11. WAIT (**Internal Processing Fail Time**)
12. BEFORE Door\_Unlock\_Delay\_Time  
    IF (Lock\_Status is present) THEN  
        VERIFY Lock\_Status = LOCKED  
    CHECK (that the door control output is in a state that would cause the door to be locked)
13. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
14. CHECK (that the door control output is in a state that would cause the door to be unlocked)
15. WAIT (Door\_Extended\_Pulse\_Time)
16. VERIFY Present\_Value = LOCK
17. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = LOCKED
18. CHECK (that the door control output is in a state that would cause the door to be locked)
- Test UNLOCK
19. WRITE Present\_Value = UNLOCK
20. WAIT (**Internal Processing Fail Time**)
21. IF (Lock\_Status is present) THEN  
    VERIFY Lock\_Status = UNLOCKED
22. CHECK (that the door control output is in a state that would cause the door to be locked)

### 7.3.2.X55.1.X7 Masked\_Alarm\_Values Tests

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Masked\_Alarm\_Values prevents an intrinsic alarm from occurring.

Test Concept: The Access Door is verified to be in an Out\_Of\_Service state and is not in an alarm state. Then a non-NORMAL enumeration value of BACnetDoorAlarmState X is written to the Door\_Alarm\_State and the Access Door object transitions to an alarm state. X is written to the Masked\_Alarm\_Value and Door\_Alarm\_State is checked to verify it returned to NORMAL. The sequence is repeated for all non-NORMAL enumeration values of BACnetDoorAlarmState.

Configuration Requirements: The Masked\_Alarm\_Values list shall be empty at the start of this test. Out\_Of\_Service shall be set to TRUE to allow writing to the Door\_Alarm\_State property. If Out\_Of\_Service is not writeable and cannot be set to TRUE by any other means, this test shall be skipped. The enumeration BACnetDoorAlarmState value X to be used in the test has to be present in either the Alarm\_Values or Fault\_Values property.

Test Steps:

1. VERIFY Status\_Flags = (FALSE ?, ?, TRUE)
2. VERIFY Door\_Alarm\_State = NORMAL
3. REPEAT X = (all valid values of the enumeration BACnetDoorAlarmState except NORMAL)  
    DO {

```

WRITE Door_Alarm_State = X
WAIT (Internal Processing Fail Time)
VERIFY Status_Flags = (TRUE ?, ?, TRUE)
WRITE Masked_Alarm_Values= { X }
WAIT (Internal Processing Fail Time)
VERIFY Door_Alarm_State = NORMAL
VERIFY Status_Flags = (FALSE ?, ?, TRUE)
WRITE Masked_Alarm_Values= { }
WAIT (Internal Processing Fail Time)
}

```

### 7.3.2.X55.1.X8 Door\_Open\_Too\_Long Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the DOOR\_OPEN\_TOO\_LONG condition is generated when the Access Door object is commanded to the LOCK state but the physical door remains open beyond Door\_Open\_Too\_Long\_Time.

Test Concept: Setup the Access Door object to trigger alarm on DOOR\_OPEN\_TOO\_LONG state using Alarm\_Values and Masked\_Alarm\_Values. Next, set the physical door to the closed state to confirm that the Access Door object is in NORMAL state. Then, unlock the physical door and set the physical door to the open state. Finally, command the Access Door object to LOCK and verify that the Door\_Alarm\_State changes to DOOR\_OPEN\_TOO\_LONG after the specified Time\_Delay.

Configuration Requirements: This test shall be skipped if the IUT does not support intrinsic alarming. The IUT shall be configured such that it can determine and change the open/closed state of a door. All writes to the Present\_Value are at a priority higher than any internal algorithms writing to this property. The Door\_Alarm\_State shall have the value NORMAL at the start of the test. The Access Door object is configured with DOOR\_OPEN\_TOO\_LONG in the Alarm\_Values property and excluded from Masked\_Alarm\_Values property if present.

Test Steps:

1. MAKE (set the physical door to the closed state)
2. VERIFY Door\_Alarm\_State = NORMAL
3. WRITE Present\_Value = UNLOCK
4. MAKE (set the physical door to the open state)
5. WRITE Present\_Value = LOCK
6. WAIT (**Internal Processing Fail Time**)
7. WHILE (Door\_Open\_Too\_Long\_Time has not expired) DO {
  - VERIFY Door\_Alarm\_State = NORMAL
 }
 WAIT (Time\_Delay)
8. VERIFY Door\_Alarm\_State = DOOR\_OPEN\_TOO\_LONG

### 7.3.2.X56 Access Point Object Tests

The Access Point object type represents the external interface of the access control decision engine for a specific door. A credential is entered, the access rights of the credential are determined and the access decision is determined based on the access rights. Testing this authentication and authorization functionality requires the support of other standard BACnet access control object types. The required and optional object types are shown in figure X1.

The access decision begins with a credential value being sent to the Access Point object for evaluation. Typically the credential value is read at a Credential Data Input object which extracts the raw credential data from the physical reader, formats the data and then sends it to the corresponding Access Point object. If the Credential Data Input object type is not supported then the vendor must provide an alternate method for the credential to be received by the Access Point.

When a credential value is received by the Access Point object it searches through the Access Credential objects to find the one with a matching credential value. For each credential value being tested a corresponding Access Credential must exist within the IUT. The only exception to this is when testing for an unknown credential (DENIED\_UNKNOWN\_CREDENTIAL).

To determine if access is granted or denied for a specific credential each Access Credential object must reference an Access Rights object which defines the appropriate access rights corresponding to the specific test being executed.

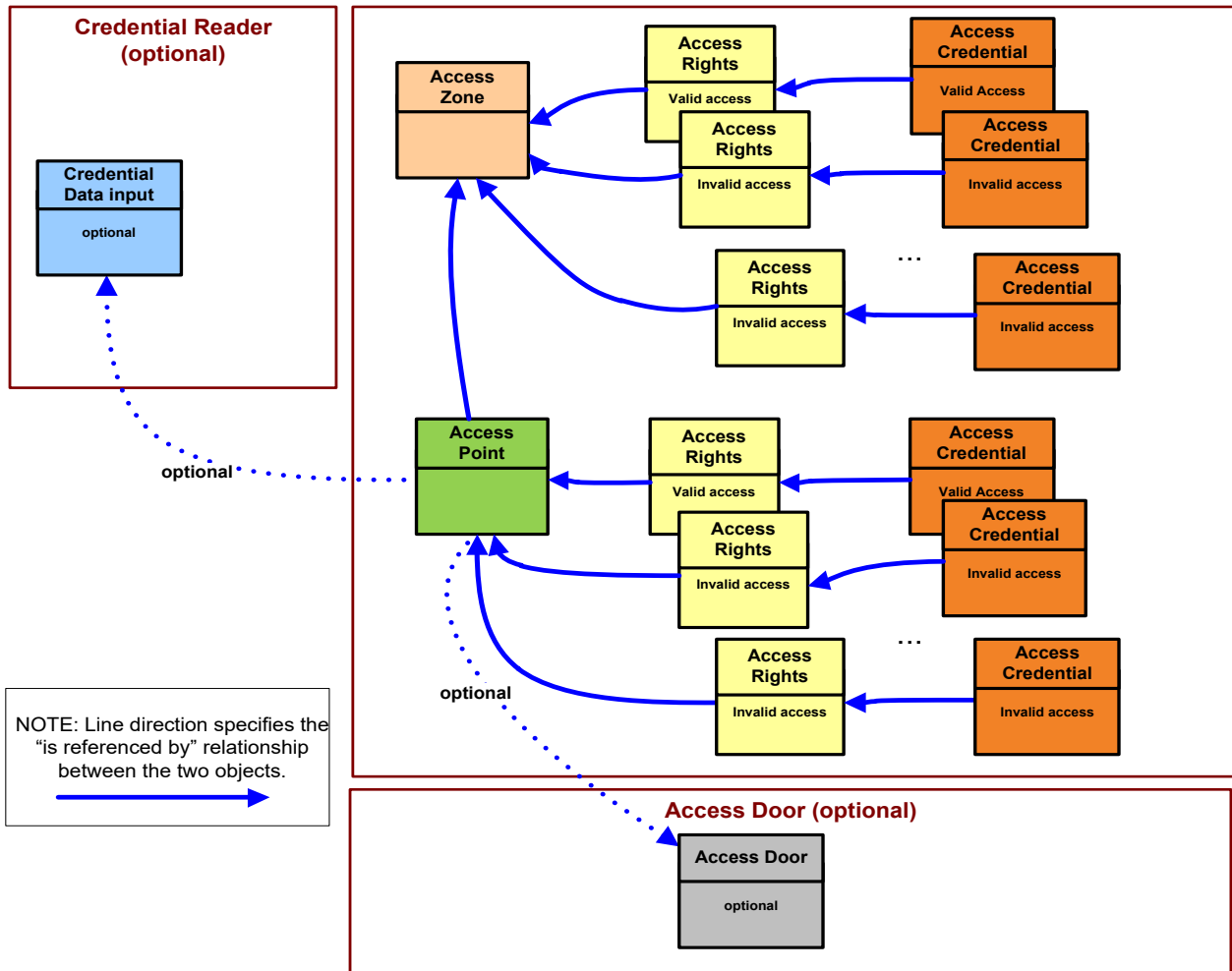
Some of the allowed and denied access tests require the Access Zone object. In this case an Access Point must be configured to be an entry access point to the access zone. These tests require that the Access Rights objects reference the Access Zone rather than the Access Point.

When the access decision is determined the IUT shall provide a method to indicate the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.

Unless specified otherwise in the specific test, the access point object (AP1) in the following tests shall have the following configuration:

- a) Authorization\_Mode shall have the value AUTHORIZE.
- b) Out\_Of\_Service shall be FALSE.
- c) Lockout shall be FALSE.
- d) Muster\_Point shall be FALSE.
- e) Authentication\_Status shall be READY.





**Figure X1:** Objects and relationships used for testing Access Point objects

### 7.3.2.X56.1 Authentication Status and Access Event Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: This test case verifies the authentication and authorization process are disabled when Out\_Of\_Service of the Access Point object is TRUE. It also verifies the interrelationship between the Out\_Of\_Service, Authentication\_Status, Access\_Event and Access\_Event\_Time properties.

Test Concept: Write TRUE to the Out\_Of\_Service property and verify that the authorization and authentication functions are disabled. Write FALSE to the Out\_Of\_Service property and verify that they are enabled.

### Configuration Requirements:

See 7.3.2.X56

### Test Steps:

- ```

1. VERIFY Authentication_Status = READY
2. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = TRUE
ELSE

```

- ```

 MAKE (Out_Of_Service TRUE)
3. VERIFY Authentication_Status = DISABLED
4. VERIFY Access_Event = OUT_OF_SERVICE
5. VERIFY Access_Event_Time = (the time Out_Of_Service was set to TRUE)
6. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
 ELSE
 MAKE (Out_Of_Service FALSE)
7. VERIFY Authentication_Status = READY
8. VERIFY Access_Event = OUT_OF_SERVICE_RELINQUISHED
9. VERIFY Access_Event_Time = (the time Out_Of_Service was set to FALSE)

```

### 7.3.2.X56.2 Allowed Access Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that a valid credential that is allowed access to this access point at this time is granted access.

Test Concept: A valid credential, that has access to the access point being tested at the current time, is presented at the access point. It is then verified that access is allowed and the appropriate access event is generated.

Configuration Requirements: See 7.3.2.X56. This test requires the following additional configuration: An active credential with valid access rights for the access point shall be represented by Access Credential object C1.

Test Steps:

1. READ EventTag = Access\_Event\_Tag
2. MAKE (present a valid credential at credential reader for this access point)
3. VERIFY Access\_Event = GRANTED
4. VERIFY Access\_Event\_Time = (the time that the credential was presented)
5. VERIFY Access\_Event\_Credential = C1
6. VERIFY Access\_Event\_Tag = EventTag + 1

### 7.3.2.X56.3 Denied Access Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that a credential that is not allowed access to this access point at this time is denied access. There are a number of reasons why a credential may be denied access and this test tests the situations which must be supported by the access point.

Test Concept: To test that a credential, which is not allowed access to this access point, is presented at the access point with the result that access is allowed and the appropriate access event is generated.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The vendor shall provide a set of credentials which correspond to Access Credential objects configured such that access to the access point shall be denied for the following reasons:
  - a. DENIED\_POINT\_NO\_ACCESS\_RIGHTS = C1
  - b. DENIED\_NO\_ACCESS\_RIGHTS = C2
  - c. DENIED\_ZONE\_NO\_ACCESS\_RIGHTS = C3
  - d. DENIED\_CREDENTIAL\_NOT\_YET\_ACTIVE = C4
  - e. DENIED\_CREDENTIAL\_EXPIRED = C5
  - f. DENIED\_CREDENTIAL\_MANUAL\_DISABLE = C6

- g. DENIED\_INCORRECT\_AUTHENTICATION\_FACTOR = C7
- h. DENIED\_OUT\_OF\_TIME\_RANGE = C8
- i. DENIED\_THREAT\_LEVEL = C9
- j. DENIED\_PASSBACK = C10
- k. DENIED\_UNEXPECTED\_LOCATION\_USAGE = C11
- l. DENIED\_MAX\_ATTEMPTS = C12
- m. DENIED\_AUTHENTICATION\_FACTOR\_LOST = C13
- n. DENIED\_AUTHENTICATION\_FACTOR\_STOLEN = C14
- o. DENIED\_AUTHENTICATION\_FACTOR\_DAMAGED = C15
- p. DENIED\_AUTHENTICATION\_FACTOR\_DESTROYED = C16
- q. DENIED\_AUTHENTICATION\_FACTOR\_DISABLED = C17
- r. DENIED\_AUTHENTICATION\_FACTOR\_ERROR = C18
- s. DENIED\_CREDENTIAL\_UNASSIGNED = C19
- t. DENIED\_CREDENTIAL\_NOT\_PROVISIONED = C20
- u. DENIED\_CREDENTIAL\_LOCKOUT = C21
- v. DENIED\_CREDENTIAL\_MAX\_DAYS = C22
- w. DENIED\_CREDENTIAL\_MAX\_USES = C23
- x. DENIED\_CREDENTIAL\_DISABLED = C24
- y. DENIED\_LOCKOUT = C25

Note to Tester: if the IUT does not support any of the above denial reasons then the corresponding credentials are not required to be supplied.

Test Steps:

1. REPEAT C = (C1...C25) DO {
  - READ EventTag = Access\_Event\_Tag
  - MAKE (present the credential corresponding to C at the credential reader for this access point)
  - VERIFY Access\_Event = (denied reason corresponding to credential C)
  - VERIFY Access\_Event\_Time = (the time that credential C was presented)
  - VERIFY Access\_Event\_Credential = C
  - VERIFY Access\_Event\_Tag = EventTag + 1
- verify unknown credential event
2. READ EventTag = Access\_Event\_Tag
3. MAKE (present a credential which does not correspond to any configured Access Credential object at the credential reader for this access point)
4. VERIFY Access\_Event = DENIED\_UNKNOWN\_CREDENTIAL
5. VERIFY Access\_Event\_Time = (the time that the credential was presented)
6. VERIFY (Access\_Event\_Credential = (4194303, ?, 4194303))
7. VERIFY Access\_Event\_Tag = EventTag + 1

### 7.3.2.X56.4 Authorization Mode Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify each authorization mode supported by this IUT.

Test Concept:

For each authorization mode supported by the IUT a valid credential is presented at the access point to verify that the appropriate action is taken.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
- b) An active credential with no valid access rights shall be represented by Access Credential object C2.

Note: If the VERIFICATION\_REQUIRED or AUTHORIZATION\_DELAYED mode is supported the vendor must provide a mechanism for external verification to be performed.

Test Steps:

-- verify GRANT\_ACTIVE mode

1. IF (GRANT\_ACTIVE is supported) THEN
  - READ EventTag = Access\_Event\_Tag
  - WRITE Authorization\_Mode = GRANT\_ACTIVE
  - MAKE (present credential C2 at credential reader for this access point)
  - VERIFY Access\_Event = GRANTED
  - VERIFY Access\_Event\_Time = (the time that credential C2 was presented)
  - VERIFY Access\_Event\_Credential = C2
  - VERIFY Access\_Event\_Tag = EventTag + 1

-- verify DENY\_ALL mode

2. IF (DENY\_ALL is supported) THEN
  - READ EventTag = Access\_Event\_Tag
  - WRITE Authorization\_Mode = DENY\_ALL
  - MAKE (present credential C1 at credential reader for this access point)
  - VERIFY Access\_Event = DENIED\_DENY\_ALL
  - VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
  - VERIFY Access\_Event\_Credential = C1
  - VERIFY Access\_Event\_Tag = EventTag + 1

-- verify VERIFICATION\_REQUIRED mode (verification authorized)

3. IF (VERIFICATION\_REQUIRED is supported) THEN
  - READ EventTag = Access\_Event\_Tag
  - WRITE Authorization\_Mode = VERIFICATION\_REQUIRED
  - MAKE (present credential C1 at credential reader for this access point)
  - VERIFY Access\_Event = VERIFICATION\_REQUIRED
  - VERIFY Access\_Event\_Time = (the time that credential C1 was presented most recently)
  - VERIFY Access\_Event\_Credential = C1
  - VERIFY Authentication\_Status = WAITING\_FOR\_VERIFICATION
  - MAKE (external verification process grants access)
  - VERIFY Access\_Event = GRANTED
  - VERIFY Access\_Event\_Time = (the time that verification process granted access)
  - VERIFY Access\_Event\_Credential = C1
  - VERIFY Access\_Event\_Tag = EventTag + 1

-- verify VERIFICATION\_REQUIRED mode (verification denied)

- READ EventTag = Access\_Event\_Tag
- WRITE Authorization\_Mode = VERIFICATION\_REQUIRED
- MAKE (present credential C1 at credential reader for this access point)
- VERIFY Access\_Event = VERIFICATION\_REQUIRED
- VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
- VERIFY Access\_Event\_Credential = C1
- VERIFY Authentication\_Status = WAITING\_FOR\_VERIFICATION
- MAKE (external verification process denies access)
- VERIFY Access\_Event = DENIED\_VERIFICATION\_FAILED

```

VERIFY Access_Event_Time = (the time that verification process denied access)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag + 1

```

```

-- verify VERIFICATION_REQUIRED mode (verification timeout)
READ EventTag = Access_Event_Tag
WRITE Authorization_Mode = VERIFICATION_REQUIRED
MAKE (present credential C1 at credential reader for this access point)
VERIFY Access_Event = VERIFICATION_REQUIRED
VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
MAKE (external verification process does not respond within verification time)
WAIT Verification_Time
VERIFY Access_Event = DENIED_VERIFICATION_TIMEOUT
VERIFY Access_Event_Time = (the time that verification process timed out)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag = EventTag + 1

```

```

-- verify AUTHORIZATION_DELAYED mode (access granted)
4. IF (AUTHORIZATION_DELAYED is supported) THEN
 WRITE Authorization_Mode = AUTHORIZATION_DELAYED
 MAKE (present credential C1 at credential reader for this access point)
 VERIFY Access_Event = AUTHORIZATION_DELAYED
 VERIFY Access_Event_Time = (the time that credential C1 was presented)
 VERIFY Access_Event_Credential = C1
 MAKE (external verification process does not respond within verification time)
 WAIT Verification_Time
 VERIFY Access_Event = GRANTED
 VERIFY Access_Event_Time = (the time that verification process timed out)
 VERIFY Access_Event_Credential = C1

```

```

-- verify AUTHORIZATION_DELAYED mode (access denied)
WRITE Authorization_Mode = AUTHORIZATION_DELAYED
READ EventTag = Access_Event_Tag
MAKE (present credential C1 at credential reader for this access point)
VERIFY Access_Event = AUTHORIZATION_DELAYED
VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
MAKE (external verification process denies access)
VERIFY Access_Event = DENIED_VERIFICATION_FAILED
VERIFY Access_Event_Time = (the time that verification process denied access)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag = EventTag + 1

```

```

-- verify NONE mode
5. IF (NONE is supported) THEN
 WRITE Authorization_Mode = NONE
 WAIT Internal Processing Fail Time
 VERIFY Authentication_Status = DISABLED

```

### 7.3.2.X56.5 Access Rights Exemptions Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify the access rights exemption functionality.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with no access rights shall be represented by Access Credential object C1.
- b) The Authorization\_Exemption list of C1 shall be empty.

Test Steps:

-- verify access is denied for the credential

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY Access\_Event = DENIED\_NO\_ACCESS\_RIGHTS
3. VERIFY Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY Access\_Event\_Credential = C1

-- verify access is granted for the credential when the master exemption set to TRUE

5. MAKE C1, Authorization\_Exemption = (ACCESS\_RIGHTS)
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY Access\_Event = GRANTED
8. VERIFY Access\_Event\_Time = (the time that the credential was presented)
9. VERIFY Access\_Event\_Credential = C1

### 7.3.2.X56.6 Change Authentication Policy Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Active\_Authentication\_Policy property of the Access Point object accepts valid authentication policy values and does not accept invalid ones. It also verifies that an error response is returned if the authentication policy is changed to a non-existent policy number.

Test Concept: The Active\_Authentication\_Policy is written with values of 1 to X, where X is the number of valid authentication policies to verify that the value is accepted. Then it is written with a value larger than X to verify that the value is rejected with a VALUE\_OUT\_OF\_RANGE error. Finally, it is written with a value of 0 to verify that the value is also rejected with the same error.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The IUT shall be configured with at least one active authentication policy.
- b) All authentication policies shall be valid policies.

Test Steps:

-- verify that the active authentication policy can set to all valid policies

1. READ Count = Number\_Of\_Authentication\_Policies
2. REPEAT X = (1 to Count) DO {  
     WRITE Active\_Authentication\_Policy = X  
     VERIFY Active\_Authentication\_Policy = X  
   }

-- verify that writing an invalid authentication policy to active authentication policy results in a reject

3. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Access\_Point object,  
     'Property Identifier' = Active\_Authentication\_Policy,  
     'Property Value' = (any value larger than Count)
4. RECEIVE BACnet-Error-PDU,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE

-- verify that writing 0 to the authentication policy results in a reject

- 5 TRANSMIT WriteProperty-Request,  
   'Object Identifier' =       Access\_Point object,  
   'Property Identifier' =    Active\_Authentication\_Policy,  
   'Property Value' =        0
- 6 RECEIVE BACnet-Error-PDU,  
   Error Class =           PROPERTY,  
   Error Code =            VALUE\_OUT\_OF\_RANGE

### 7.3.2.X56.7 Lockout State Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that access is denied for any credential when the access point is in the lockout state. To verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state. To verify that the lockout will automatically relinquish after the specified time.

