ANSI/ASHRAE Addendum a to
ANSI/ASHRAE Standard 135.1-2003

ASHRAE STANDARD

Method of Test for Conformance to BACnet®

Approved by the ASHRAE Standards Committee on January 21, 2006; by the ASHRAE Board of Directors on January 26, 2006; and by the American National Standards Institute on January 27, 2006.

This standard is under continuous maintenance by a Standing Standard Project Committee (SSPC) for which the Standards Committee has established a documented program for regular publication of addenda or revisions, including procedures for timely, documented, consensus action on requests for change to any part of the standard. The change submittal form, instructions, and deadlines may be obtained in electronic form from the ASHRAE Web site, http://www.ashrae.org, or in paper form from the Manager of Standards. The latest edition of an ASHRAE Standard may be purchased from ASHRAE Customer Service, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. E-mail: orders@ashrae.org. Fax: 404-321-5478. Telephone: 404-636-8400 (worldwide), or toll free 1-800-527-4723 (for orders in US and Canada).

When addenda, interpretations, or errata to this standard have been approved, they can be downloaded free of charge from the ASHRAE Web site at http://www.ashrae.org.

ISSN 1041-2336

Approved American National Standard
ANSI

American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.
1791 Tullie Circle NE, Atlanta, GA 30329
www.ashrae.org

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus standard developed under the auspices of the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). Consensus is defined by the American National Standards Institute (ANSI), of which ASHRAE is a member and which has approved this standard as an ANS, as "substantial agreement reached by directly and materially affected interest categories. This signifies the concurrence of more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that an effort be made toward their resolution." Compliance with this standard is voluntary until and unless a legal jurisdiction makes compliance mandatory through legislation.

ASHRAE obtains consensus through participation of its national and international members, associated societies, and public review.

ASHRAE Standards are prepared by a Project Committee appointed specifically for the purpose of writing the Standard. The Project Committee Chair and Vice-Chair must be members of ASHRAE; while other committee members may or may not be ASHRAE members, all must be technically qualified in the subject area of the Standard. Every effort is made to balance the concerned interests on all Project Committees.

The Manager of Standards of ASHRAE should be contacted for:

a. interpretation of the contents of this Standard,
b. participation in the next review of the Standard,
c. offering constructive criticism for improving the Standard,
d. permission to reprint portions of the Standard.

DISCLAIMER

ASHRAE uses its best efforts to promulgate Standards and Guidelines for the benefit of the public in light of available information and accepted industry practices. However, ASHRAE does not guarantee, certify, or assure the safety or performance of any products, components, or systems tested, installed, or operated in accordance with ASHRAE's Standards or Guidelines or that any tests conducted under its Standards or Guidelines will be nonhazardous or free from risk.

ASHRAE INDUSTRIAL ADVERTISING POLICY ON STANDARDS

ASHRAE Standards and Guidelines are established to assist industry and the public by offering a uniform method of testing for rating purposes, by suggesting safe practices in designing and installing equipment, by providing proper definitions of this equipment, and by providing other information that may serve to guide the industry. The creation of ASHRAE Standards and Guidelines is determined by the need for them, and conformance to them is completely voluntary.

In referring to this Standard or Guideline and in marking of equipment and in advertising, no claim shall be made, either stated or implied, that the product has been approved by ASHRAE.

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

Foreword

The purpose of this addendum is to make a number of independent substantive changes to the 2003 edition of ANSI/ASHRAE Standard 135.1. These modifications are the result of changes that have occurred in the parent Standard, ANSI/ASHRAE Standard 135, *BACnet--A Data Communication Protocol for Building Automation and Control Networks*. The changes made in this addendum are summarized below.

In the following document, language to be added to existing clauses of ANSI/ASHRAE Standard 135.1-2003 is indicated through the use of *italics* while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are to be added, plain type is used throughout. Unless otherwise specified, "BACnet Reference Clauses" refers to clauses in ANSI/ASHRAE Standard 135-2004.

135.1*a*-1. Add Partial Day Scheduling to the Schedule object.