Test Concept: A credential which will result in denied access is repeatedly presented at the access point until the access point becomes locked out. When the access point becomes locked valid credentials will also be denied access until the lockout relinquish time has expired.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The Max\_Failed\_Attempts property, if present, has a value greater than 0
- b) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
- c) An active credential with no valid access rights for the access point shall be represented by Access Credential object C2.
- d) The Failed\_Attempts\_Events list, if present, shall have at least one entry corresponding to the reason why C2 is denied access.
- e) The Lockout\_Relinquish\_Time has a value greater than 0

Test Steps:

-- verify that valid credentials are denied when the Lockout property is TRUE

1. WRITE Lockout = TRUE
2. WAIT **Internal Processing Fail Time**
3. VERIFY Access\_Event = LOCKOUT\_OTHER
4. VERIFY Access\_Event\_Time = (the time that TRUE was written to the Lockout property)
5. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
6. MAKE (present credential C1 at credential reader for this access point)
7. VERIFY Access\_Event = DENIED\_LOCKOUT
8. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
9. VERIFY Access\_Event\_Credential = C1

-- verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state

10. WRITE Lockout = FALSE
11. WAIT **Internal Processing Fail Time**
12. VERIFY Access\_Event = LOCKOUT\_RELINQUISH
13. VERIFY Access\_Event\_Time = (the time that FALSE was written to the Lockout property)
14. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
15. IF (Failed\_Attempts and Max\_Failed\_Attempts are supported) THEN
  - REPEAT X= (1 to Max\_Failed\_Attempts + 1) DO {
  - READ FailedAttempts = Failed\_Attempts
  - MAKE (present credential C2 at credential reader for this access point)

```

 VERIFY (Failed_Attempts = FailedAttempts + 1)
}

```

16. VERIFY (Lockout = TRUE)
17. VERIFY (Access\_Event = LOCKOUT\_MAX\_ATTEMPTS)
18. VERIFY (Access\_Event\_Time = the time that Lockout was set to TRUE)
19. VERIFY (Access\_Event\_Credential = C2)
20. MAKE (present credential C1 at credential reader for this access point)
21. VERIFY (Access\_Event = DENIED\_LOCKOUT)
22. VERIFY (Access\_Event\_Time = the time that credential C1 was presented)
23. VERIFY (Access\_Event\_Credential = C1)

-- verify that the lockout will automatically relinquish after the specified time

24. WAIT Lockout\_Relinquish\_Time
25. VERIFY (Lockout = FALSE)
26. VERIFY (Access\_Event = LOCKOUT\_RELINQUISHED)
27. VERIFY (Access\_Event\_Time = the time that Lockout was set to FALSE)
28. VERIFY Access\_Event\_Credential = (4194303, ?, 4194303)
29. MAKE (present credential C1 at credential reader for this access point)
30. VERIFY (Access\_Event = GRANTED)
31. VERIFY (Access\_Event\_Time = the time that credential C1 was presented)
32. VERIFY (Access\_Event\_Credential = C1)

### 7.3.2.X56.8 Threat Level Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify Threat\_Level is used to Grant or Deny access based on the Threat\_Authority of the Access Credential

Test Concept: Vary the Threat\_Level of the access point to be lower or equal than the Threat\_Authority of the credential to verify that access is granted at this access point. Change the Threat\_Level of the access point to the higher than the Threat\_Authority of the credential to verify that access is denied.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1. This credential shall have a Threat\_Authority of X, where  $0 < X < 100$ .

Test Steps:

-- verify that a credential with threat authority greater than threat level of the access point is granted access

1. WRITE Threat\_Level = (any value less than X)
2. MAKE (present credential C1 at credential reader for this access point)
3. VERIFY Access\_Event = GRANTED
4. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
5. VERIFY Access\_Event\_Credential = C1

-- verify that a credential with threat authority equal to the threat level of the access point is granted access

6. WRITE Threat\_Level = X
7. MAKE (present credential C1 at credential reader for this access point)
8. VERIFY Access\_Event = GRANTED
9. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
10. VERIFY Access\_Event\_Credential = C1

-- verify that a credential with threat authority less than the threat level of the access point is denied access

11. WRITE Threat\_Level = (any value greater than X)



12. MAKE (present credential C1 at credential reader for this access point)
13. VERIFY Access\_Event = DENIED\_THREAT\_LEVEL
14. VERIFY Access\_Event\_Time = (the time that credential C1 was presented)
15. VERIFY Access\_Event\_Credential = C1

### 7.3.2.X56.9 Denied Access Occupancy Upper Limit Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify occupancy counting, occupancy restrictions and occupancy exemptions.

Test Concept: When occupancy counting is enabled and a valid credential is presented at the access point then this test verifies that access is granted only if the occupancy limits are not violated. If the occupancy limits are violated then access is denied. If the credential has an occupancy exemption then the credential will be granted access regardless of the occupancy count of the zone.

Configuration Requirements:

See 7.3.2.X56. This test requires the following additional configuration:

- a) The Access Point object is configured to be an entry access point to an access zone which is represented by Access Zone object Z1.
- b) The Occupancy\_Upper\_Limit property of Z1 shall have the value X, where  $X > 0$ .
- c) A number of active credentials all with valid access rights for the access point shall be represented by Access Credential objects C1...C(X+1) which shall be configured such that each has valid access to the access point.
- d) The Occupancy\_Upper\_Limit\_Enforced property shall have a value of TRUE.
- e) The Occupancy\_Count\_Adjust property shall have a value of TRUE.
- f) The Occupancy\_Count property of Z1 shall have the value 0.
- g) The Occupancy\_Count\_Enable property of Z1 shall have a value of TRUE.
- h) The Authorization\_Exemptions property, if it exists, of C(X+1) shall be empty.

Test Steps:

-- verify that a credential with valid access is granted access while the occupancy count of the zone is less than the Occupancy\_Upper\_Limit property in the zone

1. REPEAT C = (C1...CX) DO {
  - READ Count = Z1, Occupancy\_Count
  - MAKE (present credential C at credential reader for this access point)
  - VERIFY Z1,Occupancy\_Count = (Count +1)
  - VERIFY Access\_Event = GRANTED
  - VERIFY Access\_Event\_Time = (the time that credential C was presented)
  - VERIFY Access\_Event\_Credential = C

-- verify that a credential with valid access is denied access when the occupancy count of the zone becomes greater than the Occupancy upper limit

2. READ Count = Z1, Occupancy\_Count
3. VERIFY Z1,Occupancy\_Upper\_Limit = Count
4. MAKE (present credential C(X+1) at credential reader for this access point)
5. VERIFY Access\_Event = DENIED\_UPPER\_OCCUPANCY\_LIMIT
6. VERIFY Access\_Event\_Time = (the time that credential C(X+1) was presented)
7. VERIFY Access\_Event\_Credential = C(X+1)
8. VERIFY Z1,Occupancy\_Count = Count

-- verify that a credential with valid access and an occupancy exemption is granted access when the occupancy count of the zone becomes greater than the Occupancy upper limit

9. IF (the Authorization\_Exemption property is not supported or the OCCUPANCY\_CHECK exemption is not supported)  
THEN {
  10. READ Count = Z1, Occupancy\_Count
  11. VERIFY Z1,Occupancy\_Upper\_Limit = Count
  12. WRITE C(X+1), Authorization\_Exemptions = (OCCUPANCY\_CHECK)
  13. MAKE (present credential C(X+1) at credential reader for this access point)
  14. VERIFY Access\_Event = GRANTED
  15. VERIFY Access\_Event\_Time = (the time that credential C(X+1) was presented)
  16. VERIFY Access\_Event\_Credential = C(X+1)
  17. VERIFY Z1,Occupancy\_Count = Count + 1

### 7.3.2.X56.10 Denied Access Disabled Credential Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To test that a credential is denied access at an access point when the credential is disabled even when it has valid access rights.

Test Concept: A disabled credential is presented at an access point and access is denied. For testing purposes, the credential shall be disabled by setting the Expiration\_Time to a time in the past.

Configuration Requirements:

See 7.3.2.X56.1. This test requires the following additional configuration:

- a) An access credential with valid access to AP1 shall be represented by Access Credential C1.
- b) The Credential\_Status property shall have the value ACTIVE.
- c) The Reason\_For\_Disable property shall be empty.

Test Steps:

- test granted access when credential is active
  1. VERIFY Reason\_For\_Disable = ( )
  2. VERIFY Credential\_Status = ACTIVE
  3. MAKE (present credential C1 at credential reader for access point AP1)
  4. VERIFY AP1,Access\_Event = GRANTED
  5. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
  6. VERIFY AP1,Access\_Event\_Credential = C1
- test denied access when credential is inactive
  7. MAKE (Expiration\_Time = time < current time)
  8. VERIFY Reason\_For\_Disable = (DISABLED\_EXPIRED)
  9. VERIFY Credential\_Status = INACTIVE
  10. MAKE (present credential C1 at credential reader for access point AP1)
  11. VERIFY AP1,Access\_Event = DENIED\_CREDENTIAL\_EXPIRED
  12. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
  13. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X57 Access Zone Object Tests

Many of the tests for the Access Zone object require interactions from other BACnet access control objects as a result of a valid credential being processed. The required and optional BACnet object types are shown in figure X2.

The Access Zone is defined by list of Access Points that act as entry and exit points of the zone. The configuration requires at least one Access Point object which is configured to be an entry access point and at least one Access Point that is configured to be an exit access point. Each Access Point object which is part of the Access Zone must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

Properties in the Access Zone are written by the corresponding Access Point object when a valid credential is processed. The vendor must configure the IUT to have a sufficient number of valid credentials for the test being executed. Each credential must have an associated Access Credential object. Each Access Credential must have an associated Access Rights object that provides valid access to the zone.

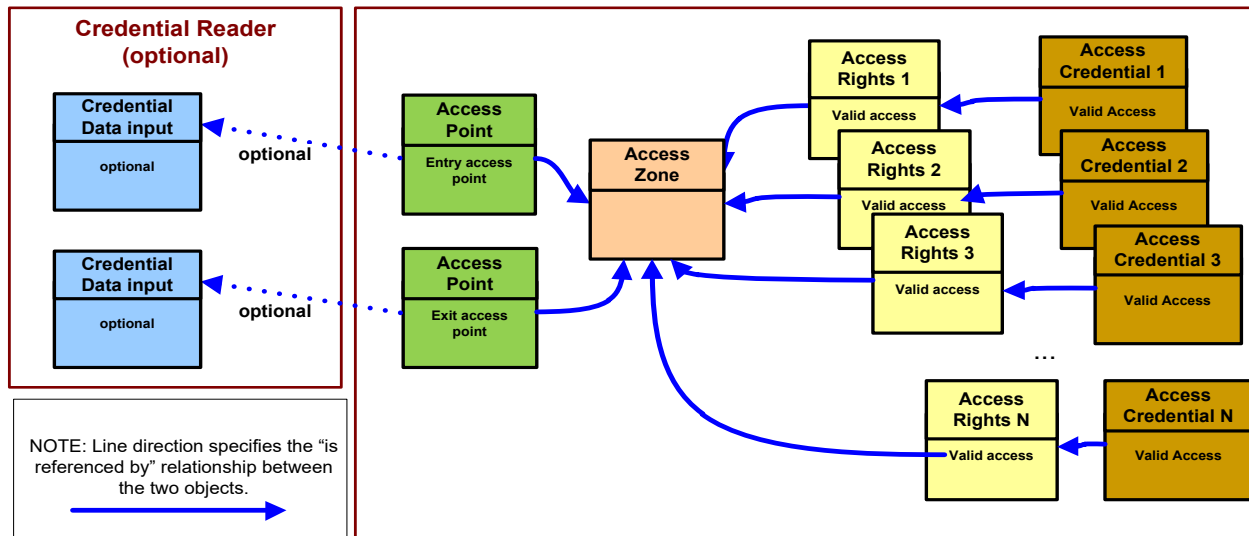


Figure X2: Objects used for testing the Access Zone object

### 7.3.2.X57.1 Occupancy State Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: This test verifies that the occupancy state reflects the occupancy conditions of the zone.

Test Concept:

Adjust the occupancy count of the zone to levels above, at and below the Occupancy\_Upper\_Limit and Occupancy\_Lower\_Limit and verify that the Occupancy\_State has the appropriate value.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- The Access Zone shall not be out of service.
- Occupancy\_Count\_Enable shall have a value of TRUE
- The Occupancy\_Upper\_Limit shall have a value X where  $X > 0$ .
- If the property is supported, the Occupancy\_Lower\_Limit shall have a value Y where  $0 < Y < X$ .

Test Steps:

-- test normal case where occupancy count is between the upper and lower limit

- WRITE Adjust\_Value = 0
- VERIFY Occupancy\_Count = 0
- WRITE Adjust\_Value = X-1
- VERIFY Occupancy\_Count = (X-1)
- VERIFY Occupancy\_State = NORMAL

-- verify the case where occupancy count is at and above the upper limit

6. WRITE Adjust\_Value = 1
  7. VERIFY Occupancy\_Count = X
  8. VERIFY Occupancy\_State = AT\_UPPER\_LIMIT
  9. WRITE Adjust\_Value = 1
  10. VERIFY Occupancy\_Count = X+1
  11. VERIFY Occupancy\_State = ABOVE\_UPPER\_LIMIT
- verify the case where occupancy count is at and above the lower limit
12. WRITE Adjust\_Value = 0
  13. VERIFY Occupancy\_Count = 0
  14. VERIFY Occupancy\_State = BELOW\_LOWER\_LIMIT
  15. WRITE Adjust\_Value = Y
  16. VERIFY Occupancy\_Count = Y
  17. VERIFY Occupancy\_State = AT\_LOWER\_LIMIT
  18. WRITE Adjust\_Value = 1
  19. VERIFY Occupancy\_Count = (Y+1)
  20. VERIFY Occupancy\_State = NORMAL
- verify occupancy state when occupancy counting is disabled
21. WRITE Occupancy\_Count\_Enable = FALSE
  22. VERIFY Occupancy\_State = DISABLED

### 7.3.2.X57.2 Occupancy Counting Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the occupancy counting functionality

Test Concept:

Present a credential at the entry and exit access points and test that the occupancy count is properly changed.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) Occupancy\_Count\_Enable shall have a value of TRUE
- c) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Occupancy\_Count\_Adjust property for this object shall have a value of TRUE.
- d) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Occupancy\_Count\_Adjust property for this object shall have a value of TRUE.
- e) Two active credentials with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential objects C1 and C2.

Test Steps:

-- verify that presenting a credential at the entry access point increases the occupancy count

1. WRITE Adjust\_Value = 0
2. VERIFY Occupancy\_Count = 0
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY Occupancy\_Count = 1
5. MAKE (present credential C2 at credential reader for access point AP1)
6. VERIFY Occupancy\_Count = 2

-- verify that presenting a credential at the exit access point decreases the occupancy count

7. MAKE (present credential C1 at credential reader for access point AP2)
8. VERIFY Occupancy\_Count = 1

9. MAKE (present credential C2 at credential reader for access point AP2)
10. VERIFY Occupancy\_Count = 0

### 7.3.2.X57.3 Keeping Track of Credentials Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the zone can keep track of the current credentials in the zone.

Test Concept:

Present a valid credential at an entry access point to this access zone. The object reference of the credential should be in the credentials in zone list. When the credential is presented to an exit access point for this zone the reference to the credential in the credential in zone list should be removed.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) The Credentials\_In\_Zone property shall be an empty list.
- c) An access point which is an entry point to this zone shall be represented by Access Point object AP1.
- d) An access point which is an exit point to this zone shall be represented by Access Point object AP2.
- e) Two active credential with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential object C1 and C2.

Test Steps:

- verify that presenting a credential at an entry access point puts it in the Credentials\_In\_Zone list
1. VERIFY Credentials\_In\_Zone = ( )
  2. MAKE (present credential C1 at credential reader for access point AP1)
  3. VERIFY Credentials\_In\_Zone = (C1)
  4. VERIFY Last\_Credential\_Added = C1
  5. VERIFY Last\_Credential\_Added\_Time = (the time that credential C1 was presented at AP1)
  6. MAKE (present credential C2 at credential reader for access point AP1)
  7. VERIFY Credentials\_In\_Zone = (C1,C2)
  8. VERIFY Last\_Credential\_Added = C2
  9. VERIFY Last\_Credential\_Added\_Time = (the time that credential C2 was presented at AP1)
- verify that presenting a credential at an exit access point removes it from the Credentials\_In\_Zone list
10. MAKE (present credential C1 at credential reader for access point AP2)
  11. VERIFY Credentials\_In\_Zone = (C2)
  12. VERIFY Last\_Credential\_Removed = C1
  13. VERIFY Last\_Credential\_Removed\_Time = the time that credential C1 was presented at AP2
  14. MAKE (present credential C2 at credential reader for access point AP2)
  15. VERIFY Credentials\_In\_Zone = ( )
  16. VERIFY Last\_Credential\_Removed = C2
  17. VERIFY Last\_Credential\_Removed\_Time = the time that credential C2 was presented at AP2

### 7.3.2.X57.4 Passback Mode Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify the passback functionality and passback exemption. .

Test Concept: A valid credential is presented at the entry access point to this access zone. When the credential is presented a second time a passback notification should be generated and if the passback mode is set to hard passback then access to the zone should be denied. If the credential has a passback exemption then access will never be denied due to a passback violation.

Configuration Requirements:

See 7.3.2.X57. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Authorization\_Mode property of AP1 shall have the value Authorize.
- c) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Authorization\_Mode property of AP2 shall have the value Authorize.
- d) An active credential with valid access rights for the access point AP1 and AP2 shall be represented by Access Credential object C1.
- e) The C1.Authorization\_Exemptions list shall be empty.

Test Steps:

-- verify soft passback mode

1. MAKE (Passback\_Mode = SOFT\_PASSBACK)
2. READ EventTag = AP1,Access\_Event\_Tag
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1, Access\_Event = GRANTED
5. VERIFY AP1,Access\_Event\_Time = (the time that credential C1 was presented)
6. VERIFY AP1,Access\_Event\_Credential = C1
7. VERIFY AP1,Access\_Event\_Tag = (EventTag + 1)
8. MAKE (present credential C1 at credential reader for access point AP1)
9. VERIFY AP1,Access\_Event = GRANTED
10. VERIFY (AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)
11. VERIFY AP1,Access\_Event\_Credential = C1
12. VERIFY AP1,Access\_Event\_Tag = (EventTag +2)
13. MAKE (present credential C1 at credential reader for access point AP2)

-- verify hard passback mode

14. MAKE (Passback\_Mode = HARD\_PASSBACK)
15. READ EventTag = AP1,Access\_Event\_Tag
16. MAKE (present credential C1 at credential reader for access point AP1)
17. VERIFY AP1,Access\_Event = GRANTED
18. VERIFY AP1,Access\_Event\_Time = (the time that credential C1 was presented most recently)
19. VERIFY AP1,Access\_Event\_Credential = C1
20. VERIFY AP1,Access\_Event\_Tag = EventTag + 1
21. MAKE (present credential C1 at credential reader for access point AP1)
22. VERIFY AP1,Access\_Event = DENIED\_PASSBACK
23. VERIFY AP1,Access\_Event\_Time = (the time that credential C1 was presented most recently)
24. VERIFY AP1,Access\_Event\_Credential = C1
25. VERIFY AP1,Access\_Event\_Tag = (EventTag + 2)

-- verify passback timeout

26. WAIT (Passback\_Timeout)
27. MAKE (present credential C1 at credential reader for access point AP1)
28. VERIFY AP1,Access\_Event = GRANTED
29. VERIFY (AP1.Access\_Event\_Time = the time that credential C1 was presented)
30. VERIFY (AP1.Access\_Event\_Credential = C1)

-- verify hard passback off

31. MAKE (Passback\_Mode = PASSBACK\_OFF)
32. READ EventTag = AP1,Access\_Event\_Tag

33. MAKE (present credential C1 at credential reader for access point AP1)
  34. VERIFY AP1.Access\_Event = GRANTED
  35. VERIFY AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)
  36. VERIFY AP1.Access\_Event\_Credential = C1
  37. VERIFY AP1.Access\_Event\_Tag = EventTag + 1
  38. MAKE (present credential C1 at credential reader for access point AP1)
  39. VERIFY AP1.Access\_Event = GRANTED
  40. VERIFY AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)
  41. VERIFY (AP1.Access\_Event\_Credential = C1
  42. VERIFY AP1.Access\_Event\_Tag = (EventTag + 2)
- verify passback exemption
43. MAKE (Passback\_Mode = HARD\_PASSBACK)
  44. MAKE (C1, Authorization\_Exemption = (PASSBACK))
  45. READ EventTag = AP1.Access\_Event\_Tag
  46. MAKE (present credential C1 at credential reader for access point AP1)
  47. VERIFY AP1.Access\_Event = GRANTED
  48. VERIFY AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)
  49. VERIFY AP1.Access\_Event\_Credential = C1
  50. VERIFY AP1.Access\_Event\_Tag = EventTag + 1
  51. MAKE (present credential C1 at credential reader for access point AP1)
  52. VERIFY AP1.Access\_Event = GRANTED
  53. VERIFY AP1.Access\_Event\_Time = (the time that credential C1 was presented most recently)
  54. VERIFY AP1.Access\_Event\_Credential = C1
  55. VERIFY AP1.Access\_Event\_Tag = (EventTag + 2)

### 7.3.2.X59 Access Rights Object Tests

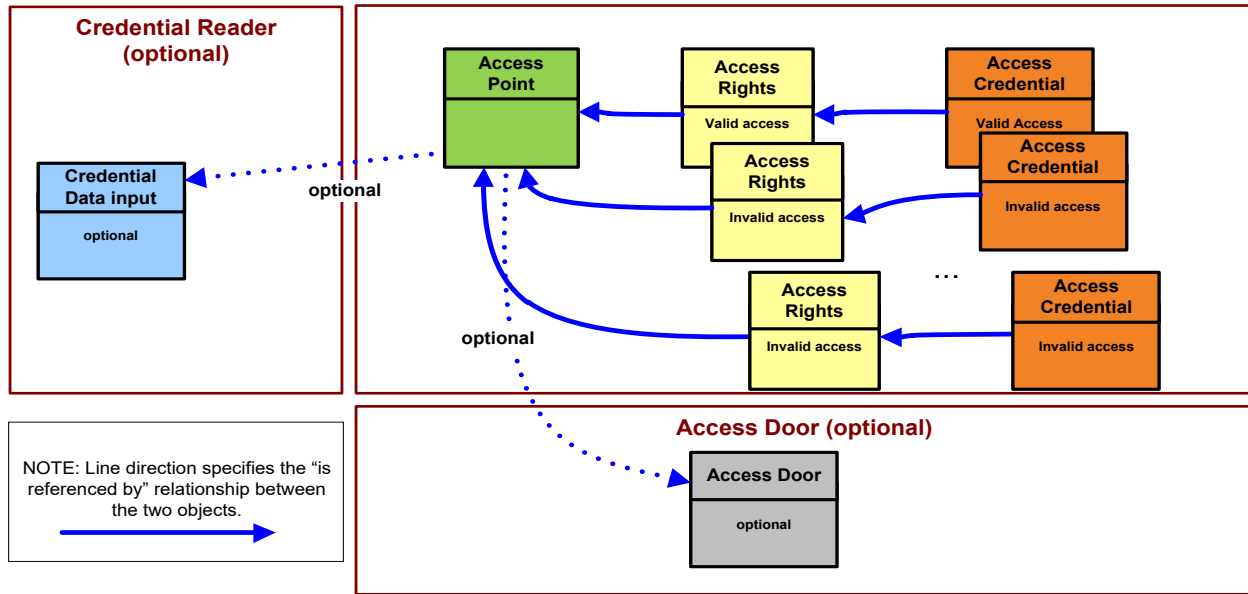
Many of the tests for the Access Rights object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in figure d.

The Access Rights object specifies both positive and negative access rights. This list of access rights is used by the Access Point object to determine the access decision. To test the access rights the vendor must configure the IUT to have at least one Access Point object which is referenced in the Access Rights objects used for this test.

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined the IUT shall provide a method to show the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.



**Figure d:** Objects used for testing the Access Rights object

### 7.3.2.X59.1 Enable Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: This test verifies that the access rights object does not allow access when the Enable property is FALSE.

Test Concept: Present a valid credential at the access point. Since the access rights object that provides the access rights to the access point is not enabled access shall be denied.

Configuration Requirements:

See 7.3.2.X59. This test requires the following additional configuration:

- An access point shall be represented by Access Point object AP1.
- An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- AR1 shall have the Enable property set to TRUE
- An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

- verify access granted with this access rights object when enable property is TRUE

  1. MAKE (present credential C1 at credential reader for access point AP1)
  2. VERIFY AP1,Access\_Event = GRANTED
  3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
  4. VERIFY AP1,Access\_Event\_Credential = C1

- verify access denied with this access rights object when Enable property is FALSE

  5. WRITE AR1, Enable = FALSE
  6. MAKE (present credential C1 at credential reader for access point AP1)
  7. VERIFY AP1,Access\_Event = DENIED\_NO\_ACCESS\_RIGHTS
  8. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
  9. VERIFY AP1,Access\_Event\_Credential = C1



### 7.3.2.X59.2 Negative Rules Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Configuration Requirements:

See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies negative access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = DENIED\_POINT\_NO\_ACCESS\_RIGHTS
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X59.3 Positive Access Rules Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the positive access rules explicitly enable access to an access point.

Test Concept: Present a credential at the access point. The access point at which the credential is trying to get entry is in the positive rules list and that rule is valid at the current time. Access to the access point is granted.

Configuration Requirements:

See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access\_Event = GRANTED
3. VERIFY AP1,Access\_Event\_Time = (the time that the credential was presented)
4. VERIFY AP1,Access\_Event\_Credential = C1

### 7.3.2.X59.4 Accompaniment Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the accompaniment functionality works properly.

Test Concept: Present a credential which needs accompaniment at the access point. Wait for the accompaniment time, as specified in the access point. When this times out the credential should be denied entry to the access point. Present the first credential again at the access point. Present a second credential at the access point which has the rights to accompany the first credential. Access should be granted to the first credential.

Configuration Requirements:

See 7.3.2.X59. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1. If the Accompaniment\_Time property is supported it shall be set to a value > 0.

- b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1 that requires accompaniment.
- d) An active credential which has access rights which allow it to accompany a credential with access rights specified by AR1 through AP1, shall be represented by Access Credential object C2.
- e) An active credential which has valid access rights to AP1 but which does not meet the accompaniment requires of AR1, shall be represented by Access Credential object C3.

Test Steps:

-- valid access through the access point

1. READ Tag = Access\_Event\_Tag
2. MAKE (present credential C1 at credential reader for access point AP1)
3. MAKE (present credential C2 at credential reader for access point AP1)
4. VERIFY AP1,Access\_Event = GRANTED
5. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access\_Event\_Credential = C1
7. VERIFY AP1,Access\_Event\_Tag = (Tag + 1)

-- no accompaniment presented

8. MAKE (present credential C1 at credential reader for access point AP1)
9. WAIT AP1,Accompaniment\_Time
10. VERIFY AP1,Access\_Event = DENIED\_NO\_ACCOMPANIMENT
11. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
12. VERIFY AP1,Access\_Event\_Credential = C1
13. VERIFY Access\_Event\_Tag = (Tag + 2)

-- Invalid accompaniment

14. MAKE (present credential C1 at credential reader for access point AP1)
15. MAKE (present credential C3 at credential reader for access point AP1)
16. VERIFY AP1,Access\_Event = DENIED\_INCORRECT\_ACCOMPANIMENT
17. VERIFY AP1,Access\_Event\_Time = (the time that the credential C1 was presented)
18. VERIFY AP1,Access\_Event\_Credential = C1
19. VERIFY AP1,Access\_Event\_Tag = (Tag + 3)

### 7.3.2.X60 Access Credential Object Tests

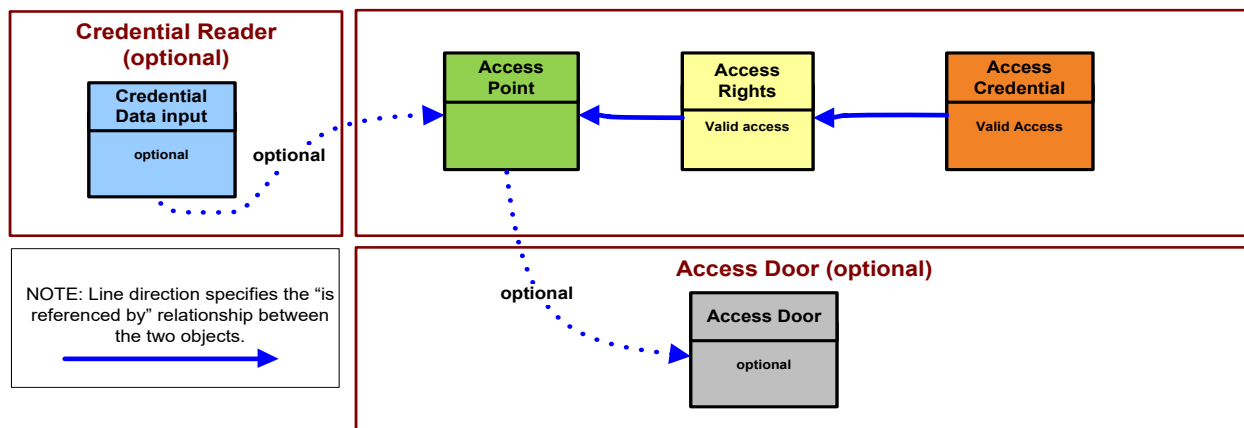
Many of the tests for the Access Credential object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in figure e.

The Access Credential defines a list of credential values (authentication factors) that are used to identify the credential. The credential values are used by the Access Point object in determining the access control decision. To test the Access Credential object the vendor must configure the IUT to have at least one Access Point object. Each Access Credential used in this test must reference an Access Rights object which references the Access Point object

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined the IUT shall provide a method to show the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.



**Figure e:** BACnet Objects used for testing the Access Credential object

### 7.3.2.X60.1 Credential Status, Credential Disable and Reason for Disable Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify the ability to disable the credential and set the associated reason and to enable the credential.

Test Concept: The credential status is set to INACTIVE and the corresponding reason or reasons are written to the Reason\_For\_Disable list.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- An access credential shall be represented by Access Credential C1.
- The Credential\_Status property shall have the value ACTIVE.
- The Reason\_For\_Disable property shall be empty.

Note: This tests only verifies the most common disable reasons (DISABLED\_MANUAL, DISABLED\_NOT\_YET\_ACTIVE, DISABLED\_EXPIRED).

Test Steps:

- ```
-- test DISABLED_MANUAL
1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. MAKE (add DISABLED_MANUAL to Reason_For_Disable property)
4. VERIFY Reason_For_Disable = (DISABLED_MANUAL)
5. VERIFY Credential_Status = INACTIVE
6. MAKE (remove DISABLED_MANUAL from Reason_For_Disable property)
7. VERIFY Reason_For_Disable = ()
8. VERIFY Credential_Status = ACTIVE

-- test DISABLED_NOT_YET_ACTIVE
9. VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ()
11. MAKE (set Activation_Time = time > current time)
12. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
13. VERIFY Credential_Status = INACTIVE
14. MAKE (set Activation_Time = time < current time)
```

15. VERIFY Reason_For_Disable = ()
16. VERIFY Credential_Status = ACTIVE
- test DISABLED_EXPIRED
17. VERIFY Credential_Status = ACTIVE
18. VERIFY Reason_For_Disable = ()
19. MAKE (set Expiration_Time = time < current time)
20. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
21. VERIFY Credential_Status = INACTIVE
22. MAKE (set Expiration_Time = time > current time)
23. VERIFY Reason_For_Disable = ()
24. VERIFY Credential_Status = ACTIVE

7.3.2.X60.2 Activation Time and Expiration Time Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To test the activation time and expiration time functionality of this object.

Test Concept: The Activation_Time of the credential is set to a time in the future and the credential should be disabled. The Expiration_Time is set to a time in the past and the credential should be disabled.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) The Credential_Status property shall have the value ACTIVE.
- b) The Reason_For_Disable property shall be empty.
- c) The Activation_Time shall have an initial value of 0xFF
- d) The Expiration_Time shall have an initial value of 0xFF.

Test Steps:

- test activation time
1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. MAKE (set Activation_Time = time > current time)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
6. MAKE (set Activation_Time = time < current time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()
- test expiration time
9. VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ()
11. MAKE (Expiration_Time = time < current time)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
14. MAKE (Expiration_Time = time > current time)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ()

7.3.2.X60.3 Disabled Access Rights Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify the enable field disables an access right for this credential when set to FALSE.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential where Assigned_Access_Rights[1].Assigned-Access-Rights = AR1 shall be represented by Access Credential object C1. Assigned_Access_Rights[1].Enable shall be TRUE.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. WRITE Assigned_Access_Rights[1].Enable = FALSE
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY AP1,Access_Event = DENIED_NO_ACCESS_RIGHTS
8. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
9. VERIFY AP1,Access_Event_Credential = C1

7.3.2.X60.4 Days Remaining and Uses Remaining Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To test the days remaining and uses remaining functionality of this object.

Test Concept:

Set the Days_Remaining property to 0. The credential will become inactive and the corresponding reason for disable put in the Reason_For_Disable property.

Set the Uses_Remaining to 0. The credential will become inactive and the corresponding reason for disable will be put in the Reason_For_Disable property.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) The Credential_Status property shall have the value ACTIVE.
- b) The Reason_For_Disable property shall be empty.
- c) Days_Remaining shall have a value of -1 or >0.
- d) Uses_Remaining shall have a value of -1 or >0.

Test Steps:

-- test day remaining

1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. MAKE (Days_Remaining = 0)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_MAX_DAYS)
6. MAKE (Days_Remaining = -1 OR Days_Remaining > 0)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()

-- test uses remaining

9. VERIFY Credential_Status = ACTIVE

10. VERIFY Reason_For_Disable = ()
11. MAKE (Uses_Remaining = 0)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_MAX_USES)
14. MAKE (Uses_Remaining = -1 OR Uses_Remaining > 0)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ()

7.3.2.X60.5 Absentee Limit Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify the absentee limit functionality of this object.

Test Concept: Set the Absentee_Limit property some value ≥ 0 . Use the credential to access an access point. Change the current date to one that is greater than (current date + Absentee_Limit) and attempt to gain access to an access point. Access should be denied because the credential should be disabled due to inactivity.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) Absentee_Limit ≥ 0
- b) The Credential_Status property shall have the value ACTIVE.
- c) The Reason_For_Disable property shall be empty.
- d) Days_Remaining shall have a value > 0 .
- e) Last_Use_Time shall be set to a valid date and time.

Test Steps:

1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. TRANSMIT UTCTimeSynchronization-Request, 'Time' = (any day and time greater than Absentee_Limit days)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_INACTIVITY)

- clear the condition

6. MAKE (set Absentee_Limit > number of days since Last_Use_Time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()

7.3.2.X60.6 Last Access Point, Last Use Time and Last Access Event Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Last Access Point, Last Access Event and Last Use Time properties are updated when this credential is used at an access point.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1.
- b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)

2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. VERIFY Last_Access_Point = AP1
6. VERIFY Last_Use_Time = AP1,Access_Event_Time
7. VERIFY Last_Access_Event = GRANTED

7.3.2.X60.7 Extended Time Enable Test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that a credential used at an access point with the extended flag set to TRUE results in the corresponding door to pulse open for a Pulse_Extended period of time as defined in the door object.

Configuration Requirements:

See 7.3.2.X60. This test requires the following additional configuration:

- a) An access door shall be represented by Access Door object AD1 and Door_Delay_Time shall be 0.
- b) An access point, which controls AD1, shall be represented by Access Point object AP1.
- c) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- d) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.
- e) The Extended_Time_Enable property shall be FALSE.

Test Steps:

-- verify that PULSE_UNLOCK is written when the extended time enable is FALSE

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = PULSE_UNLOCK

-- verify that EXTENDED_PULSE_UNLOCK is written when the extended time enable is TRUE

6. WRITE Extended_Time_Enable = TRUE
7. MAKE (present credential C1 at credential reader for access point AP1)
8. VERIFY AP1,Access_Event = GRANTED
9. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
10. VERIFY AP1,Access_Event_Credential = C1
11. BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = EXTENDED_PULSE_UNLOCK

7.3.2.X61 Credential Data Input Object Tests

The Credential Data Input object type represents a device or process that reads an authentication factor from a physical device such as a card reader, key pad or biometric device. There are countless variations and authentication formats supported for these devices. As such, there is not a standard format or device configuration that can be mandated for these tests.

The vendor must configure the IUT such that the Credential Data Input device can read an authentication factor from the corresponding physical device including setting the Supported_Formats[1] property to the correct authentication factor format (figure f). This configuration is considered to be a local matter and will not be tested.

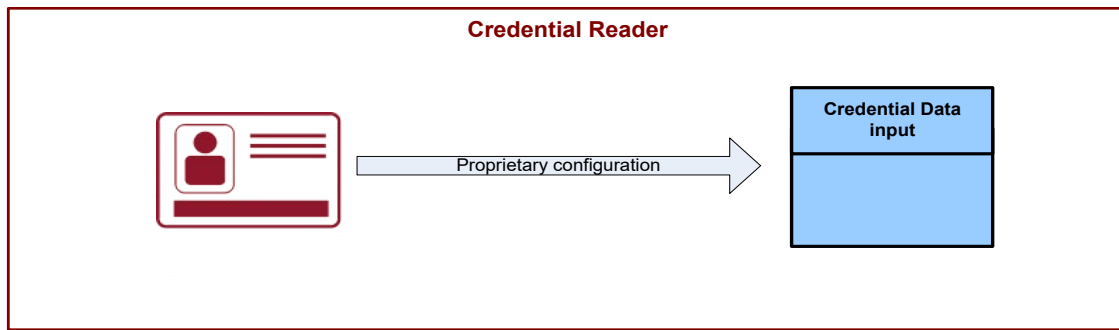


Figure f: Credential Data Input configuration

7.3.2.X61.1 Return from Out Of Service Undefined test

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the Present_Value. Format-Type becomes undefined when out of service is set to false.

Configuration Requirements:

See 7.3.2.X61. This test requires the following additional configuration:

- a) The Out_Of_Service property shall be TRUE.

Test Steps:

1. WRITE Out_Of_Service = FALSE
2. VERIFY Present_Value. Format-Type = UNDEFINED
3. VERIFY Present_Value. Format-Class = 0

7.3.2.X61.2 Read Valid Authentication Factor Test

Purpose: To verify that Present_Value is set to the proper value when an authentication factor with a recognized format is read at the corresponding physical device.

Configuration Requirements:

See 7.3.2.X61. This test requires the following additional configuration:

- a) The Out_Of_Service property shall be FALSE.
- b) Two authentication factors, AF1 and AF2, shall be provided which can be read by the physical device which have the format specified in Supported_Formats[1].

Test Steps:

-- test AF1

1. MAKE (present AF1 at the credential reader)
2. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type
3. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
4. VERIFY Present_Value. Value = the authentication format value of AF1

-- test AF2

5. MAKE (present AF2 at the credential reader)
6. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type
7. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
8. VERIFY Present_Value. Value = the authentication format value of AF2

8. APPLICATION SERVICE INITIATION TESTS

8.1 AcknowledgeAlarm Service Initiation Tests

Dependencies: None.

BACnet Reference Clause: 13.5.

Purpose: To verify that the IUT is capable of acknowledging alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotification services.

Configuration: For this test, the tester shall choose 1 object, O1, in the TD, which is configured to send event notifications to the IUT. The tester places O1 into an alarm state such that the transition requires an acknowledgment.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
 'Initiating Device Identifier' = TD,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),
 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request,
 'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1, or *OFFNORMAL* if S1 is an off-normal state
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgement' = (any valid value)
5. TRANSMIT BACnet-SimpleACK-PDU

8.1.X2 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the 'Initiating Device Identifier' Parameter

Reason for Change: Added a new test ensuring correct behaviour acknowledging an event notification that has been forwarded by a Notification Forwarder.

Purpose: To verify that the IUT is correctly implemented to send AcknowledgeAlarm directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification, in alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotification services. Requests having the 'Initiating Device Identifier' parameter equal-to and different from SOURCE, are both tested.

Test Concept: Two times, a purposefully constructed event notification is sent, and it is observed that IUT is acknowledging directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification.

Configuration: For this test, the tester shall choose an object O1, and tester places O1 into an alarm state such that the transition requires an acknowledgment. Then purposefully the EventNotification packet which is sent is crafted to represent that a Notification Forwarder was involved, so though SOURCE is from the TD, the O1 resides in a different device TD1. Then the steps are repeated but with 'Initiating Device Identifier' representing that O1 is in TD and thus the same device. Each time it is observed that the AcknowledgeAlarm is sent to the device represented as the 'Initiating Device Identifier'.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
 'Initiating Device Identifier' = TD1, // representing that O1 is in different device from SOURCE
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),
 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request, DESTINATION=TD1
 'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgement' = (any valid value)
5. TRANSMIT BACnet-SimpleACK-PDU
6. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
 'Initiating Device Identifier' = TD, // representing that O1 is present in same device as SOURCE
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),
 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
7. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
8. MAKE (the IUT acknowledge O1)
9. RECEIVE AcknowledgeAlarm-Request,
 'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state
 'Time Stamp' = T1,

'Acknowledgement Source' = (any valid value),

'Time of Acknowledgement' = (any valid value)

10. TRANSMIT BACnet-SimpleACK-PDU

8.2 ConfirmedCOVNotification Service Initiation Tests

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

Reason for Change: Add more primitive value objects. Updated description of the 'List of Values' to improve readability. Updated 'Configuration Requirements'. Add clarification to test that the last COVNotification shall reflect the correct values.

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

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

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

Notes to Tester: *The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.*

Test Steps:

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

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

```

'Property Identifier' = COV_Increment,
'Property Value' = (a value "increment" that will be used below)
7. IF (Out_Of_Service is writable) THEN
    WRITE X, Out_Of_Service = TRUE

    BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same value used in step 1),
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = X,
            'Time Remaining' = (any value appropriate for the Lifetime selected),
            'List of Values' = (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~~ a *Numeric Object's Status_Flags Property*

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

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

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

notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write to Status_Flags or Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped

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

Test Steps:

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

1. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
 'Monitored Object Identifier' = X,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the initial Present_Value and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out_Of_Service = TRUE | WRITE X, Status_Flags = (a value that differs from initial Status_Flags) |
 MAKE (Status_Flags = any value that differs from initial Status_Flags)
- ~~6. IF (WriteProperty is used in step 5) THEN~~
 ~~RECEIVE BACnet-SimpleACK-PDU~~
76. 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)
87. TRANSMIT BACnet-SimpleACK-PDU
98. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Monitored Object Identifier' = X
- ~~109. RECEIVE BACnet-SimpleACK-PDU~~
- ~~110. 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~~ *Discrete Valued Object's Present_Value Property*

Reason for Change: Updated the 'Configuration Requirements'. Removed extraneous SimpleACKs that appear after WRITE statements. Modified descriptive text for 'List of Values' properties. Add clarification to test that the last COVNotification shall reflect the correct values.

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.*

Notes to Tester: *The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.*

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 Current Status_Flags)
 - TRANSMIT BACnet-SimpleACK-PDU
6. IF (Present_Value is now writable) THEN
 - WRITE X, Present_Value = (any value that differs from "initial Present_Value" ReportedPV)
 - ELSE
 - MAKE (Present_Value = any value that differs from "initial Present_Value" ReportedPV)

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

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

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. ~~Removed extraneous SimpleACKs after WRITE statements.~~ L shall be set to a value less than 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 ~~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
76. 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 initialthe current Present_Value, and new-Current Status_Flags)
87. TRANSMIT BACnet-SimpleACK-PDU
98. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' = (the same value used in step 1),
        'Monitored Object Identifier' = X
109. RECEIVE BACnet-SimpleACK-PDU
110 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~~Multi-state or other Discrete Valued 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. Add clarification to test that the last COVNotification shall reflect the correct values.

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.*

Notes to Tester: *The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.*

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 BACnet-SimpleACK-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
 Current Status_Flags)
 TRANSMIT BACnet-SimpleACK-PDU
6. IF (Present_Value is now writable) THEN
 WRITE X, Present_Value = (any value that differs from "initial-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-Current 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~~ Multi-state or other Discrete Valued 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~~Current Status_Flags)
- ~~87. TRANSMIT BACnet-SimpleACK-PDU~~
98. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Monitored Object Identifier' = X
- ~~109. RECEIVE BACnet-SimpleACK-PDU~~
- ~~110. 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 Setpoint, and initial 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, initialcurrent Setpoint, and
initialcurrent 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 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' =                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)
87. TRANSMIT BACnet-SimpleACK-PDU
98. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' =                (the same value used in step 1),
    'Monitored Object Identifier' =                O1
109. RECEIVE BACnet-SimpleACK-PDU
110. IF (Out_Of_Service was changed in step 5) THEN
WRITE O1, Out_Of_Service = FALSE
RECEIVE BACnet-SimpleACK-PDU

```

8.2.X9 ConfirmedCOVNotification Pulse Converter changing Present_Value

Reason for Change: New test.