Rationale
This change is a modification of tests related to the Schedule object to account for the addition of partial day scheduling in Addendum 135*a*-1 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-1

[Change clause **7.3.2.22.4**, p. 81]

7.3.2.22.4 Weekly_Schedule and Exception_Schedule Interaction Test

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

BACnet Reference Clauses: ~~12.22.7, 12.22.8~~*12.24, 12.24.4, 12.24.7, 12.24.8*.

Purpose: To verify that an Exception_Schedule takes precedent over a coincident BACnetDailySchedule.

Test Concept: The IUT is configured with a Weekly_Schedule and an Exception_Schedule that apply to the same time. The local date and time are changed to the time when the Exception-Schedule is supposed to take control and the Present_Value is read to verify that the scheduled write operation occurs. The local date and time are changed again to a value that would cause another change if the Weekly_Schedule were in control. The Present_Value is read to verify the Exception_Schedule is still controlling.

Configuration Requirements: The IUT shall be configured with a Schedule object containing a Weekly_Schedule and an Exception_Schedule that apply to the same dates. The BACnetSpecialEvents in the Exception_Schedule shall have a higher EventPriority than any other coincident BACnetSpecialEvent. The BACnetTimeValue pairs shall be assigned values such that the values written by the Weekly_Schedule are distinguishable from the values written by the Exception_Schedule. Let D_1 represent the date and time when the Exception_Schedule is configured to take control and write value V_1. There shall be at least one BACnetTimeValue pair in the Weekly_Schedule that specifies a time, D_2, that is after D_1 but before the Exception_Schedule expires. The Weekly_Schedule is configured to write value V_2 at time D_2.

For BACnet implementations with a Protocol_Revision of 4 or higher, the date D_2 shall be chosen to occur between D_1 and any entry in the Exception schedule that schedules a NULL value.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D_1) | MAKE (the local date and time = D_1)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present_Value = V_1
4. (TRANSMIT TimeSynchronization-Request, 'Time' = D_2) | MAKE (the local date and time = D_2)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present_Value = V_1

[Change clause **7.3.2.22.5**, p. 81]

7.3.2.22.5 Exception_Schedule Restoration Test

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

BACnet Reference Clauses: ~~12.22.7, 12.22.8~~*12.24.4, 12.24.8, 12.24.9*.

Purpose: To verify the restoration behavior in an Exception_Schedule.

Test Concept: The IUT is configured with a Schedule object containing an Exception_Schedule with BACnetTimeValue entries that do not include the time 00:00. The local date and time are changed to a value between 00:00 and the first entry in the Exception_Schedule. Present_Value is read to verify that ~~the write operation from the last entry in the day occurs~~*it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4*. The IUT is reset and the Present_Value is checked again to verify that ~~the write operation from the last entry in the day occurs~~*it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4.*

Configuration Requirements: The IUT shall be configured with a Schedule object that contains an Exception_Schedule that has more than one scheduled write operation for a particular day *and the first scheduled write is scheduled to occur before the first entry in the corresponding Weekly_Schedule entry*. None of the write operations shall be scheduled for time 00:00 and there shall be no higher priority coincident BACnetSpecialEvents. In the test description D_1 represents a time between 00:00 on the day the Exception_Schedule is active and the time of the first schedule write operation in the BACnetSpecialEvent. V_{last} represents the value that is scheduled to be written in the last BACnetTimeValue pair for the day.

Test Steps:

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

[Change clause **7.3.2.22.6**, p. 82]

7.3.2.22.6 Weekly_Schedule Restoration Test

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

BACnet Reference Clause: ~~12.22.7~~*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 ~~the write operation from the last entry in the day occurs~~*it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4*. The IUT is reset and the Present_Value is checked again to verify that ~~the write operation from the last entry in the day occurs~~*it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4.*

Configuration Requirements: The IUT shall be configured with a Schedule object that contains a Weekly_Schedule that has more than one scheduled write operation for a particular day. None of the write operations shall be scheduled for time 00:00 and there shall be no higher priority coincident BACnetSpecialEvents. In the test description D_1 represents a time between 00:00 and the time of the first scheduled write operation in the BACnetDailySchedule. V_{last} represents the value that is scheduled to be written in the last BACnetTimeValue pair for the day.