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.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

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

Reason for Change: New Test

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. For some implementations writing to the Out_Of_Service property will accomplish this task. For implementations where it is not possible to write 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. COV_Period is configured high enough that it does not trigger many COV notifications during the execution of the test.

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

8.2.X11 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Access Door 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. 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 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. If no object has a Door_Alarm_State property, then step 9,10,11 shall be skipped. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, step 5,6,7 shall be skipped

Test Steps:

REPEAT X = (one supported object of type Access Door) 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, initial Status_Flags, and
 Door_Alarm_State if X has a Door_Alarm_State property)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = TRUE
 ELSE
 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' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (ReportedPV=current Present_Value, new Status_Flags, and
 Door_Alarm_State if X has a Door_Alarm_State property)
7. TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present_Value is writable) THEN
 WRITE X,Present_Value = (any value that differs from ReportedPV)

- ELSE
 MAKE (Present_Value = any value that differs from ReportedPV)
9. 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, new Status_Flags, and Door_Alarm_State if X has a Door_Alarm_State property)
10. TRANSMIT BACnet-SimpleACK-PDU
11. IF (Door_Alarm_State is now writable) THEN
 WRITE Door_Alarm_State = (any value that differs from its initial Door_Alarm_State)
 ELSE
 MAKE (Door_Alarm_State = any value that differs from its initial Door_Alarm_State)
12. 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, new Status_Flags, and Door_Alarm_State)
13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (the same value used in the SubscribeCOV-Request),
 'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
 WRITE 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~~, a Numeric Object's Present_Value Property

Reason for Change: Addendum 135-2008w-1, and 135-2-1-i-1 Add more primitive value objects and the Lighting Output Object. Add clarification to test that the last COVNotification shall reflect the correct values.

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

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

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

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

Reason for Change: 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, *Lighting Output*, ~~and~~ Analog Value, *Large Analog Value*, *Integer Value*, and *Positive Integer Value* objects.

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

8.3.3 Change of Value Notification from a ~~Binary Input, Binary Output, and Binary Value~~ Discrete Valued Object's Present_Value Property

Reason for Change: Renamed test. Add clarification to test that the last COVNotification shall reflect the correct values.

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

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

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

8.3.4 Change of Value Notification from a ~~Binary Input, Binary Output, and Binary Value~~ Discrete Valued Object's Status_Flags Property

Reason for Change: Renamed test.

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

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

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~~ Multi-state or other Discrete Valued Object Present_Value Property

Reason for Change: Addendum 135-2008w-1 Add more primitive value objects. Add clarification to test that the last COVNotification shall reflect the correct values.

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.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

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~~ Multi-state or other Discrete Valued 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.

8.3.7 Change of Value Notification from Loop Object Present_Value Property

Reason for Change: Add clarification to test that the last COVNotification shall reflect the correct values.

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

Test Steps: The steps for this test case are identical to the test steps in 8.2.7 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.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

8.3.10 Device Restart Notifications

Reason for Change: CR-0437 pointed out that test 135.1-2013 8.3.10 does not work for devices that don't have a Local_Time property and use a sequence number in Restart Notifications.

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests to each entry in its Restart_Notification_Recipients property when it resets.

Test Concept: The IUT is configured to send restart notifications and is then reset. The TD checks for the restart notifications.

Device restart notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, subscription is made through the Restart_Notification_Recipients property instead of SubscribeCOV. Second, the 'Subscriber Process Identifier' parameter always has a value of zero.

Configuration Requirements: For each Recipient of the Restart_Notification_Recipients property in the IUT which is of the device form, there shall be a device on the network that will answer Who-Is requests so that the IUT can determine addressing information before sending the restart notification.

Test Steps:

```

1. IF (Restart_Notification_Recipients is writable) THEN
    WRITE(Restart_Notification_Recipients = any non-empty list of Recipients)
ELSE
    MAKE (Restart_Notification_Recipients contain any non-empty list of Recipients)
2. READ T1 = Local Time
32. MAKE (the IUT reset)
43. REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
    BEFORE Notification Fail Time
    RECEIVE UnconfirmedCOVNotification-Request,
        DESTINATION = X,
        'Subscriber Process Identifier' = 0,
        'Initiating Device Identifier' = IUT,
        'Monitored Object Identifier' = (the IUT Device Identifier),
        'Time Remaining' = 0,
        'List of Values' = (System_Status=OPERATIONAL,
                           Time_Of_Device_Restart = (T2),
                           Last_Restart_Reason=(any valid restart reason, R))
    }
54. VERIFY Time_Of_Device_Restart = T2
6. CHECK (T1 = T2)
75. VERIFY Last_Restart_Reason = R
6. IF (T2 is not a sequence number) THEN
    VERIFY Local_Time ~ T2
ELSE
    MAKE (the IUT reset)
    REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
        BEFORE Notification Fail Time
        RECEIVE UnconfirmedCOVNotification-Request,
            DESTINATION = X,
            'Subscriber Process Identifier' = 0,
            'Initiating Device Identifier' = IUT,
            'Monitored Object Identifier' = (the IUT Device Identifier),
            'Time Remaining' = 0,
            'List of Values' = (System_Status=OPERATIONAL,
                               Time_Of_Device_Restart = (T3),
                               Last_Restart_Reason=(any valid restart reason, R))

        CHECK (T3 > T2)

```

Note to tester: Not all IUTs can accurately differentiate between the types of restart reasons and thus no requirements are placed on the value returned in the restart notification(s) ~~step 4~~. The test shall pass regardless of the order in which the restart notifications are sent to the recipients. ~~IUT generates the UnconfirmedCOVNotification-Requests in step 4. The value of T2 shall be the same for each notification sent out in step 4. If the Restart_Notification_Recipients list has multiple recipients, then the Time_Of_Device_Restart value is expected to be the same in all notifications resulting from the same restart.~~

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)
 - IF (X is the first entry in the COVU_Recipients) THEN
 - READ T1 = Local_Time
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)
 - IF (X is the first entry in the COVU_Recipients) THEN
 - READ T2 = Local_Time
4. 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.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

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.

8.3.X14 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value, Status_Flag and Door_Alarm_State property of Access Door objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X11 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no 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.

8.4.4 COMMAND_FAILURE Tests

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

BACnet Reference Clauses: 12.7, 12.12, 12.19, 13.2, 13.3.4, and 13.8.

Purpose: To verify the correct operation of the COMMAND_FAILURE algorithm.

Test Concept: The Feedback_Value (Feedback_Property_Reference) shall be decoupled from the input signal that is normally used to verify the output. Initially Present_Value (referenced property) and Feedback_Value (Feedback_Property_Reference) are in agreement. Present_Value (the referenced property) is changed and an event notification should be transmitted indicating a transition to an OFFNORMAL state. The Feedback_Value (Feedback_Property_Reference) is changed to again agree with the Present_Value (referenced property). A second event notification is transmitted indicating a return to a NORMAL state.

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. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. The Feedback_Value property shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

In the test description below Present_Value is used as the referenced property and Feedback_Value is used to verify the output. If an Event Enrollment object is being tested these properties shall be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. IF (the object being tested is not an Event Enrollment object) THEN
 VERIFY Status_Flags = (~~FALSE, FALSE, FALSE, FALSE~~)(FALSE, FALSE, ?, ?)
3. IF (Present_Value is writable) THEN
 WRITE Present_Value = (a different value)
 ELSE
 MAKE (Present_Value take on a different value)
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 object being tested),
 'Time Stamp' = (the current local time),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = COMMAND_FAILURE,
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = Present_Value, Status_Flags, Feedback_Value
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. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
 VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
10. IF (Feedback_Value is writable) THEN
 WRITE Feedback_Value = (a value consistent with Present_Value)
 ELSE
 MAKE (Feedback_Value take on a value consistent with Present_Value)
11. WAIT (Time_Delay)
12. 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' = (the current local time),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = COMMAND_FAILURE,
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = NORMAL,
 'Event Values' = Present_Value, Status_Flags, Feedback_Value

13. TRANSMIT BACnet-SimpleACK-PDU
14. IF (the object being tested is not an Event Enrollment object) THEN
 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15. VERIFY Event_State = NORMAL
16. IF (Protocol_Revision is present and Protocol_Revision \geq 1) THEN
 VERIFY Event_Time_Stamp = (the timestamp in step 5, *, the timestamp in step 12)

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 "*" in steps 9 and 16 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 5.

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 = (Tnormal, *, 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 = (Tnormal, 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.X6 Extended Algorithm Tests (ConfirmedEventNotifications)

Reason for Change: Addition of Alert Enrollment object in Protocol_Revision 13.

Purpose: To verify the correct generation EXTENDED event notifications.

Test Concept: The object begins the test in a NORMAL state. The IUT is manipulated in order to cause a transition of the object under test. The event notification content is verified for correctness.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the transition type that will be generated. The Issue_Confirmed_Notifications property shall have a value of TRUE.

Test Steps:

1. IF (the object generates TO-OFFNORMAL transitions) THEN
 2. READ ES1 = Event_State
 3. MAKE (an OFFNORMAL condition exist)
 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 event generating object),

'Time Stamp' = (TS1: the current local time),

'Notification Class' = (the configured notification class),

'Priority' = (the value configured for TO_OFFNORMAL),

'Event Type' = EXTENDED,

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = ES1,

'To State' = (ES2: any offnormal valid event state),

'Event Values' = ((any valid vendor id),
 (any valid event-type),
 (a list of 0 or more valid parameters)
)
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 = ES2
9. VERIFY Event_Time_Stamps = (TS1, *, *)
10. IF (the object generates TO_NORMAL transitions) THEN
 11. READ ES1 = Event_State
 12. MAKE (a NORMAL condition exist)
 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),

```

'Time Stamp' =          (TS2: the current local time),
'Notification Class' =  (the configured notification class),
'Priority' =            (the value configured to correspond to a
                        TO-NORMAL transition),
'Event Type' =          EXTENDED,
'Notify Type' =         EVENT | ALARM,
'AckRequired' =        TRUE | FALSE,
'From State' =          ES1,
'To State' =            NORMAL,
'Event Values' =        ( (any valid vendor id),
                        (any valid event-type),
                        (a list of 0 or more valid parameters)
                        )

```

14. TRANSMIT BACnet-SimpleACK-PDU
15. IF (the object being tested is NOT an Event Enrollment object) THEN
 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
16. VERIFY Event_State = NORMAL
17. VERIFY Event_Time_Stamp = (*, *, TS2)

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 any valid value.

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' = (Tnormal: 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 \geq 13)) THEN
 VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
- 7. VERIFY pCurrentState = HIGH_LIMIT
- 8. IF (Protocol_Revision is present AND Protocol_Revision \geq 1) THEN
 VERIFY Event_Time_Stamps = (Tnormal, *, *)
- 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 \geq 13)) THEN
 VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
- 14. VERIFY pCurrentState = NORMAL
- 15. IF (Protocol_Revision is present AND Protocol_Revision \geq 1) THEN
 VERIFY Event_Time_Stamps = (Tnormal, *, 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 \geq 13)) THEN
 VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
- 21. VERIFY pCurrentState = LOW_LIMIT
- 22. IF (Protocol_Revision is present AND Protocol_Revision \geq 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 \geq 13)) THEN
 VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
- 28. VERIFY pCurrentState = NORMAL
- 29. IF (Protocol_Revision is present AND Protocol_Revision \geq 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

8.4.X9 CHANGE_OF_RELIABILITY with the FAULT_OUT_OF_RANGE Algorithm

Purpose: To verify the correct operation of the FAULT_OUT_OF_RANGE event algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_OUT_OF_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. 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, to have no fault conditions present. 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.

Test Steps:

1. VERIFY pCurrentReliability = NO_FAULT_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN
 - WRITE pMonitoredValue = (a value less than pMinimumNormalValue | a value greater than pMaximumNormalValue)
- ELSE
 - MAKE (pMonitoredValue = a value less than pMinimumNormalValue | a value greater than pMaximumNormalValue)
4. BEFORE **Notification Fail Time**,
 - RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the class corresponding to the object O1 being tested),
 - 'Priority' = (the value configured to correspond to a TO_FAULT 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' = (pCurrentReliability, pStatusFlags, pMonitoredValue, pMinimumNormalValue | pMaximumNormalValue)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentReliability = UNDER_RANGE | OVER_RANGE
7. VERIFY pCurrentState = FAULT
8. VERIFY pStatusFlags = (TRUE, TRUE, ?, ?)
9. IF (pMonitoredValue is writable) THEN
 - WRITE pMonitoredValue = (a value greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue)
- ELSE
 - MAKE (pMonitoredValue = a value, greater than or equal to pMinimumNormalValue, and pMonitoredValue is less than or equal to pMaximumNormalValue)
10. BEFORE **Notification Fail Time**,
 - RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the class corresponding to the object O1 being tested),
 - 'Priority' = (the value configured to correspond to a TO_FAULT 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' = (pCurrentReliability, pStatusFlags, pMonitoredValue,
 pMinimumNormalValue | pMaximumNormalValue)

11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY pCurrentReliability = NO_FAULT_DETECTED
13. VERIFY pCurrentState = NORMAL
14. VERIFY pStatusFlags = (FALSE, FALSE,?, ?)

8.4.X10 CHANGE_OF_DISCRETE_VALUE Test (ConfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE_OF_DISCRETE_VALUE event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_DISCRETE_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelay (pTimeDelay is the value for Time_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a ConfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event_Enrollment, EE1 is configured with an Object_Property_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date, Time, BACnetObjectIdentifier, or BACnetDateTime. Event_Enable is configured with a value of (T,T,T) and Event_Algorithm_Inhibit = FALSE. The Event_Parameters are configured with an Event Algorithm of CHANGE_OF_DISCRETE_VALUE and a value for Time_Delay that is within the allowable range for the IUT. The configured notification class is configured to send confirmed notifications to the TD. EE1 shall have an Event_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY Event_State = Normal
2. MAKE (the referenced property have a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE Notification Fail Time
 - RECEIVE ConfirmedEventNotification
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = EE1,
 - 'Time Stamp' = (the current local datetime or time or sequence number),
 - 'Notification Class' = (the notification class configured for EE1),
 - 'Priority' = (the value configured to correspond to TO-NORMAL),
 - 'Event Type' = CHANGE_OF_DISCRETE_VALUE,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = ALARM | EVENT,
 - 'Ack Required' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = NORMAL,
 - 'Event Values' = (x, Status_Flags of O1)
5. TRANSMIT BACnet SimpleAck-PDU
6. 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)

Reason for Change: Addition of Alert Enrollment object in Protocol_Revision 13.

Purpose: To verify the correct generation EXTENDED event notifications.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the transition type that will be generated. The Issue_Confirmed_Notifications property shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.X6 except that the event notification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

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

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),

```


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

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 Steps:

- 224

'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

8.5.X9.15 CHANGE_OF_RELIABILITY with the FAULT_OUT_OF_RANGE Algorithm

Purpose: To verify the correct operation of the FAULT_OUT_OF_RANGE event algorithm.

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT_OUT_OF_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. 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, to have no fault conditions present. 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.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.X9.15 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.X9.15 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.X10 CHANGE_OF_DISCRETE_VALUE Test (UnconfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE_OF_DISCRETE_VALUE event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_DISCRETE_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelayNormal (pTimeDelay is the value for Time_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a UnconfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event_Enrollment, EE1 is configured with an Object_Property_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date,

Time, BACnetObjectIdentifier, or BACnetDateTime. Event_Enable is configured with a value of (T,T,T) and Event_Algorithm_Inhibit = FALSE. The Event_Parameters are configured with an Event Algorithm of CHANGE_OF_DISCRETE_VALUE and a value for Time_Delay that is within the allowable range for the IUT. The configured notification class is configured to send unconfirmed notifications to the TD. EE1 shall have an Event_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY Event_State = Normal
2. MAKE (the referenced property have a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE Notification Fail Time
 - RECEIVE UnconfirmedEventNotification
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = EE1,
 - 'Time Stamp' = (the current local datetime or time or sequence number),
 - 'Notification Class' = (the notification class configured for EE1),
 - 'Priority' = (the value configured to correspond to TO-NORMAL),
 - 'Event Type' = CHANGE_OF_DISCRETE_VALUE,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = ALARM | EVENT,
 - 'Ack Required' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = NORMAL,
 - 'Event Values' = (x, Status_Flags of O1)
5. 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 value)~~ 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 and then cancelled or allowed to expire. A ConfirmedCOVNotification is then sent to the IUT to verify it returns the appropriate error or a Simple-Ack.

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. IF (the IUT can cancel the subscription) THEN
 - RECEIVE SubscribeCOVProperty – Request,
 - 'Subscriber Process Identifier' = (the process identifier used in step 1),
 - 'Monitored Object Identifier' = X
 - 'Monitored Property Identifier' = Y
 - 'COV Increment' = (Any REAL value –optional)
 - ELSE
 - 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 \geq 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 \geq 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 \geq 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 ReadPropertyMultiple Shall Be Used After ReadPropertyMultiple Fails

The tests defined in this clause are used to verify that an IUT which initiates 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 properties 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 - 8.21.8X3~~ 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, P₁, from an object, O₁. The TD shall be configured to not support execution of WritePropertyMultiple.

The WriteProperty request initiated by IUT shall contain array of elements in P₁, which shall fit in the APDU and segment limitations of the IUT.

Test Steps:

1. MAKE (the IUT accept a new value for P₁ including all elements of the array from the user)
2. RECEIVE WriteProperty-Request,
 'Object Identifier' = O₁,
 'Property Identifier' = P₁,
 'Property Value' = (the value provided to the IUT for P₁)
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

8.32.1 Object Identifier Selection with no Device Instance Range

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send and receive a unicast response.

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with no 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,
 'Object Identifier' = *ObjectI* (~~any object identifier~~)
2. TRANSMIT
 DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST
 SOURCE = TD,
 I-Have-Request,
 'Device Identifier' = (the TD's Device object)
 'Object Identifier' = *ObjectI*
3. CHECK (for any vendor-defined observable actions)

Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.

8.32.2 Object Name Selection with no Device Instance Range

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send and receive a unicast response.

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with no 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,
 'Object Name' = *VI* (~~any CharacterString~~)
2. TRANSMIT
 DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST
 SOURCE = TD,
 I-Have-Request,
 'Device Identifier' = (the TD's Device object)
 'Object Name' = *VI*
3. CHECK (for any vendor-defined observable actions)

Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.

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. The allowance for Unicast I-Have is added.

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: $0 \leq X \leq$ 'Device Instance Range High Limit'),
 'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' $\leq Y \leq$ 4,194,303),
 'Object Identifier' = *Object1* (~~any object identifier~~)

2. TRANSMIT

DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST
SOURCE = TD,
 I-Have-Request,
 'Device Identifier' = (the TD's Device object)
 'Object Identifier' = *Object1*

3. CHECK (for any vendor-defined observable actions)

Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.

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. The allowance for Unicast I-Have is added.

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: $0 \leq X \leq$ 'Device Instance Range High Limit'),
 'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' $\leq Y \leq$ 4,194,303),
 'Object Name' = *VI* (~~any CharacterString~~)

2. TRANSMIT

DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST
SOURCE = TD,
 I-Have-Request,
 'Device Identifier' = (the TD's Device object)
 'Object Name' = *VI*

3. CHECK (for any vendor-defined observable actions)

Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.

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: $40 \leq X \leq$ 'Device Instance Range High Limit'),
 'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' $\leq Y \leq$ 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 \geq 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 \geq 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 = ~~FALSE~~(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 = ~~FALSE~~(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

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 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 '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 \geq 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 \geq 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 an 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 reported shall have an Error Class of SERVICES and Error Code* ~~in step 7 shall be of INVALID_EVENT_STATE~~. For devices claiming a Protocol Revision less than 5, Error Class of SERVICES and 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 \geq 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 an *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 reported shall have an Error Class of SERVICES and Error Code in step 7 shall be of INVALID_EVENT_STATE. For devices claiming a Protocol Revision less than 5, an Error Class of SERVICES and 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 Acknowledgment Source 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.

9.2.1.X5 ConfirmedCOVNotification from Access Door Object

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier value > 0),
 - 'Monitored Object Identifier' = (any Access Door object, X),
 - 'Issue Confirmed Notifications' = TRUE,
 - 'Lifetime' = (a value greater than one minute)
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' = (the time remaining in the subscription),
 - 'List of Values' = (the initial Present_Value, initial Status_Flags, and Door_Alarm_State if X has a Door_Alarm_State property)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying

information on a workstation screen are carried out)

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, *PI*),
 '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, PI),~~
 '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 (the IUT can cancel the subscription) THEN
 RECEIVE SubscribeCOV – Request,
 'Subscriber Process Identifier' = (PI),
 'Monitored Object Identifier' = X
 ELSE
 MAKE (the IUT stop resubscribing, if it resubscribes automatically)
53. WAIT ~~(a value two times at least Lifetime, but sufficient to ensure the subscription has expired)~~
64. TRANSMIT ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = ~~(the process identifier used in step 2 PI),~~
 '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)
75. 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 21),
 '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 21),
 '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

9.3.1.X6 UnconfirmedCOVNotification from Access Door Object

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier value > 0),
 - 'Monitored Object Identifier' = (any Access Door object, X),
 - 'Issue Confirmed Notifications' = FALSE,
 - 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,
 - 'Subscriber Process Identifier' = (the process identifier used in step 1),
 - 'Initiating Device Identifier' = TD,
 - 'Monitored Object Identifier' = X,
 - 'Time Remaining' = (the time remaining in the subscription),
 - 'List of Values' = (the initial Present_Value, initial Status_Flags, and Door_Alarm_State if X has a Door_Alarm_State property)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

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)
79. WAIT (the lifetime of the subscription)
810. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
911. 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, V₁ through V₅, and then check to ensure that 5 notifications are sent when the monitored object changes.

Test Steps

1. REPEAT (X=V₁ to V₅) DO {
 - TRANSMIT SubscribeCOV-Request,
 - 'Subscriber Process Identifier' = X,
 - 'Monitored Object Identifier' = (any object supporting COV notifications),
 - 'Issue Confirmed Notifications' = TRUE | FALSE,
 - 'Lifetime' = (any valid value that will allow the subscription to outlast the test)
 - RECEIVE BACnet-SimpleACK-PDU
 - IF (if confirmed notifications were requested) THEN
 - BEFORE **Notification Fail Time**
 - 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
 - BEFORE **Notification Fail Time**
 - 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=V₁ to V₅) 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 \geq 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 \geq 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. CR-0369 clarified that the testing shall only supply a 'Lifetime' parameter in the SubscribeCOV-Request less than the maximum unsigned value supported.

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 less than the maximum unsigned value supported by the IUT, but large enough to produce a Result(-) result 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)

9.11 SubscribeCOVProperty Service Execution Tests

9.11.1 Positive SubscribeCOVProperty Service Execution Tests

9.11.1.1 Confirmed COV Notifications

Reason for Change: Remove the allowance for devices which do not support both confirmed and unconfirmed notifications.

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription for confirmed COV notifications. ~~An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.11.1.2.~~

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier),
 - 'Monitored Object Identifier' = (any object supporting COV notifications),
 - 'Issue Confirmed Notifications' = TRUE,
 - 'Lifetime' = (any value > 0 ~~if automatic cancellation is supported, otherwise 0~~),
 - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 - ~~IF (the IUT supports confirmed notifications) THEN~~
 - RECEIVE BACnetConfirmedCOVNotification-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' = (any value > 0 ~~if automatic cancellation is supported, otherwise 0~~),
 - 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
 - ELSE
 - RECEIVE BACnet-Error-PDU,
 - Error Class = SERVICES,
 - Error Code = SERVICE_REQUEST_DENIED | OTHER
4. TRANSMIT BACnet-SimpleACK-PDU

9.11.1.2 Unconfirmed COV Notifications

Reason for Change: Remove the allowance for devices which do not support both confirmed and unconfirmed notifications.

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription for Unconfirmed COV notifications. ~~An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.11.1.1.~~

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier),
 - 'Monitored Object Identifier' = (any object supporting COV notifications),
 - 'Issue Confirmed Notifications' = FALSE,
 - 'Lifetime' = (any value > 0 ~~if automatic cancellation is supported, otherwise 0~~),
 - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 - ~~IF (the IUT supports unconfirmed notifications) THEN~~
 - RECEIVE BACnetUnconfirmedCOVNotification-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' = (any value > 0 ~~if automatic cancellation is supported, otherwise 0~~),
 'List of Values' = (values appropriate to the property subscribed to, and any other properties
 the IUT provides with it, such as Status-Flags)

~~ELSE~~

~~RECEIVE BACnet-Error-PDU,~~

~~Error Class = SERVICES,~~

~~Error Code = SERVICE_REQUEST_DENIED | OTHER~~

9.11.1.5 Canceling Expired or Non-Existing Subscriptions

Reason for change: Added missing verification that the IUT did not send a COV notification, and removed superfluous note to tester.

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to cancel a subscription that no longer exists.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any unused process identifier or an identifier from a previously terminated subscription),
 'Monitored Object Identifier' = (any unused object or an object from a previously terminated subscription),
 'Monitored Property Identifier' = (any unused property or a property from a previously terminated subscription)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Notification Fail Time**
4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
5. CHECK(*the IUT did not issue a COV notification*)

~~Notes to Tester: The IUT shall not transmit a COV notification message. An error message is not an acceptable response.~~

9.11.1.7 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 SubscribeCOVProperty 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.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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),
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
- ~~3. BEFORE **Notification Fail Time**~~
3. IF (the subscription was for confirmed notifications) THEN
 BEFORE **Notification Fail Time**
 RECEIVE BACnetConfirmedCOVNotification-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' = ~~(the requested subscription lifetime~~ *A value approximately equal to, but not greater than, the requested subscription lifetime*),
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

BEFORE Notification Fail Time

RECEIVE BACnetUnconfirmedCOVNotification-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' ~ = (the requested lifetime)
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)

4. MAKE (a change to the monitored object that ~~should~~ causes a COV notification)

5. WAIT *a period longer than the resolution of the IUT's COV subscription lifetime timer*

~~5. BEFORE Notification Fail Time~~

6. IF (the subscription was for confirmed notifications) THEN

BEFORE Notification Fail Time

RECEIVE BACnetConfirmedCOVNotification-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 the requested subscription lifetime),
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

BEFORE Notification Fail Time

RECEIVE BACnetUnconfirmedCOVNotification-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 the requested subscription lifetime),
 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

7. WAIT *a period longer than the resolution of the IUT's COV subscription lifetime timer*

8. MAKE (a change to the monitored object that causes a COV notification)

9. IF (the subscription was for confirmed notifications) THEN

BEFORE Notification Fail Time

RECEIVE BACnetUnconfirmedCOVNotification-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)

ELSE

BEFORE Notification Fail Time

RECEIVE BACnetUnconfirmedCOVNotification-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
including the changed value that triggered the notification)

610. WAIT (the lifetime of the subscription)

711. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)

12. CHECK (verify that the IUT did not transmit a COV notification message)

~~Notes to Tester: The IUT shall not transmit a COV notification message addressed to the TD after step 6.~~

9.11.1.9 Client-Supplied COV Increment

Reason for Change: Modify the test to work with all numeric datatypes.

Purpose: To verify that the IUT correctly generates COV notifications when the client supplies the COV increment in the SubscribeCOVProperty request. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Concept: A subscription for COV notification is made for a property of *numeric* datatype ~~REAL~~. The subscription request specifies a COV increment. The monitored property is changed by an amount less than the increment, and the TD waits to ensure that the IUT does not generate a notification. The monitored property is changed by an amount slightly more than is required to cause a COV notification, and the TD waits for the notification.

Test Configuration: If the property being subscribed to has a related COV_Increment property in the object, then the value of the COV_Increment property should be significantly different than the COV increment provided in the subscription service.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier),
 - 'Monitored Object Identifier' = (any object supporting COV notifications),
 - 'Issue Confirmed Notifications' = TRUE | FALSE,
 - 'Lifetime' = (any value that will ensure no re-subscription is required to complete the test),
 - 'Monitored Property Identifier' = (any valid property supporting COV notifications),
 - 'COV Increment' = (any valid increment value)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 - IF (the subscription was for confirmed notifications) THEN
 - RECEIVE BACnetConfirmedCOVNotification-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' ~= (the requested lifetime),
 - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
 - TRANSMIT BACnet-SimpleACK-PDU
 - ELSE
 - RECEIVE BACnetUnconfirmedCOVNotification-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' ~= (the requested lifetime),
 - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
4. MAKE (the monitored property change by less than the COV increment)

5. CHECK (verify that the IUT did not transmit a notification message for the monitored property)
6. MAKE (the monitored property change by slightly more than COV Increment less the amount changed in step 54)
7. BEFORE **Notification Fail Time**
 - IF (the subscription was for confirmed notifications) THEN
 - RECEIVE BACnetConfirmedCOVNotification-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' = ?,
 - 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)
 - TRANSMIT BACnet-SimpleACK-PDU
 - ELSE
 - RECEIVE BACnetUnconfirmedCOVNotification-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' = ?,
 - 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

9.11.1.X10 Accepts SubscribeCOVProperty-Requests with 8 Hour Lifetimes

Reason for Change: No tests exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT correctly accepts lifetimes of at least 8 hours.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 - 'Subscriber Process Identifier' = (any valid process identifier),
 - 'Monitored Object Identifier' = (any object supporting COV notifications),
 - 'Issue Confirmed Notifications' = TRUE | FALSE,
 - 'Lifetime' = 28800
 - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
 - IF (the subscription was for confirmed notifications) THEN
 - RECEIVE BACnetConfirmedCOVNotification-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' ~ = (the requested lifetime),
 - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
 - TRANSMIT BACnet-SimpleACK-PDU
 - ELSE
 - RECEIVE BACnetUnconfirmedCOVNotification-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' ~ = (the requested lifetime),
 - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
4. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (the same identifier used in Step 1),
 'Monitored Object Identifier' = (the same identifier used in the subscription),
 'Monitored Property Identifier' = (the same object used in the subscription)

5. RECEIVE BACnet-SimpleACK-PDU

9.11.1.X11 Confirmed Change of Value Notification from Property Value

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Property Value.

Test Concept: A property 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 Value of the monitored Property is changed and a notification shall be received. The subscribed property may be changed using the WriteProperty service or by another means. 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.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = X
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
 'Monitored Property Identifier' = Y (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
 RECEIVE BACnetConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same identifier used in the subscription),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (a change to the monitored object PROPERTY that causes a COV notification)
6. BEFORE Notification Fail Time
 RECEIVE BACnetConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same identifier used in the subscription),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
7. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (the same identifier used in Step 1),
 'Monitored Object Identifier' = X
 'Monitored Property Identifier' = Y
8. RECEIVE BACnet-SimpleACK-PDU

9.11.1.X12 Unconfirmed Change of Value Notification from Property Value

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

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.X11 except that the SubscribeCOVProperty 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

9.11.1.X21 Confirmed Change of Value Notification from Status_Flags Property

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

Test Concept: A property 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.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = X
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
 'Monitored Property Identifier' = Y (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
 RECEIVE BACnetConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same identifier used in the subscription),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (values appropriate to the property subscribed to and initial Status_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (Status_Flags = any value that differs from "initial Status_Flags")
6. BEFORE Notification Fail Time
 RECEIVE BACnetConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same identifier used in the subscription),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (initial values appropriate to the property subscribed to and new Status_Flags)
7. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (the same identifier used in Step 1),
 'Monitored Object Identifier' = X
 'Monitored Property Identifier' = Y
8. RECEIVE BACnet-SimpleACK-PDU

9.11.1.X22 Unconfirmed Change of Value Notification from Status_Flags Property

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

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.X21 except that the SubscribeCOVProperty 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

9.11.2 Negative SubscribeCOVProperty Service Execution Tests**9.11.2.1 The Monitored Object Does Not Support COV Notification**

Reason for Change: Update the accepted error responses as per changes made in Protocol_Revision 15.

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object does not support COV notifications.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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,
 - 'Monitored Property Identifier' = (any property in the object)
- ~~2. RECEIVE BACnet-Error-PDU,~~
 - ~~Error Class = SERVICES,~~
 - ~~Error Code = SERVICE_REQUEST_DENIED | OTHER~~
2. IF (Protocol_Revision < 15) THEN
 - RECEIVE
 - (BACnet-Error-PDU,
 - Error Class = SERVICES,
 - Error Code = SERVICE_REQUEST_DENIED | OTHER) |
 - (BACnet-Error-PDU,
 - Error Class = OBJECT,
 - Error Code = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
 - (BACnet-Error-PDU,
 - Error Class = PROPERTY,
 - Error Code = NOT_COV_PROPERTY)
 - ELSE
 - RECEIVE
 - (BACnet-Error-PDU,
 - Error Class = OBJECT,
 - Error Code = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
 - (BACnet-Error-PDU,
 - Error Class = PROPERTY,
 - Error Code = NOT_COV_PROPERTY)

9.11.2.2 The Monitored Property Does Not Support COV Notification

Reason for Change: Update the accepted error responses as per changes made in Protocol_Revision 15.

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object supports COV notifications but not on the requested property.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = (any object that supports COV notifications),
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = 60,
 'Monitored Property Identifier' = (any property of the chosen object that does not support COV notifications)
- ~~2. RECEIVE BACnet-Error-PDU,
 Error Class = SERVICES,
 Error Code = SERVICE_REQUEST_DENIED | OTHER~~
2. IF (Protocol_Revision < 15) THEN
 RECEIVE
 (BACnet-Error-PDU,
 Error Class = SERVICES,
 Error Code = SERVICE_REQUEST_DENIED | OTHER) |
 (BACnet-Error-PDU,
 Error Class = PROPERTY,
 Error Code = NOT_COV_PROPERTY)
 ELSE
 RECEIVE BACnet-Error-PDU,
 Error Class = PROPERTY,
 Error Code = NOT_COV_PROPERTY

9.11.2.X11 Monitored Object Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-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
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-Error-PDU,
 Error Class = OBJECT,
 Error Code = UNKNOWN_OBJECT

9.11.2.X12 Monitored Property Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored property does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = (object supporting COV notifications),
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = 60

'Monitored Property Identifier' = (any valid property supporting COV notifications which does not exist for specified object)

2. RECEIVE BACnet-Error-PDU,
 Error Class = PROPERTY,
 Error Code = UNKNOWN_PROPERTY

9.11.2.X13 There Is No Space For Subscription

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty 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 SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = PID,
 'Monitored Object Identifier' = (object supporting COV notifications),
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = 6000
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
 2. RECEIVE BACnet-SimpleACK-PDU
 | (BACnet-Error-PDU,
 Error Class = RESOURCES,
 Error Code = NO_SPACE_TO_ADD_LIST_ELEMENT)
- }

9.11.2.X14 The Lifetime Parameter is Out of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = (object supporting COV notifications),
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = (a value larger than that supported by the IUT),
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. IF (Protocol_Revision is present and Protocol_Revision => 15) 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)

9.12 Atomic ReadFile Service Execution Tests

9.12.1 Positive AtomicReadFile Service Execution Tests

9.12.1.2.1 Reading an Entire *Stream-Based* File

Reason for Change: Change to allow testing of files larger than that which can be returned in a single request.

Purpose: To verify that the IUT correctly responds to a request to read an entire file.

Test Concept: The test consists of reading the contents of the file using a sequence of AtomicReadFile requests and verifying that the appropriate known file data is returned.

Configuration Requirements: The AtomicReadFile service execution tests require that the TD has knowledge of the exact contents of a known file F1. The test procedures assume that the IUT is already configured with the known file data provided by the manufacturer. In the test procedures "X" will designate the File object identifier and Z the 'File Start Position' initialized at "0". When performing the AtomicReadFile services, a Maximum Requested Octet Count (MROC) 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.

Test Steps:

```

1. TRANSMIT AtomicReadFile Request,
   'File Identifier' = S,
   'File Start Position' = 0,
   'Requested Octet Count' = (the number of octets in the test file)
2. RECEIVE AtomicReadFile ACK,
   'End of File' = TRUE,
   'File Start Position' = 0,
   'File Data' = (the known contents of the test file)
1. VERIFY File_Access_Method = STREAM_ACCESS
2. WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
    TRANSMIT AtomicReadFile-Request,
    'Object Identifier' = X,
    'File Start Position' = Z (the next unread octet),
    'Requested Octet Count' = MROC
    RECEIVE AtomicReadFile-ACK,
    'End Of File' = TRUE | FALSE,
    'File Start Position' = Z
    'File Data' = (the known contents of the test file of length MROC if 'End Of File' is
                  FALSE or of length MROC or less if 'End Of File' is TRUE)
}
3. CHECK(that the returned file data is F1)

```

9.13 AtomicWriteFile Service Execution Tests

9.13.1 Positive AtomicWriteFile Service Execution Tests

9.13.1.2.1 Writing an Entire *Stream-Based* File

Reason for Change: Allow the entire file content to be written in all cases.

Purpose: To verify that the IUT correctly responds to a request to write an entire file.

Test Concept: The tests consist of modifying the contents of the files using the AtomicWriteFile service and verifying that the appropriate changes to the file data took place

Configuration Requirements: The test data shall contain at least as many octets as the initial data for the file. The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file objects shall be configured with initial data that differs from the test data. In the test procedures "X" will designate the File object identifier and Z the File Start Position' initialized at "1" at the beginning. When performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. 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. ~~TRANSMIT AtomicReadFile-Request,~~
~~'File Identifier' = S,~~
~~'File Start Position' = 0,~~
~~'Requested Octet Count' = (any number ≥ the number of octets in the test data)~~
2. ~~RECEIVE AtomicReadFile-ACK,~~
~~'End of File' = TRUE,~~
~~'File Start Position' = 0,~~
~~'File Record Data' = (the initial data)~~
3. ~~TRANSMIT AtomicWriteFile-Request,~~
~~'File Identifier' = S,~~
~~'File Start Position' = 0,~~
~~'File Data' = (the test data)~~
4. ~~RECEIVE AtomicWriteFile-ACK,~~
~~'File Start Position' = 0~~
5. ~~TRANSMIT AtomicReadFile-Request,~~
~~'File Identifier' = S,~~
~~'File Start Position' = 0,~~
~~'Requested Octet Count' = (any number > the number of octets in the test data)~~
6. ~~RECEIVE AtomicReadFile-ACK,~~
~~'End of File' = TRUE,~~
~~'File Start Position' = 0,~~
~~'File Data' = (the test data)~~
7. ~~VERIFY (R), Modification_Date = (the current date and time)~~
8. ~~VERIFY (R), ARCHIVE = FALSE~~
9. ~~VERIFY (R), Number_Of_Records = (the number of records in the test data)~~
 1. ~~VERIFY Read_Only = FALSE~~
 2. ~~WRITE Archive = TRUE~~
 3. ~~VERIFY File_Access_Method = STREAM ACCESS~~
 4. ~~IF (File_Size is not equal to the size of the test file) THEN~~
~~WRITE File_Size = 0~~
 5. ~~REPEAT Z = (0 through the file size, in increments of MWDL) DO {~~
~~TRANSMIT AtomicWriteFile-Request~~
~~'File Identifier' = X~~
~~'File Start Position' = Z~~
~~'File Data' = (file contents, the number of octets being the lesser of (file size - Z) and MWDL)~~
~~RECEIVE AtomicWriteFile-ACK~~
~~'File Start Position' = Z~~
~~}~~
 6. ~~VERIFY File_Size = (file size of the test data)~~
 7. ~~VERIFY Modification_Date = (the current date and time)~~
 8. ~~VERIFY ARCHIVE = FALSE~~

9.13.1.2.3 Appending Data to the End of a File

Reason for Change: Added configuration requirements, fixed purpose, and removed reliance on AtomicReadFile within the test steps.

Purpose: To verify that the IUT correctly responds to a request to write to the end of a file. ~~If the IUT does not support files that can not be modified except by replacing the entire file, and this restriction is clearly stated in the PICS, then this test may be ignored.~~

Configuration Requirements: *The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file objects shall be configured with initial data that differs from the test data. In the test procedures "X" will designate the File object identifier and. When performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. 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. TRANSMIT ReadProperty-Request,
 - 'Object Identifier' = S,
 - 'Property Identifier' = File_Size
2. RECEIVE ReadProperty-ACK,
 - 'Object Identifier' = S,
 - 'Property Identifier' = File_Size,
 - 'Property Value' = (the current size in octets, designated "InitialNumOctets" below)
- ~~3. TRANSMIT AtomicReadFile-Request,~~
 - ~~'File Identifier' = S,~~
 - ~~'File Start Position' = 0,~~
 - ~~'Requested Octet Count' = (any number > InitialNumOctets)~~
- ~~4. RECEIVE AtomicReadFile-ACK,~~
 - ~~'End of File' = TRUE,~~
 - ~~'File Start Position' = 0,~~
 - ~~'File Data' = (the initial data)~~
3. TRANSMIT AtomicWriteFile-Request,
 - 'File Identifier' = S,
 - 'File Start Position' = -1,
 - 'File Data' = (the test data, the number of octets being the lesser of (file size - Z) and MWDL)
4. RECEIVE AtomicWriteFile-ACK,
 - 'File Start Position' = InitialNumOctets,
- ~~7. TRANSMIT AtomicReadFile-Request,~~
 - ~~'File Identifier' = S,~~
 - ~~'File Start Position' = InitialNumOctets,~~
 - ~~'Requested Octet Count' = (the number of octets in the test data)~~
- ~~8. RECEIVE AtomicReadFile-ACK,~~
 - ~~'End of File' = TRUE,~~
 - ~~'File Start Position' = InitialNumOctets,~~
 - ~~'File Data' = (the test data)~~
5. VERIFY (R), Modification_Date = (the current date and time)
6. VERIFY (R), ARCHIVE = FALSE
7. VERIFY (R), File_Size = (the number of octets in the test data + InitialNumOctets)

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_SUPPORTED 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_SUPPORTED 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_TypeSpecifier' = (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_TypeSpecifier' = (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  $\geq$  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  $\geq$  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) THEN  
    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) THEN  
    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.

#### 9.18.1.X4 Reading Array Properties at different Array Indexes

Reason for Change: No test exists for this functionality.

Purpose: This test verifies the IUT can execute ReadProperty service requests on a single element in an array property.

Test Concept: This test will execute a ReadProperty service request to read a single element from the selected property by specifying the array-index in the request.. Another request is made to read an element of an array where the array index is out of range.

Configuration Requirement: O1 is any object in the IUT database having array property P1 having size X.

Test Steps:

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN  
    READ V = P1, ARRAY INDEX = 1  
    CHECK (V is any valid value of the correct data type for property P1)  
    READ V = P1, Array Index =X  
    CHECK (V is any valid value of the correct data type for property P1)
3. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'Property Array Index' = (X+1)
4. RECEIVE BACnet-Error-PDU,  
    Error Class = PROPERTY,  
    Error Code = INVALID\_ARRAY\_INDEX

#### 9.18.1.X8 ReadProperty Service when Non-BACnet Device Offline

Purpose: To verify that the ReadProperty service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadProperty Request,

- 'Object Identifier' = Object1,
- 'Property Identifier' = P1
- 5. RECEIVE ReadProperty-ACK,
- 'Object Identifier' = Object1,
- 'Property Identifier' = P1,
- 'Property Value' = (V, any valid value)

#### 9.20.1.X9 ReadPropertyMultiple Service when Non-BACnet Device Offline

Purpose: To verify that the ReadPropertyMultiple service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadPropertyMultiple-Request,
- 'Object Identifier' = Object1,
- 'Property Identifier' = P1
5. RECEIVE ReadPropertyMultiple-ACK,
- 'Object Identifier' = Object1,
- 'Property Identifier' = P1,
- 'Property Value' = (any valid value)

#### 9.21.1.X10 ReadRange Service when Non-BACnet Device Offline

Purpose: To verify that the ReadRange Service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadRange-Request,
- 'Object Identifier' = Object1,
- 'Property Identifier' = P1
5. RECEIVE ReadRange-ACK,
- 'Object Identifier' = Object1,
- 'Property Identifier' = P1,
- 'Property Value' = (any valid value)

## 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 P1the value of P1 specified in the EPICS),  
 'Object Identifier' = Object2,  
 'Property Identifier' = P2,  
 'Property Value' = (any valid value for P2the 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 P1the value of P1 specified in the EPICS),  
 'Property Identifier' = P2,  
 'Property Value' = (any valid value for P2the value of P2 specified in the EPICS),  
 'Property Identifier' = P3,  
 'Property Value' = (any valid value for P3the value of P3 specified in the EPICS),  
 'Object Identifier' = Object2,  
 'Property Identifier' = P4,  
 'Property Value' = (any valid value for P4the value of P4 specified in the EPICS)  
 'Property Identifier' = P5,  
 'Property Value' = (any valid value for P5the value of P5 specified in the EPICS),  
 'Property Identifier' = P6  
 'Property Value' = (any valid value for P6the 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 P1the value of P1 specified in the EPICS)*,  
 'Property Identifier' = P2,  
 'Property Value' = *(any valid value for P2the 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 P4the 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 P1the value of P1 specified in the EPICS)*,  
 'Property Identifier' = P2,  
 'Property Value' = *(any valid value for P2the 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 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' = ObjectX,
    - 'Property Identifier' = OPTIONAL
  - RECEIVE ReadPropertyMultiple-ACK,
    - 'Object Identifier' = ObjectX,
  - REPEAT P = (each optional property supported by 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 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.

#### 9.20.1.X2 ReadPropertyMultiple Array Properties

Purpose: To verify that the IUT can execute ReadPropertyMultiple service requests when the requested property is an array, when its size as well as when a single element of the array is requested. Another request is made to read an element of an array where the array index is out of range.

Configuration Requirement: O1 is any object in the IUT database having array property P1 having size X.

Test Steps:

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN
3. TRANSMIT ReadPropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = 1
4. RECEIVE ReadPropertyMultiple-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = 1,
  - 'Property Value' = (V, any valid value of the correct data type for property P1)
5. TRANSMIT ReadPropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,

6.        'Property Array Index' = X,  
RECEIVE ReadPropertyMultiple-ACK,  
          'Object Identifier' = O1,  
          'Property Identifier' = P1,  
          'Property Array Index' = X,  
          'Property Value' = (V, any valid value of the correct data type for property P1)
7.        CHECK (V is any valid value of the correct data type for property P1)
8.        TRANSMIT ReadPropertyMultiple-Request,  
          'Object Identifier' = O1,  
          'Property Identifier' = P1,  
          'Property Array Index' = (X+1)
9.        RECEIVE ReadPropertyMultiple-Error,  
          Error Class = PROPERTY,  
          Error Code = INVALID\_ARRAY\_INDEX

## 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.

Configuration Requirements: The IUT must be configured with at least one non-empty list property.

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) )|  
          (ReadRange-ACK  
              'Object Identifier' = X,  
              'Property Identifier' = Y,  
              ('Result Flags' = (FALSE, FALSE, FALSE),  
              'Item Count' = (C = 0)  
              'Item Data' = ( ))  
      IF (C <> 0) THEN  
          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.

*Configuration Requirements: If all the list properties applicable for the object under testing are supported, then this test shall be skipped.*

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (any object that exists in the IUT),
  - 'Property Identifier' = (any list property *applicable for that object but* 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),  
     'Property 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  $N: 1 \leq N \leq \text{the size of the array}$ )
- ~~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  $N: 1 \leq N \leq \text{the size of the array}$ )~~  
     '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,  
 'Property 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 description 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 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 = ✗

#### 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~~ 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  $\geq$  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)

#### 9.22.2.X2 Resizing a writable fixed size array property

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WriteProperty service.

Test Concept: Select an object (O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Test Steps:

1. READ X = (O1), P1 ARRAY INDEX = 0
2. WRITE P1 = (Entire Array with any valid value greater than Array Size X)
3. RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
4. VERIFY (O1), P1 = X, ARRAY INDEX = 0
5. WRITE P1 = (Entire Array with any valid value less than Array Size X)
6. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
7. VERIFY (O1), P1 = X, ARRAY INDEX = 0
8. WRITE P1 = (any valid value greater than Array Size X), ARRAY INDEX = 0
9. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
10. VERIFY (O1), P1 = X, ARRAY INDEX = 0,
11. WRITE P1 = (any valid value less than Array Size X), ARRAY INDEX = 0
12. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
13. VERIFY (O1), P1 = X, ARRAY INDEX = 0

## 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 23)
6. VERIFY (Object1), P2 = (the value specified for P2 in step 23)

### 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),  $P1 = (\text{the value specified for this property in the EPICS})$
2. VERIFY (Object1),  $P2 = (\text{the value specified for this property in the EPICS})$
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),  $P1 = (\text{the value specified for } P1 \text{ in step 3})$
6. VERIFY (Object1),  $P2 = Y(\text{the value specified for this property in the EPICS})$

#### 9.23.2.4 Writing Non-Array Properties with an Array Index

Reason for Change: Modified test to remove dependency on EPICS values. Added Property Array Index to error received.

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 *having a value X*. An attempt will be made to write to this property using an array index Y. 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),  $P1 = (\text{the value defined for this property in the EPICS})$~~
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' = (Y, any positive integer)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Array Index' = Y
4. VERIFY (Object1),  $P1 = X(\text{the value defined for this property in the EPICS})$

#### 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 description 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 *having value X*. An attempt will be made to write to this property using an array index, Y, 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' = (Y, any ~~value~~ *positive integer* that is larger ~~than~~ *than* 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,  
     '*Property Array Index*' = Y
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) THEN  
     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

#### 9.23.2.X2 Resizing a writable fixed size array property using WritePropertyMultiple service

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WritePropertyMultiple service.

Test Concept: Select an object(O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Test Steps:

1. READ X = (O1), P1, ARRAY INDEX = 0
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (Entire Array with any valid value greater than Array Size X)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
     'ObjectIdentifier' = O1,  
     'PropertyIdentifier' = P1
4. VERIFY (O1), P1= X, ARRAY INDEX = 0
5. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (Entire Array with any valid value less than Array Size X)
6. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
     'ObjectIdentifier' = O1,  
     'PropertyIdentifier' = P1
7. VERIFY (O1), P1= X, ARRAY INDEX = 0
8. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value greater than Array Size X),  
     'Property Array Index' = 0

9. RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
 'ObjectIdentifier' = O1,  
 'PropertyIdentifier' = P1  
 'Property Array Index'=0
10. VERIFY (O1), P1= X, ARRAY INDEX = 0
11. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1,  
 'Property Value' = (any valid value less than Array Size X),  
 'Property Array Index' = 0
12. RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
 'ObjectIdentifier' = O1,  
 'PropertyIdentifier' = P1  
 'Property Array Index'= 0
13. VERIFY (O1), P1= X, ARRAY INDEX = 0

### 9.23.2.X3 Writing first element of 'List of Write Access Specifications' with Object Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported object and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to a single property in two different objects. The first object is not supported. The second object is supported and the property is writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not 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 O2 and P2 will be used to designate the writable object and property having value X used for this test. The designation Bad Object will be used to indicate an object that is not supported or not present in IUT database P1 is any valid Property Identifier.

Test Steps:

1. VERIFY (O2), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = BadObject,  
 '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)  
 'Object Identifier' = O2,  
 'Property Identifier' = P2,  
 'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = OBJECT,  
 'Error Code' = UNKNOWN\_OBJECT,  
 'Object Identifier' = BadObject,  
 'Property Identifier' = P1 |  
 (RECEIVE WritePropertyMultiple-Error,  
 'Error Class' = OBJECT,  
 'Error Code' = UNSUPPORTED\_OBJECT\_TYPE,  
 'Object Identifier' = BadObject,

'Property Identifier' = P1)

4. VERIFY (O2), P2 = X

#### 9.23.2.X4 Writing first element of 'List of Write Access Specifications' with a Write Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for a read only property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported but read only. The second property is supported and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not 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 O1 will be used to designate the object, P1 the read only property having value X, P2 the writable property having value Y used for this test.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O2), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = X ),  
    'Property Identifier' = P2,  
    'Property Value' = (any valid value not equal to Y)
4. RECEIVE WritePropertyMultiple-Error,  
    'Error Class' = PROPERTY,  
    'Error Code' = WRITE\_ACCESS\_DENIED,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1
5. VERIFY (O1), P2 = Y

#### 9.23.2.X5 WritePropertyMultiple Reject Test for first element of 'List of Write Access Specifications'

Purpose: This test case verifies that if IUT does sends a Reject-PDU or Error-PDU then the write attempt for the remaining element of 'List of Write Access Specifications' do not take place.

Test Concept: Two writable properties, P1 having value X and P2 having value Y are written to the IUT but the portion of the WritePropertyMultiple specifying P1 is made invalid by omitting the 'Property Value' parameter. The value of the properties are checked to ensure that it has not changed.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O2), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Object Identifier' = O2,  
    'Property Identifier' = P2

- 'Property Value' = (Any valid value not equal to Y))
4. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = SERVICES,  
     'Error Code' = INVALID\_TAG  
     'Object Identifier' = O1  
     'Property Identifier' = P1) |  
     (RECEIVE BACnet-Reject-PDU,  
     'Reject Reason' = INVALID\_TAG | MISSING\_REQUIRED\_PARAMETER  
         | INCONSISTENT\_PARAMETERS | INVALID\_PARAMETER\_DATA\_TYPE  
         | TOO\_MANY\_ARGUMENTS)
  4. VERIFY (O1), P1 = X
  5. VERIFY (O2), P2 = Y

#### 9.23.2.X6 Writing first element of 'List of Write Access Specifications' with a Property Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is not supported for this object. The second property is supported for this object and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not 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 O1 will be used to designate the object, P1 the unsupported property, and P2 the writable property having value X used.

Test Steps:

1. VERIFY (O1), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     '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)  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = UNKNOWN\_PROPERTY,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1
4. VERIFY (O1), P2 = X

#### 9.23.2.X9 Date Non-Pattern Properties Test using WritePropertyMultiple Service

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 {
2. TRANSMIT WritePropertyMultiple-Request  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1 updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_TAG)  
     }

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

**9.23.2.X10 Time Non-Pattern Properties Test using WritePropertyMultiple Service**

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) Do {
2. TRANSMIT WritePropertyMultiple-Request  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1 updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         'Reject Reason' = INVALID\_TAG)  
     }

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.



**9.23.2.X11 DateTime Non-Pattern Properties Test using WritePropertyMultiple Service**

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

Test Concept: The property being tested, P<sub>1</sub>, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT<sub>1</sub> is selected which is within the range that the IUT will accept for the property. The value, V<sub>1</sub>, written to the property is the datetime DT<sub>1</sub> 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 {
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (DT<sub>1</sub> updated with the special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1)  
   | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
   | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_TAG)  
   }

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

**9.23.2.X12 BACnetDateRange Non-Pattern Properties Test using WritePropertyMultiple Service**

Purpose: To verify that the property being tested does not accept special date field values, except for fully unspecified start of the range or fully unspecified end of the range, or both.

Test Concept: A BACnetDateRange property, or property that is a complex datatype containing BACnetDateRange P1 is written with each of the special field values to ensure that the property does not accept them. Each half of the dateRange DR1 is selected so it is within the range that the IUT will accept for the property. The value, V1 written to the property is the dateRange DR1 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 {
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (DR1 with startDate updated with special value SV)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = Object1,

```

 'Property Identifier' = P1
 | (BACnet-Reject-PDU
 'Reject Reason' = INVALID_PARAMETER_DATATYPE)
 | (BACnet-Reject-PDU
 'Reject Reason' = INVALID_TAG)
4. TRANSMIT WritePropertyMultiple-Request,
 'Object Identifier' = O1,
 'Property Identifier' = P1,
 'Property Value' = (DR1 with endDate updated with special value SV)
5. RECEIVE WritePropertyMultiple-Error,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE,
 'Object Identifier' = Object1,
 'Property Identifier' = P1)
 | (BACnet-Reject-PDU
 'Reject Reason' = INVALID_PARAMETER_DATATYPE)
 | (BACnet-Reject-PDU
 'Reject Reason' = INVALID_TAG)
}

```

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

## 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 *any appropriate*  
         Error Code of COMMUNICATION\_DISABLED, 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

## 9.25 ConfirmedPrivateTransfer Service Execution Tests

Reason for change: renumbered so negative tests could be added

This clause defines the tests necessary to demonstrate support for executing ConfirmedPrivateTransfer service requests.

### 9.25.1 Positive ConfirmedPrivateTransfer Service Execute Tests

#### 9.25.1.1 Correctly Executes a Supported ConfirmedPrivateTransfer Service

Dependencies: None.

BACnet Reference Clause: 16.2.

Purpose: To verify the ability to correctly execute a ConfirmedPrivateTransfer service request.

Test Concept: The service procedure implied by a particular private transfer service is defined by the vendor. This test simply verifies that an appropriate acknowledgment is returned and that any externally visible actions defined by the vendor are observed.

Configuration Requirements: The IUT shall be configured to execute at least one ConfirmedPrivateTransfer service. The service parameters that are to be provided in the request and a list of any externally visible actions that should be apparent to the tester shall also be provided.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the  
                   EPICS),  
     'Service Number' = (any service number provided by the vendor),  
     'Service Parameters' = (the service parameters provided for this service)
2. RECEIVE ConfirmedPrivateTransfer-ACK,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the  
                   EPICS),  
     'Service Number' = (the service number used in step 1),  
     'Result Block' = (the expected results provided by the vendor)
3. CHECK (Did the externally visible actions take place?)

#### 9.25.2.1 Correctly Executes a Non-Supported ConfirmedPrivateTransfer Service

Reason for Change: There is no existing test verifying correct error responses for unknown ConfirmedPrivateTransfer services.

Purpose: To verify that the IUT correctly responds with an error when an unsupported ConfirmedPrivateTransfer is received.

Test Concept: Send a non-supported ConfirmedPrivateTransfer request to the IUT and verify that it responds with an error or rejects the service.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (VID: any valid value not supported by the IUT),  
     'Service Number' = (SID: any valid value, that when combined with Vendor ID is not supported by the IUT),  
     'Service Parameters' = (any valid values)
2. RECEIVE  
     (ConfirmedPrivateTransfer-Error,  
         'Error Class' = any  
         'Error Code' = any  
         'Vendor ID' = (VID),  
         'Service Number' = (SID),  
         'Result Block' = (the expected results provided by the vendor)) |  
     (BACnet-Reject-PDU,  
         'Reject Reason' = any reason) |  
     (BACnet-Abort-PDU  
         'Abort Reason' = any reason)
3. CHECK (that the IUT exhibits the vendor defined results)

## 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 = ~~SERVICES~~,  
        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 = ~~SERVICES~~,  
        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 = ~~SERVICES~~,  
            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)~~  
 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)~~  
 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)~~

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.

1. READ InitialDate = Local\_Date  
 2. READ InitialTime = Local\_Time  
 3. TRANSMIT  
     DA = LOCAL BROADCAST,  
     SA = TD,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = TimeSynchronization-Request,  
     date = NewDate; combined with NewTime is different than the InitialDate/InitialTime  
             pair  
     time = NewTime; combined with NewDate is different than the InitialDate/InitialTime  
             pair  
 4. VERIFY Local\_Date = NewDate  
 5. VERIFY Local\_Time ~= NewTime

Notes to Tester: Select date and time such that either one or both of them is different from initial date and time.

### 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 95 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:*

~~Test Steps: The test steps are identical to the steps in 9.30.1.1 except that in step 9 the UTCTimeSynchronization request is used and the date and time conveyed represent UTC.~~

~~Passing Results: The passing results are identical to 9.30.1.1 except that the date in step 9 shall be corrected for InitialUTC\_Offset, and the time in step 13 shall be corrected for both Initial\_UTC\_Offset and Daylight\_Savings\_Status (as defined in BACnet 16.7.2).~~

1. READ InitialDate = Local\_Date
2. READ InitialTime = Local\_Time
3. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = UTCTimeSynchronization-Request,
  - date = NewUtcDate: combined with NewUtcTime and converted to local time is different than the InitialDate/InitialTime pair
  - time = NewUtcTime: combined with NewUtcDate and converted to local time is different than the InitialDate/InitialTime pair
4. VERIFY Local\_Date = (NewUtcDate converted to local date/time using UTC\_Offset and Daylight\_Saving\_Status)
5. VERIFY Local\_Time ~= (NewUtcTime converted to local date/time using UTC\_Offset and Daylight\_Saving\_Status)

*Notes to Tester: Select date and time such that either one or both of them is different from initial date and time. The IUT may update the Daylight\_Savings\_Status during the execution of the UTCTimeSynchronization request.*

#### 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 9.30.1.1 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. The allowance for Unicast I-Have is added.

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~~ **Unconfirmed Response Fail Time**
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~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. The allowance for Unicast I-Have is added.

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~~ **Unconfirmed Response Fail Time**
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~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. The allowance for Unicast I-Have is added.

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:  $0 \leq L <$  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~~ **Unconfirmed Response Fail Time**
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
  - ~~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. The allowance for Unicast I-Have is added.

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:  $0 \leq L <$  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~~ **Unconfirmed Response Fail Time**
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | *TD*,
  - ~~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. The allowance for Unicast I-Have is added.

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:  $0 \leq L <$  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~~ **Unconfirmed Response Fail Time**
4. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST | *TD*,
  - ~~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. The allowance for Unicast I-Have is added.

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~~ **Unconfirmed Response Fail Time**

4. RECEIVE  
 DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 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. The allowance for Unicast I-Have is added.

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:  $0 \leq L <$  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~~ **Unconfirmed Response Fail Time**

4. RECEIVE  
 DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 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. The allowance for Unicast I-Have is added.

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~~ **Unconfirmed Response Fail Time**
  4. RECEIVE  
DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
——— SA = IUT,  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = ~~Object1~~(the object identifier specified in step 1),

#### 9.32.1.11 Object Name Version, Directed to a Specific MAC Address

Reason for Change: Modified test to remove dependency on EPICS values. The allowance for Unicast I-Have is added.

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~~ **Unconfirmed Response Fail Time**
4. RECEIVE  
DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
——— 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)

#### 9.32.1.12 Who-Has After Object\_Name Changed

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send a unicast response.

Dependencies: Who-Has Service Execution Tests, 9.32.1.2

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Name property of an object in the device is changed.

Test Concept: The Object\_Name property of the referenced object is read to determine its initial value. The Object\_Name property is then changed to a different value, V2, which is not already used by an object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Name' parameter, using the values V1 and V2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Name property and has the value V1. If IUT does not support objects with modifiable Object\_Name properties, then this test shall be skipped.

Test Steps:

1. READ V1 = O1, Object\_Name
2. IF (Object\_Name is writable) THEN  
WRITE O1, Object\_Name = V2  
ELSE  
MAKE (O1, Object\_Name = V2)
3. TRANSMIT

- DESTINATION = GLOBAL BROADCAST,  
Who-Has-Request,  
'Object Name' = V1
4. WAIT ~~Internal Processing Fail Time~~ *Unconfirmed Response Fail Time*
  5. CHECK (Verify that the IUT does not respond with an I-Have request)
  6. TRANSMIT  
DESTINATION = GLOBAL BROADCAST,  
Who-Has-Request,  
'Object Name' = V2
  7. RECEIVE ~~DESTINATION~~ *DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST | *TD*,  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = O1,  
'Object Name' = V2

### 9.32.1.13 Who-Has After Object\_Identifier Changed

Reason for Change: The BACnet standard (per addendum 135-2012ar-5) now allows the IUT to send a unicast response.

Dependencies: Who-Has Service Execution Tests, 9.32.1.1

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Identifier property of an object in the device is changed.

Test Concept: The Object\_Identifier property of the referenced object, O1, is verified to contain the value O1. The Object\_Identifier property is then changed to a different value, O2, which is not already in use by a different object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Identifier' parameter, using the values O1 and O2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Identifier property. If the IUT does not support objects with modifiable Object\_Identifier, then this test shall be skipped.

Test Steps:

1. VERIFY O1, Object\_Identifier = O1
2. IF (O1 is writable) THEN  
WRITE O1, Object\_Identifier = O2  
ELSE  
MAKE (O1, Object\_Identifier = O2)
3. TRANSMIT  
DESTINATION = GLOBAL BROADCAST,  
Who-Has-Request,  
'Object Identifier' = O1
4. WAIT ~~Internal Processing Fail Time~~ *Unconfirmed Response Fail Time*
5. CHECK (Verify that the IUT does not respond with an I-Have request)
6. TRANSMIT  
DESTINATION = GLOBAL BROADCAST,  
Who-Has-Request,  
'Object Identifier' = O2
7. RECEIVE ~~DESTINATION~~ *DESTINATION* = LOCAL BROADCAST | GLOBAL BROADCAST | *TD*,  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = O2



'Object Name' = *VI*(~~the object name specified in the EPICS for this object~~)

### 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. The allowance for Unicast I-Have is added.

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 *VI* = (*Object1*), *Object\_Name*
2. TRANSMIT  
~~DESTINATION = LOCAL BROADCAST,~~  
~~SA = TD,~~  
DNET = GLOBAL BROADCAST,  
SNET = (*X*: any remote network number),  
SADR = (*Y*: 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~~ **Unconfirmed Response Fail Time**
4. RECEIVE  
DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network *X*~~specified in step 1~~)  
| TD (DNET = *X*, DADR = *Y*),  
I-Have-Request,  
'Device Identifier' = (the IUT's Device object),  
'Object Identifier' = *Object1*(~~the object identifier specified in step 1~~),  
'Object Name' = *VI*(~~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. The allowance for Unicast I-Have is added.

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 *VI* = (*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~~ **Unconfirmed Response Fail Time**
4. RECEIVE  
DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network *X*~~specified in step 1~~)  
| TD (DNET = *X*, DADR = *Y*),  
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  $\geq 0$  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  $\geq$  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)

#### 9.33.2 Execution of Who-Is Service Requests Originating from a Remote Network

##### 9.33.2.3 General Inquiry, Directed to a Remote Device

Purpose: To verify that the IUT responds with an I-Am service that is of the form global broadcast, remote broadcast or unicast the ability of the IUT to recognize the origin of a Who-Is service request, directed to the IUT, and respond such that the device originating the request receives the response.

Test Steps:

1. TRANSMIT
  - DESTINATION = IUT,
  - SNET = (any remote network number),
  - SADR = (any MAC address valid for the specified network),
  - Who-Is-Request
2. WAIT ~~Internal Processing~~ **Unconfirmed Response Fail Time**
3. RECEIVE
  - DESTINATION = GLOBAL BROADCAST | ~~LOCAL BROADCAST~~ REMOTE BROADCAST (to the network specified by SNET in step 1) | TD
  - I-Am-Request,
  - 'I Am Device Identifier' = (the IUT's Device object),
  - 'Max APDU Length Accepted' = (the value specified in the EPICS),
  - 'Segmentation Supported' = (the value specified in the EPICS),
  - 'Vendor Identifier' = (the identifier registered for this vendor)

### 9.X40 WriteGroup Tests

#### 9.X40.1.X1 Channel and Group Number Test

Purpose: To verify that the Channel object executes a WriteGroup service request only when containing a specified channel number and Group Number by a request, and the Channel object ignores a request otherwise. If a Group Number is 0, the Channel object ignores a service even when its Control\_Groups property is set to 0.

Test Concept: The Channel Object, O1 will be assigned a specific value to its channel number and Group Number. When a device containing O1 receives a WriteGroup service, O1 executes the request and propagate a specified value to its destination only if A) the O1's channel number is the same as specified number by the request and B) the O1's Control\_Groups contains the specified Group Number, except for a case when a Group Number was 0.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commandable property of an object on either a local or remote device. For a commandable property, all prioritized commands has to be relinquished and any minimum on/off time has to be accounted for prior to the test. An initial value of a commandable property must be the same as RD, which is its Relinquish\_Default value. The value to be propagated must be a valid value that does not require coercion.

#### Test Steps:

- Obtain the data which will be used for the Channel Write Fail Time later in the steps
- 1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Obtain the Channel Object's target object reference
- 2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
- Set arbitrary numbers for Channel\_Number and Control\_Group to O1
- 3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = Channel\_Number
  - 'Property Value' = (CN: Any valid value)
  - 'Property Identifier' = Control\_Groups
  - 'Property Value' = (CG: Any length of an array containing at least 1 non-zero element)
- 4. RECEIVE BACnet-Simple-ACK-PDU
- Send a WriteGroup with a mismatching channel number and Group Number
- 5. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (A valid value larger than 0 and not contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
- 6. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate a value to its target references
- 7. VERIFY PR = RD
- Send a WriteGroup with a matching channel number and a mismatching group number
- 8. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (A valid value larger than 0 and not contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)
- 9. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate value to its target references
- 10. VERIFY PR = RD
- Send a WriteGroup with a mismatching channel number and a matching group number
- 11. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (Any non 0 values contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
- 12. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate value to its target references
- 13. VERIFY PR = RD
- Send a WriteGroup service with a matching channel number and group number
- 14. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (Any non 0 values contained by CG)
  - 'Write Priority' = (any valid value),
  - 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)
- Make sure that O1 did propagate value to its target references
- 15. VERIFY PR = Y
- Change Control\_Groups to 0

16. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = Control\_Groups  
     'Property Value' = 0
17. RECEIVE BACnet-Simple-ACK-PDU  
     -- Send a WriteGroup with 0 Group number
18. TRANSMIT WriteGroup-Request,  
     'Group Number' = 0  
     'Write Priority' = (any valid value),  
     'Change List' = (CN, no overriding priority, Z: a valid value different than Y)
19. WAIT **Channel Write Fail Time** \* LEN  
     -- Make sure that O1 did not propagate value to its target references
20. VERIFY PR = Y

#### 9.X40.1.X2 Write Priority and Overriding Priority Test

Purpose: To verify that the overridingPriority, if provided, specifies the priority for writing the value. Otherwise the 'Write Priority' parameter specifies the priority for writing.

Test Concept: The Channel Object, O1 receives the WriteGroup with P1 as its 'Write Priority' and it is verified that P1 is used for writing the value. O1 then receives another WriteGroup with P1 as its 'Write Priority' and P2 as its overridingPriority and it is verified that P2 is used for writing the value.

Configuration Requirements: Configure one of the entry for the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with Priority\_Array on either a local or remote device. All prioritized commands has to be relinquished and any minimum on/off time has to be accounted for prior to the test.

Test Steps:

- Obtain the data which will be used for the Channel Write Fail Time later in the steps
1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Write to O1 using P1 as its Write Priority
2. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = (P1: Any valid value)  
     'Change List' = (O1's channel number, no overriding priority, Y: any valid value)
3. WAIT **Channel Write Fail Time** \* LEN  
     -- Make sure that P1 is used for writing the value
4. VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1  
     -- Write to O1 using P1 as its Write Priority, P2 as its overridingPriority
5. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = P1  
     'Change List' = (O1's channel number, P2: Any valid value different than P1, Z: any valid value different than Y)
6. WAIT **Channel Write Fail Time** \* LEN  
     -- Make sure that P2 is used for writing the value
7. VERIFY (O2), Priority\_Array = Z, ARRAY INDEX = P2  
     -- Make sure that no update on Priority\_Array[P1]
8. VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1

#### 9.X40.1.X3 Relinquish Control Test

Purpose: To verify that if a BACnetGroupChannelValue specifies a NULL value, it serves the same function as if NULL had been used with WriteProperty.

Test Concept: The Channel Object, O1 receives the WriteGroup service to propagate a value to its destination object property reference, PR. PR is verified to have the value updated accordingly. The O1 then receives another WriteGroup service with the BACnetGroupChannelValue specifying a NULL value. PR is verified to have its Relinquish\_Default value.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with a Relinquish\_Default set to RD on either a local or remote device. All prioritized commands has to be relinquished and any minimum on/off time has to be accounted for prior to the test.

Test Steps:

- Obtain the data which will be used for the Channel Write Fail Time later in the steps
- 1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Obtain the Channel Object's target object reference
- 2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
- Let the Channel Object propagate a value to its target
- 3. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (One of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, X: any valid value different than RD)
- 4. WAIT **Channel Write Fail Time** \* LEN
- 5. VERIFY PR = X
- Let the Channel Object relinquish control of the target
- 6. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (One of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, NULL)
- 7. WAIT **Channel Write Fail Time** \* LEN
- 8. VERIFY PR = RD

#### 9.X40.1.X4 Inhibit Delay Test with WriteGroup

Purpose: In the case of WriteGroup, verify that Execution\_Delay always occurs unless the WriteGroup service parameter 'Inhibit Delay' is TRUE, and the Channel object property Allow\_Group\_Delay\_Inhibit is present and has the value TRUE.

Test Concept: Setup List\_Of\_Object\_Property\_References of the Channel Object, O1 to contain 2 valid entries PR1, and PR2 and provide each with an execution delay (ED1 and ED2) which are larger than 0. Allow\_Group\_Delay\_Inhibit is set to TRUE. When a WriteGroup service is sent to O1 without 'Inhibit Delay' parameter, it is verified that Execution\_Delay occurs. When another WriteGroup service is sent to O1 with 'Inhibit Delay' set to False, it is verified that Execution\_Delay still occurs. Finally, when another WriteGroup service is sent with 'Inhibit Delay' set to TRUE, it is verified that Execution\_Delay does not occur.

Configuration Requirements: PR1 and PR2 shall be references to writable properties and shall be the same datatype. ED1 and ED2 shall be values which are large enough that the delay between writes is sufficient for the test. V1 and V2 shall be of the expected datatype for PR1 and PR2 so that no coercion occur, and shall be different values.

-- Setup the Channel object

- 1. WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)
- 2. WRITE Execution\_Delay = (ED1, ED2)
- 3. WRITE Allow\_Group\_Delay\_Inhibit = TRUE
- 4. READ V1 = PR1
- 5. READ V2 = PR2
- Send a WriteGroup without 'Inhibit Delay' parameter
- 6. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)

'Change List' = (O1's channel number, no overriding priority, X: any valid value different than V1 or V2)

7. **WAIT Channel Write Fail Time**  
 -- Make sure that Execution\_Delay occurs
8. VERIFY PR1= V1
9. VERIFY PR2 = V2
10. WAIT (ED1)
11. VERIFY PR1 = X
12. VERIFY PR2 = V2
13. WAIT (ED2 – ED1)
14. VERIFY PR2 = X
- Send a WriteGroup with 'Inhibit Delay' set to FALSE
15. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = (Any valid value)  
     'Change List' = (O1's channel number, no overriding priority, Y: any valid value different from X)  
     'Inhibit Delay' = FALSE
16. **WAIT Channel Write Fail Time**  
 -- Make sure that Execution\_Delay occurs
17. VERIFY PR1= X
18. VERIFY PR2 = X
19. WAIT (ED1)
20. VERIFY PR1 = Y
21. VERIFY PR2 = X
22. WAIT (ED2 – ED1)
23. VERIFY PR2 = Y
- Send a WriteGroup with 'Inhibit Delay' set to TRUE
24. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = (Any valid value)  
     'Change List' = (O1's channel number, no overriding priority, Z: any valid value different from Y)  
     'Inhibit Delay' = TRUE
- Make sure that Execution\_Delay does NOT occurs
25. **WAIT Channel Write Fail Time**
26. VERIFY PR1= Z
27. VERIFY PR2 = Z

## 10. NETWORK LAYER PROTOCOL TESTS

### 10.1.1 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*)

## 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.

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,
  - Rejection Reason = 3 (unknown network layer message type),
  - DNET = (any value)

#### 10.2.3.2 Route Message from a Local Device to a Local Device

Reason For Change: Modified test to support new Extended MSTP Frame testing.

Purpose: To verify that the IUT can route a *maximum sized NPDU* unicast message from a local device on Network 1 to a device on Network 2.

*Test Concept: A message is sent from Network 1 destined for Network 2 via the IUT and router functionality is verified by observing the message on Network 2. The sequence is repeated in the opposite direction to verify the IUT can also route*



messages from Network 2 to Network 1. The messages may have an NPDU length, *L*, such that *L* equals the Maximum NPDU length as defined in Table 6-1 for the smaller data link.

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SA = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = (*OI* = any object identifier),  
     'Property Identifier' = (*PI* = any property of the specified object with a *CharacterString* datatype),  
     'Property Value' = (*V* = *CharacterString* with a length such that the NPDU length = *L*)
2. RECEIVE PORT B,  
     DA = D2C,  
     SA = IUT,  
     SNET = 1,  
     SADR = D1A,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = ~~*OI*(any object identifier),~~  
     'Property Identifier' = ~~*PI* (any property of the specified object),~~  
     'Property Value' = ~~*V*~~
3. TRANSMIT PORT B,  
     DA = IUT,  
     SA = D2C,  
     DNET = 1,  
     DADR = D1A,  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = (*OI* = any object identifier),  
     'Property Identifier' = (*PI* = any property of the specified object with a *CharacterString* datatype),  
     'Property Value' = (*V* = *CharacterString* with a length such that the NPDU length = *L*)
4. RECEIVE PORT A,  
     DA = D1A,  
     SA = IUT,  
     SNET = 2,  
     SADR = D2C,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ~~ReadProperty-Request~~ WriteProperty-Request,  
     'Object Identifier' = ~~*OI*(any object identifier),~~  
     'Property Identifier' = ~~*PI* (any property of the specified object),~~  
     'Property Value' = ~~*V*~~

## 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

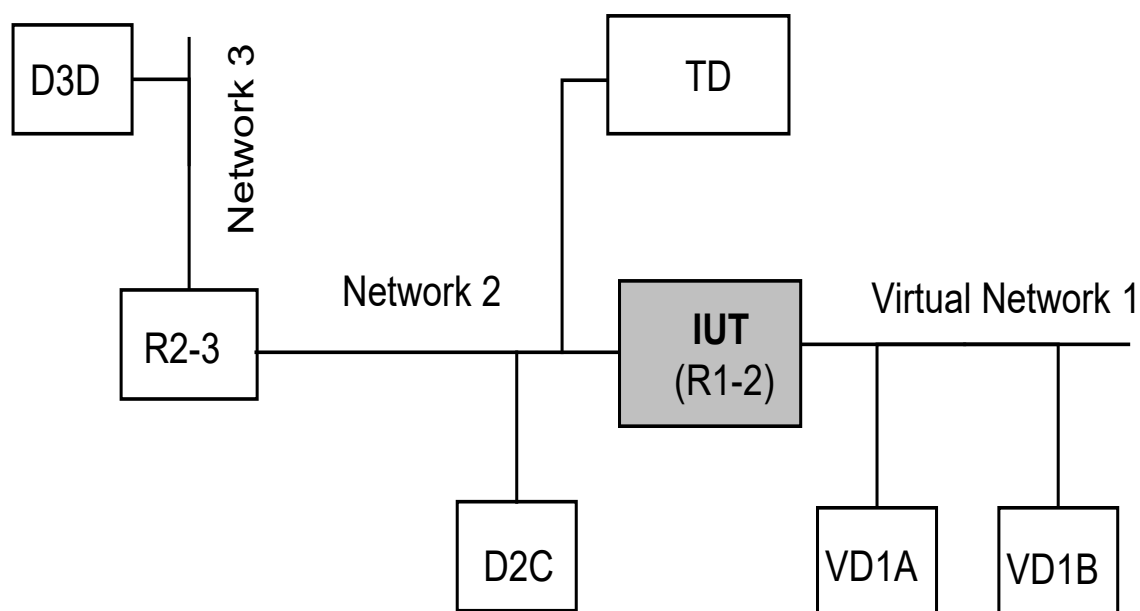
Some devices (typically gateways) can route BACnet packets between a physical BACnet LAN and one or more virtual BACnet LANs that contain one or more virtual BACnet devices. See H.1 and H.2 in the BACnet standard for a description of virtual BACnet LANs and virtual BACnet devices.

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)

#### 10.8.3.6.X1 Silently Drop Messages to a Virtual Device that is Offline

Purpose: To verify that the IUT does not return any message in response to an NPDU with a destination that is offline.

Test Concept: The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which is derived from the data in a virtual device is read from the IUT. Verify that when a virtual device is off-line, that the IUT sends no response to messages that are directed to that off-line device.



**Configuration Requirements:** The IUT acting as a virtual router, shall be configured so that a virtual device VD1A which can sometimes be online, is initially online for this test. If no virtual device can become off-line, then this test shall be skipped.

**Test Steps:**

1. CHECK (any vendor-specified indication, that the virtual device is online)
2. MAKE (the virtual device containing Object1 go offline)
3. MAKE (the IUT notice that the virtual device is offline)
4. TRANSMIT ReadProperty-Request,  
     DESTINATION = V1DA  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1
5. CHECK (that no responsive message is returned from IUT)
6. TRANSMIT  
     DESTINATION = VD1A,  
     Message Type = (any valid value)
7. CHECK (that no responsive message is returned from IUT)

#### 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 $T_{\text{frame\_gap}}$

Reason for Change: Added 'Configuration 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     |

#### 12.1.3.X1 Frame Type Based on Transmitted NPDU size

Reason for Change: No test exists for this functionality. This test is not in any SSPC proposal.

Purpose: To verify that the IUT selects the correct frame type based on the transmitted NPDU size.

Test Concept: The IUT is made to send a frame such that the NPDU size is less than or equal to 501 octets (a non-COBS encoded frame) and the frame type is checked. The IUT is then made to send a frame such that the NPDU size is greater than 501 octets (a COBS encoded frame) and the frame type is checked.

It is expected that this test can be executed using ReadPropertyMultiple service requests to generate responses from the IUT of different sizes. If the IUT does not support execution of ReadPropertyMultiple service requests and the IUT cannot be made to send a frame larger than 501 octets by any other means, this test shall be skipped.

Test Steps:

1. MAKE (the IUT generate a frame with an NPDU less than 501 octets)
2. CHECK (Frame type = 4 or 5)
3. MAKE (the IUT generate a frame with an NPDU greater than 501 octets)
4. CHECK (Frame type = 32 or 33)

#### 12.1.3.X2 Executing COBS Encoded Frames

Purpose: To verify that the IUT can properly execute COBS encoded frames

Test Concept: A COBS encoded service request is sent to the IUT and proper execution of the request is verified.

It is expected that this test can be executed for server devices using a large ReadPropertyMultiple service request that the server can execute. If the IUT does not support execution of ReadPropertyMultiple service requests, the vendor must provide instructions and means for verifying correct execution of a request.

Test Steps:

1. IF (the IUT supports execution of ReadPropertyMultiple service requests) THEN
 

READ V = (Object1), P1

TRANSMIT (a ReadPropertyMultiple request with an NPDU larger than 501 octets including (Object1), P1)

VERIFY (Object1), P1 = V
2. ELSE
 

(Use vendor supplied instructions to verify execution of a service request)

### 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

##### 13.1.12.X1 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

### 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. Changed test to account for optional properties.

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
  - IF (Backup\_Preparation\_Time is present) THEN
    - READ BPT = Backup\_Preparation\_Time
  - ELSE
    - READ BPT = APDU\_Timeout
  - IF (Restore\_Preparation\_Time is present) THEN
    - READ RPT = Restore\_Preparation\_Time
  - ELSE
    - READ RPT = APDU\_Timeout
  - IF (Restore\_Completion\_Time is present) THEN
    - READ RCT = Restore\_Completion\_Time
  - ELSE

```

 READ RCT = APDU_Timeout
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 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
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 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) THEN
 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 (*Backup\_And\_Restore\_State is present or (Protocol\_Revision is present and Protocol\_Revision ≥ 10/13)*) 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  
    IF (*Backup\_And\_Restore\_State is present or Protocol\_Revision ≥ 13*) THEN  
        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) THEN  
        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) THEN  
        WRITE X, File\_Size = 0  
    IF (Y = RECORD\_ACCESS) THEN  
        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 (Backup_And_Restore_State is present or (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
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 VERIFY Backup_And_Restore_State = IDLE
26. READ DR3 = Database_Revision
27. CHECK (DR3 \neq DR1)
28. IF (Database_Revision was changed in step 16) THEN
 CHECK (DR3 \neq 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]
}

```

### 13.8.1.2 Attempting a Backup Procedure While Already Performing a Backup Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Backup Procedure from one client and then is commanded to start a Backup Procedure from a different client.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq 10$ ) THEN
 IF (*Backup\_Preparation\_Time is present*) THEN
 READ BPT = Backup\_Preparation\_Time
 ELSE
 READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,
 'Reinitialized State of Device' = STARTBACKUP,
 'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq 10$ ) THEN
 WAIT BPT
5. TRANSMIT
 SNET = (N, any remote network number),
 SADR = (M, any MAC address valid for the specified network),
 ReinitializeDevice-Request,



- 'Reinitialized State of Device' = STARTBACKUP,
- 'Property Identifier' = (any valid password),
- 6. RECEIVE
  - DNET = N,
  - DADR = M,
  - BACnet-Error PDU,
  - Error Class = DEVICE,
  - Error Code = CONFIGURATION\_IN\_PROGRESS
- 7. TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = ENDBACKUP,
  - 'Property Identifier' = (any valid password)
- 8. RECEIVE Simple-ACK-PDU

### 13.8.1.3 Attempting a Backup Procedure While Already Performing a Restore Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Restore Procedure.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN
  - IF (Restore\_Preparation\_Time is present) THEN*
  - READ RPT = Restore\_Preparation\_Time
  - ELSE*
  - READ RPT = APDU Timeout*
  - IF (Restore\_Completion\_Time is present) THEN*
  - READ RCT = Restore\_Completion\_Time
  - ELSE*
  - READ RCT = APDU Timeout*
2. TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = STARTRESTORE,
  - 'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN
  - WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = STARTBACKUP,
  - 'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,
  - Error Class = DEVICE,
  - Error Code = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
  - 'Reinitialized State of Device' = ABORTRESTORE,
  - 'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU
9. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN
  - WAIT RCT

Note 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.

#### 13.8.1.4 Attempting a Restore Procedure While Already Performing a Backup Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Restore Procedure from one client and then is commanded to start a Restore Procedure from a different client.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN  
         READ BPT = Backup\_Preparation\_Time  
     ELSE  
         READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTBACKUP,  
     'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT BPT
5. TRANSMIT  
     SNET = (N, any remote network number),  
     SADR = (M, any MAC address valid for the specified network),  
     ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTRESTORE,  
     'Property Identifier' = (any valid password),
6. RECEIVE  
     DNET = N,  
     DADR = M,  
     BACnet-Error PDU,  
     Error Class = DEVICE,  
     Error Code = CONFIGURATION\_IN\_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ENDBACKUP,  
     'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU

#### 13.8.1.5 Attempting a Restore Procedure While Already Performing a Restore Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Restore Procedure.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN

```

 READ RCT = Restore_Completion_Time
 ELSE
 READ RCT = APDU_Timeout
2. TRANSMIT ReinitializeDevice-Request,
 'Reinitialized State of Device' = STARTRESTORE,
 'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
 WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
 'Reinitialized State of Device' = STARTRESTORE,
 'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,
 Error Class = DEVICE,
 Error Code = CONFIGURATION_IN_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
 'Reinitialized State of Device' = ABORTRESTORE,
 'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
 WAIT RCT

```

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.

#### 13.8.1.6 Ending Backup and Restore Procedures via Timeout

Reason For Change: Modified how the test WAITs for Protocol\_Revision < 10. Changed test to account for optional properties.

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
 IF (Backup_Preparation_Time is present) THEN
 READ BPT = Backup_Preparation_Time
 ELSE
 READ BPT = APDU_Timeout
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
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 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 (Backup_And_Restore_State is present or (Protocol_Revision is present and Protocol_Revision ≥ 10/13)) 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
 IF (Restore_Preparation_Time is present) THEN
 READ RPT = Restore_Preparation_Time
 ELSE
 READ RPT = APDU_Timeout
 IF (Restore_Completion_Time is present) THEN
 READ RCT = Restore_Completion_Time
 ELSE
 READ RCT = APDU_Timeout
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
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 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
 IF (Backup_And_Restore_State is present or Protocol_Revision ≥ 13) THEN
 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.

### 13.8.1.7 Ending Backup and Restore Procedures via Abort

Reason for Change: Changed test to account for optional properties.

Purpose: This test case verifies that the IUT will leave the BACKUP\_IN\_PROGRESS and DOWNLOAD\_IN\_PROGRESS states upon a command to abort.

Test Steps:

```

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
 IF (Backup_Preparation_Time is present) THEN
 READ BPT = Backup_Preparation_Time
 ELSE
 READ BPT = APDU_Timeout
2. TRANSMIT ReinitializeDevice-Request,

```

- 'Reinitialized State of Device' = STARTBACKUP,
- 'Password' = (any valid password)
- 3. RECEIVE BACnet-Simple ACK-PDU
- 4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT BPT
- 5. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ENDBACKUP,  
    'Password' = (any valid password)
- 6. RECEIVE BACnet-Simple ACK-PDU
- 7. IF (*Backup\_And\_Restore\_State is present or* (Protocol\_Revision is present and Protocol\_Revision  $\geq$  ~~10~~13)) THEN  
    VERIFY Backup\_And\_Restore\_State = IDLE
- 8. VERIFY System\_Status != BACKUP\_IN\_PROGRESS
- 9. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    *IF (Restore\_Preparation\_Time is present) THEN*  
        READ RPT = Restore\_Preparation\_Time  
    *ELSE*  
        READ RPT = APDU\_Timeout  
    *IF (Restore\_Completion\_Time is present) THEN*  
        READ RCT = Restore\_Completion\_Time  
    *ELSE*  
        READ RCT = APDU\_Timeout
- 10. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTRESTORE,  
    'Password' = (any valid password)
- 11. RECEIVE BACnet-Simple ACK-PDU
- 12. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT RPT
- 13. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ABORTRESTORE,  
    'Password' = (any valid password)
- 14. RECEIVE BACnet-Simple ACK-PDU
- 15. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT RCT  
    *IF (Backup\_And\_Restore\_State is present or Protocol\_Revision  $\geq$  13) THEN*  
        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  $\geq$  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)
```

#### 13.8.1.10 Starting and Ending a Backup Procedure when a Password is not Required

Reason for Change: Changed test to account for optional properties.

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  $\geq$  10) THEN  
     IF (Backup\_Preparation\_Time is present) THEN  
         READ BPT = Backup\_Preparation\_Time  
     ELSE  
         READ BPT = APDU\_Timeout
2. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = STARTBACKUP,  
     'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
     WAIT BPT
5. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = ENDBACKUP,  
     'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Backup\_And\_Restore\_State is present or (Protocol\_Revision is present and Protocol\_Revision  $\geq$  40/3)) THEN  
     VERIFY Backup\_And\_Restore\_State = IDLE
8. VERIFY System\_Status != BACKUP\_IN\_PROGRESS

#### 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. Changed test to account for optional properties.

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  $\geq$  10) THEN  
     IF (Restore\_Preparation\_Time is present) THEN  
         READ RPT = Restore\_Preparation\_Time  
     ELSE  
         READ RPT = APDU\_Timeout  
     IF (Restore\_Completion\_Time is present) THEN  
         READ RCT = Restore\_Completion\_Time  
     ELSE  
         READ RCT = APDU\_Timeout

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  $\geq$  10) THEN  
    WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ~~END~~ABORTRESTORE,  
    'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT RCT
8. VERIFY System\_Status != DOWNLOAD\_IN\_PROGRESS

#### 13.8.1.12 System\_Status during a Backup Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: This test case verifies that the IUT correctly sets its System\_Status during a Backup procedure. If the IUT does not change its operational behavior during a Backup Procedure, then this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    *IF (Backup\_Preparation\_Time is present) THEN*  
        READ BPT = Backup\_Preparation\_Time  
    *ELSE*  
        *READ BPT = APDU\_Timeout*
2. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = STARTBACKUP,  
    'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    WAIT BPT
5. VERIFY System\_Status = BACKUP\_IN\_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = ENDBACKUP,  
    'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. WAIT a vendor specified period of time for the device to complete the backup operation
9. VERIFY System\_Status != BACKUP\_IN\_PROGRESS

#### 13.8.1.13 System\_Status during a Restore Procedure

Reason for Change: Changed test to account for optional properties.

Purpose: This test case verifies that the IUT correctly sets its System\_Status during a Restore procedure. If the IUT does not change its operational behavior during a Restore Procedure, this test shall be omitted.

Test Steps:

1. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  10) THEN  
    *IF (Restore\_Preparation\_Time is present) THEN*



- ```

    READ RPT = Restore_Preparation_Time
  ELSE
    READ RPT = APDU_Timeout
  IF (Restore_Completion_Time is present) THEN
    READ RCT = Restore_Completion_Time
  ELSE
    READ RCT = APDU_Timeout
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RPT
5. VERIFY System_Status = DOWNLOAD_IN_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ABORTRESTORE,
   'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RCT
9. 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.X13 General Application State Machine Tests

13.X13.1 Ignore Confirmed Broadcast Requests

Reason for Change: No existing test.

Purpose: This test case verifies that the IUT will quietly discard any Confirmed-Request-PDU, whose destination address is a multicast or broadcast address, received from the network layer.

Test Concept: The TD transmits the Confirmed-Request-PDU services whose destination address is a multicast or broadcast address. The IUT is required to silently drop the requests because it should only respond to unicast confirmed requests.

Test Steps:

1. TRANSMIT Any BACnet-Confirmed-Request-PDU,
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,
2. CHECK (that the IUT does not send any packets in response to above Confirmed-Request-PDU)

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)

14.1.X11 Processing Forwarded-NPDU request initiated from different port

Purpose: To verify that an IUT will correctly process a Forwarded-NPDU message received from a device located at an address where it has a different UDP port number from those in the source and destination of a Forwarded-NPDU.

Test Concept: The IUT and the TD (acting as a BBMD) are configured such that they have the same UDP port number (P1). The originating device (D2) is selected having different UDP port number (P2) than the IUT and TD. The behavior of the IUT is verified when it correctly responds to the Forwarded-NPDU message from the device having different UDP number.

Configuration Requirements: The IUT is on the same subnet as the TD and on the same port number (P1). D2 is a device on a different subnet and has an address using port P2.

Test Steps:

1. TRANSMIT Forwarded-NPDU,
 Originating-Device = D2 -- (with UDP port P2)
 NPDU = Who-Is
2. IF (the IUT responds with Unicast I-Am) THEN
 RECEIVE Original-Unicast-NPDU,
 DESTINATION = D2, -- (with UDP port P2)
 NPDU = I-Am
 ELSE
 RECEIVE Original-Broadcast-NPDU -- (with UDP port P1.)
 NPDU = I-Am

14.1.X12 Processing Forwarded-NPDU request initiated from different port when registered as a Foreign Device into a BBMD.

Purpose: To verify that an IUT when configured as a Foreign Device into a BBMD, will correctly process a Forwarded-NPDU message received from a device located at an address where it has a different UDP port number from those in the source and destination of a Forwarded-NPDU when registered as a Foreign Device.

Test Concept: The IUT and the TD, acting as the BBMD are configured such that they have the same UDP port number (P1). The IUT must be on a different IP subnet than the BBMD. The IUT is registered as a Foreign Device with the BBMD. The originating device (D2) is selected having different UDP port number (P2) than the IUT and BBMD. The behavior of the IUT is verified when it correctly responds to the Forwarded-NPDU message from the device having different UDP number.

Configuration Requirements: TD is acting as a BBMD with port P1. D2 is a device at an address using a different port P2.

Test Steps:

1. TRANSMIT Forwarded-NPDU,
 Originating-Device = D2 -- (with UDP port P2)
 NPDU = Who-Is
2. IF (the IUT responds with Unicast I-Am) THEN
 RECEIVE Original-Unicast-NPDU,
 DESTINATION = D2, -- (with UDP port P2)
 NPDU = I-Am
 ELSE
 RECEIVE Distribute-Broadcast-to-Network
 DESTINATION = TD, -- (with UDP port P1)
 NPDU = I-Am,

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 DAESTINATION = 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)
43. 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,~~
 - SOURCEA = 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 DAESTINATION = 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 DAESTINATION = 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 DAESTINATION = 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) *THEN*
 - 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)*
32. WAIT (-Vendor specified period for BDT to be saved in non-volatile memory)
43. MAKE (the IUT reset)
54. TRANSMIT
 - DA = IUT,
 - SA = D1,
 - Read-Broadcast-Distribution-Table
65. RECEIVE
 - DA = D1,
 - SA = IUT,
 - Read-Broadcast-Distribution-Table-Ack,
 - List of BDT Entries
76. 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 DA=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,~~ SOURCEA = 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 DAESTINATION = 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.

14.8 Foreign Device Tests

14.8.1 Registering as a Foreign Device

Reason for Change: Renumbered to allow multiple foreign device tests.

Dependencies: None

BACnet Reference Clause: J.5.2

Purpose: This test case verifies that the IUT can register as a foreign device with a BBMD.

Test Concept: The IUT is caused to register as a foreign device with the TD.

Configuration Requirements: The IUT is configured to register as a foreign device with the TD.

Test Steps:

1. RECEIVE DESTINATION = TD, SOURCE = IUT,
 Register-Foreign-Device
2. TRANSMIT DESTINATION = IUT, SOURCE = TD,
 BVLC-Result,
 'Result Code' = Successful completion

14.8.X1 Register-Foreign-Device Enable and Disable Test

Reason For Change: This tests that the behavior in test 14.8 can be configured by the product end-user.

Purpose: Verify that the option to issue Register-Foreign-Device requests can be configured by the product end-user.

Test Concept: Using a product end-user interface, configure the mode for use of Register-Foreign-Device requests, and then configure the mode to cease use of Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is here configured shall be part of the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
2. RECEIVE DA = BBMD1,
 Register-Foreign-Device

3. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion
4. MAKE (the IUT not in mode for use of Register-Foreign-Device requests)
5. WAIT (more than 31 seconds longer than the 'Time-to-Live' parameter used in Register-Foreign-Device requests)
6. CHECK (that the IUT did not send any Register-Foreign-Device requests)

14.8.X2 Recurring Register-Foreign-Device Test

Reason For Change: This tests in continuous manner what 14.9.1 tests just once.

Purpose: Verify that mode for use of Register-Foreign-Device and setting of 'BBMD Address' parameter are persistent across reset, and that the issuance of Register-Foreign-Device precedes the first issuance of any broadcast, when in that mode.

Test Concept: IUT is put in a mode to use Register-Foreign-Device requests, and it is observed that Register-Foreign-Device requests are sent sufficiently frequently to prevent expiration of the registration at the BBMD.

Configuration Requirements: The product's setting of 'BBMD Address' parameter is configured as BBMD1. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
2. RECEIVE DA = BBMD1,
 Register-Foreign-Device
3. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion
4. BEFORE (the time configured for the 'Time-to-Live' parameter used for Register-Foreign-Device requests)
 RECEIVE DA = BBMD1,
 Register-Foreign-Device
5. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion
6. BEFORE (the time configured for the 'Time-to-Live' parameter used for Register-Foreign-Device requests)
 RECEIVE DA = BBMD1,
 Register-Foreign-Device
7. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion

Notes to Tester: There is no need for the recurring request to be sent any more quickly than precisely the 'Time-to-Live' since the standard mandates that the BBMD preserve the registration for 30 seconds past the 'Time-to-Live'.

14.8.X3 BBMD Address Configuration Test

Reason For Change: This tests that the behavior in test 14.8 can be configured by the product end-user.

Purpose: Verify that the parameter in Register-Foreign-Device in test 14.8 can be configured by the product end-user.

Test Concept: Using a product end-user interface, configure the 'BBMD Address' parameter that is used in Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is configured for a 'BBMD Address' can be anything in the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (through the product's end-user interface, the setting of 'BBMD Address' parameter equal BBMD1)
2. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
3. RECEIVE DA = BBMD1,
 Register-Foreign-Device
4. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion

14.8.X4 Transmits a Broadcast at Startup preceded by Register-Foreign-Device

Reason For Change: This tests in the specific case of startup, what test 14.9.1 expects to observe during ordinary ongoing operation.

Purpose: Verify that mode for use of Register-Foreign-Device and setting of 'BBMD Address' parameter are persistent across reset, and that the issuance of Register-Foreign-Device precedes the first issuance of any broadcast, when in that mode.

Test Concept: IUT is put in a mode to use Register-Foreign-Device requests, persistently so it will be re-established, then IUT is reset, and the timing of Register-Foreign-Device request to re-establish that precedes the first issuance of any broadcast.

Configuration Requirements: The product's setting of 'BBMD Address' parameter is configured as BBMD1. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (IUT enter mode for use of Register-Foreign-Device requests, persistently
so it will be re-established after any reset)
2. MAKE (IUT reset)
3. RECEIVE DA = BBMD1,
Register-Foreign-Device
4. TRANSMIT BVLC-Result,
'Result Code' = Successful completion
5. RECEIVE DA = BBMD1,
Distribute-Broadcast-To-Network,
NPDU = (any broadcast)
6. TRANSMIT BVLC-Result,
'Result Code' = Successful completion

Notes to Tester: For the I-Am, one can precede the Register-Foreign-Device command, as long as then after the Register-Foreign-Device occurs, it is followed by a Distribute-Broadcast-To-Network again, of that I-Am.

14.8.X5 Time-to-Live Configuration Test

Reason For Change: Adds verification that the behavior in test 14.8 can be configured by the product end-user.

Purpose: Verify that the parameter in Register-Foreign-Device in test 14.8 can be configured by the product end-user, through a reasonable range (120 through 28800 is sufficient; the absolute upper limit is 65535 seconds, approximately 17 hours).

Test Concept: Using a product end-user interface, configure the 'Time-to-Live' parameter that is used in Register-Foreign-Device requests.

Configuration Requirements: The means by which the product is configured can be anything in the product's end-user interface. BBMD1 is the TD simulating a correctly functioning BBMD implementation.

Test Steps:

1. MAKE (through the product's end-user interface, the setting of 'Time-to-Live' parameter equal 120,
or any larger value supported by the implementation)
2. MAKE (IUT enter mode for use of Register-Foreign-Device requests)
3. RECEIVE DA = BBMD1,
Register-Foreign-Device,
'Time-to-Live' = (value configured in step 1)
4. TRANSMIT BVLC-Result,
'Result Code' = Successful completion

14.9.1 Distribute-Broadcast-To-Network

Reason for Change: Modified test to enforce the behavior specified in clause J.2.12.

Dependencies: 14.8, "Registering as a Foreign Device"

BACnet Reference Clause: J.2.10

Purpose: This test case verifies that the IUT, registered as a foreign device, can issue a request to a BBMD to broadcast the message on all subnets in the BBMD's BDT.

Test Concept: The IUT is configured to register itself as a foreign device with the TD, then after registration is achieved it is caused to initiate a broadcast message to be conveyed to the BBMD for distribution. If the IUT does not support foreign device registration, or cannot initiate broadcast messages conveying a BACnet NPDU, then this test shall be omitted.

Test Steps:

1. RECEIVE DESTINATION = TD, SOURCE = IUT,
Register-Foreign-Device
2. TRANSMIT DESTINATION = IUT, SOURCE = TD,
BVLC-Result,
'Result Code' = Successful completion
3. MAKE (*a condition that would make the IUT normally initiate a broadcast*)
4. RECEIVE DESTINATION = TD, SOURCE = IUT,
Distribute-Broadcast-To-Network
5. CHECK (*that the IUT does not transmit an Original-Broadcast-NPDU on this port*)

BACnet Testing Laboratories - Specified Tests

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	<ul style="list-style-type: none"> 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	26-Oct-2004	Roland Laird	<ul style="list-style-type: none"> Modified all Clause 14 tests
0.10		Roland Laird	<ul style="list-style-type: none"> Continuation of BACnet/IP modifications - changes highlighted inline
0.11	27-Oct-2004	Carl Neilson	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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 $2 < x \leq 254$ 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	<ul style="list-style-type: none"> Added WhoHas tests 9.32.1.X1, 9.32.1.X2
0.17	05-Oct-2005	Jim Butler Carl Neilson	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Added ARCNET tests & re-arranged section 2.

BACnet Testing Laboratories - Specified Tests

0.19	27-Oct-2005	Carl Neilson	<ul style="list-style-type: none"> Added 7.3.1.X3 Array Sizing Test Added 13.X2.1 APDU Retry and Timeout Removed router qualification tests. Added reason for change to 13.X2.1 & modified note to tester
0.20	17-Jan-2006	Carl Neilson	<ul style="list-style-type: none"> Fixed incorrect numbering of BACnet/IP sections Deleted old comment as end of 7.3.2.20.3.X1 Added 8.8.1 & 8.8.2 that include transmission of final BACnet-ComplexACK-PDUs
4.0.0	13-Sep-2006	Carl Neilson	<ul style="list-style-type: none"> Changed revision numbering
4.0.1	04-Apr-2007	Carl Neilson	<ul style="list-style-type: none"> Round 4 changes (excl SCHED)
4.0.2	02-May-2007	Carl Neilson	<ul style="list-style-type: none"> Added 9.10.1.X2
4.0.3	11-Jun-2007	Lori Tribble	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Removed some of the highlighting. Updated tests per mtg 10/15/2007
4.0.7	18-Dec-2007	Lori Tribble	<ul style="list-style-type: none"> Added Virtual Routing tests Added List Manipulation Tests
4.0.8	22-Feb-2008	Lori Tribble	<ul style="list-style-type: none"> Fixed BTL-7.3.1.11 per TGTC-18.
4.0.9	01-Apr-2008	Lori Tribble	<ul style="list-style-type: none"> Updated page header format.
4.0.10	16-Apr-2008	Lori Tribble	<ul style="list-style-type: none"> 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

BACnet Testing Laboratories - Specified Tests

				<p>TGTC-41 - Modified 7.3.2.23.6.1</p> <p>TGTC-43 - Added 8.22.1, 8.22.2. Modified 8.22.X2.</p> <p>BTL-CRR-0018 - Modified 9.10.2.1</p> <p>BTL-CRR-0050 - Modified 7.2.1.10</p> <p>BTL-CRR-0051 - Modified 13.X1.3, 13.X1.6, 13.X1.7</p> <ul style="list-style-type: none"> • 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 2008	16,	Lori Tribble	<ul style="list-style-type: none"> • Accepted Changes made above • Updated table of contents • Added Reason for Change to tests that did not have it.
4.0.13	May 2008	21,	Lori Tribble	<ul style="list-style-type: none"> • Marked test 9.10.2.1 for further review • 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 2008	20,	Lori Tribble	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Accepted all changes made previously. • Made format changes.
4.0.18	Oct 21, 2008		Lori Tribble	<ul style="list-style-type: none"> • Updated Reason For Change for all tests that now have SSPC proposals. • Added client side schedule tests • Removed test 9.23.1.X

BACnet Testing Laboratories - Specified Tests

5.0.1	Oct 21, 2008	Lori Tribble	<ul style="list-style-type: none"> • Changed COV test from 24 hour lifetime to 8 hour lifetime. • 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	<ul style="list-style-type: none"> • • 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	<ul style="list-style-type: none"> • 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

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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_i not D₁ • 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	<ul style="list-style-type: none"> • Accepted all changes per acceptance by BTL-WG 6/18/2008.
6.0.1	26-Jan-2011	Duffy O'Craven	<ul style="list-style-type: none"> • 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

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> 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 9.24.2.1, 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 tests 7.3.2.23.8 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.

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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.

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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.

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 <i>and an instance number that is creatable</i>". • 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.

BACnet Testing Laboratories - Specified Tests

			<ul style="list-style-type: none"> Removed entire Chapter 14, replicated in 135-2009e-1
9.0.final	01-Dec-2011	Duffy O'Craven	<ul style="list-style-type: none"> Updated from 9.0.15 to 9.0.final, accepting all change tracking
12.0.1	25-Jul-2012	Lori Tribble	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Accepted all changes and Changed Name
12.1.1	27-Sept-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0b
12.1.2	27-Sept-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0c
12.1.3	30-Sept-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0d
12.1.4	30-Sept-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0e
12.1.5	1-Oct-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0f
12.1.6	1-Oct-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.0g
12.1.7	1-Oct-2013	Lori Tribble	<ul style="list-style-type: none"> Applied Errata 9/30/2013
14.0.a	1-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.1a
14.0.b	1-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.1b
14.0.c	1-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.1c
14.0.d	1-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.1d
14.0.e	1-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> Applied Addendum 12.1e
14.0.plus_errata	3-Nov-2014	Lori Tribble	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Removed comments, and pdated to 14.0.final without change.
15.0.05	24-Aug-2017	Lori Tribble	<ul style="list-style-type: none"> Applied Addenda 14.0b-j plus errata
15.0.08	25-Sep-2017	Lori Tribble	<ul style="list-style-type: none"> Removed test 8.4.X9.
15.0.11	11-Oct-2017	Lori Tribble	<ul style="list-style-type: none"> Applied errata from voting members
15.0.final	11-Oct-2017	Lori Tribble	<ul style="list-style-type: none"> Accepted all changes
15.1.1	30-Mar-2018	Lori Tribble	<ul style="list-style-type: none"> Applied addenda a, b , c, d, and errata
15.1.2	6-Apr-2018	Lori Tribble	<ul style="list-style-type: none"> Accepted all changes
15.1.3	26-Apr-2018	Lori Tribble	<ul style="list-style-type: none"> Reformatted almost all tests to meet 135.1 formats, applied errata
15.1.4	1-May-2018	Lori Tribble	<ul style="list-style-type: none"> Applied Errata
15.1.5	3-May-2018	Lori Tribble	<ul style="list-style-type: none"> Fixed formatting and numbering issues.
15.1.final	1-June-2018	Lori Tribble	<ul style="list-style-type: none"> Renamed to final
15.2.1		Lori Tribble	<ul style="list-style-type: none"> Applied addenda e
15.2.2		Lori Tribble	<ul style="list-style-type: none"> Applied addenda f
15.2.3		Lori Tribble	<ul style="list-style-type: none"> Applied addenda g
15.2.4	11-Nov-2018	Lori Tribble	<ul style="list-style-type: none"> Removed some highlights
15.2.final	11-Nov-2018	Lori Tribble	<ul style="list-style-type: none"> Accepted all changes and updated revision
15.2.5	13-Dec-2018	Lori Tribble	<ul style="list-style-type: none"> Reverted back to 15.2.4 due to formatting issues. Accepted all changes again and updated revision and date.

BACnet Testing Laboratories - Specified Tests

15.2.final2	14-Dec-2018	Lori Tribble	• Updated date and revision.
16.0.1	19-Aug-2019	Lori Tribble	• Converted to docx. Updated TOC.
16.0.2	19-Aug-2019	Lori Tribble	• Applied Addenda h.
16.0.3	26-Aug-2019	Lori Tribble	• Applied Addenda i
16.0.4	27-Sug-2019	Lori Tribble	• Applied Addenda j
16.0.5	28-Aug-2019	Lori Tribble	• Applied Addenda k
16.0.6	29-Aug-2019	Lori Tribble	• Applied Addenda l
16.0.7	29-Aug-2019	Lori Tribble	• Applied Addenda m
16.0.8	29-Aug-2019	Lori Tribble	• Applied Addenda n
16.0.9	30-Aug-2019	Lori Tribble	• Applied Addenda o
16.0.10	30-Aug-2019	Lori Tribble	• Applied Addenda p
16.0.11	30-Aug-2019	Lori Tribble	• Applied Addenda r
16.0.12	30-Aug-2019	Lori Tribble	• Applied Addenda s
16.0.13	31-Aug-2019	Lori Tribble	• Applied Errata
16.0.14	19-Sep-2019	Lori Tribble	• Applied all changes. Formatting changes.
16.0.15	25-Sep-2019	Lori Tribble	• Applied Errata.
16.0.final	25-Sep-2019	Lori Tribble	• Renamed to Final
16.0.final.v2	11-Nov-2019	Emily Hayes	• Renamed to Final.v2
16.1	10-Dec-2019	Lori Tribble	• Applied Errata, added PR21 and PR22 items, renamed to 16.1
16.1.final	10-Jan-2020	Emily Hayes	• Accepted update. Renamed to Final