Test Steps:

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

135.1*a*-2. Enable reporting of proprietary events by the Event Enrollment object.

Rationale
This change is a modification of tests related to the Event Enrollment object to account for changes made to enable the reporting of proprietary events in Addendum 135*a*-2 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-2

[Add new clause **8.4.9**, p. 138]

8.4.9 EXTENDED Tests

Dependencies: None

BACnet Reference Clauses: 13.2 and 13.3

Purpose: This test verifies the correct generation of extended ConfirmedEventNotification messages. It applies to event generating objects with an Event_Type of EXTENDED. This applies to any Event Enrollment object that uses a proprietary algorithm.

Test Concept: The event generating object begins the test in a NORMAL state. The event generating object is made to transition to any state by any means necessary. The resulting ConfirmedEventNotification message is received and verified.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for whichever transition shall be used for the test. 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.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (the event generating object transition)
3. WAIT (Time_Delay)
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 being tested),
'Time Stamp' =	(the current local time),
'Notification Class' =	(the configured notification class, or absent if the event generating object is using a Recipient property instead),
'Priority' =	(the value configured for this transition),
'Event Type' =	EXTENDED,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	(the state the object was made to transition to),
' Event Values' =	VendorId, extendedEventType, and any other values as defined by the vendor

Passing Result: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

135.1*a*-3. Allow detailed error reporting when all ReadPropertyMultiple accesses fail.

Rationale
This change is a modification of tests related to execution of the ReadPropertyMultiple service to account for changes made in the way errors are reported in Addendum 135*a*-3 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-3

[Change clause **9.20.2.2**, p. 245]

9.20.2.2 Reading Multiple Properties with Access Errors for Every Property

Purpose: This test case verifies the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for only unsupported properties.

Test Concept: The selections for objects and properties for this test shall consist of objects that are not supported, properties that are not supported for the selected objects, or a combination of the two such that there are no object, property combinations that represent a supported property.

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

~~2. RECEIVE BACnet-Error-PDU,~~
~~'Error Class' = OBJECT | PROPERTY,~~
~~'Error Code' = (any valid error code for the returned error class)~~

2. RECEIVE
(BACnet-Error-PDU,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class)) |
(ReadPropertyMultiple-ACK,
'Object Identifier' = Object1,
'Property Identifier' = P1,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class),
'Property Identifier' = P2,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class),
'Property Identifier' = P3,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class),
'Object Identifier' = Object2,
'Property Identifier' = P4,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class),
'Property Identifier' = P5,
'Error Class' = OBJECT | PROPERTY,
'Error Code' = (any valid error code for the returned error class),

'Property Identifier' = *P6,*
'Error Class' = *OBJECT | PROPERTY,*
'Error Code' = *(any valid error code for the returned error class))*

135.1*a*-4. Remove the Recipient property from the Event Enrollment object.

Rationale
This change is a modification of tests related to the Event Enrollment object to account for the fact that the Recipient property and other related properties were removed in Addendum 135*a*-4 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-4

[Change clause **4.5.10.11**, p. 14]

4.5.10.11 Event Enrollment

{
object-identifier: (event-enrollment, ❑)
object-name: "❑"
object-type: event-enrollment
description: "❑"
event-type: ❑
notify-type: ❑
event-parameters: {❑, ❑...}
object-property-reference: (❑)
event-state: ❑
event-enable: (❑,❑,❑)
acked-transitions: (❑,❑,❑)
notification-class: ❑
~~recipient: ❑~~
~~process-identifier: ❑~~
~~priority: ❑~~
~~issue-confirmed-notifications: ❑~~
-- The following four properties were removed from Event Enrollment objects in protocol revision 4:
-- recipient: ❑
-- process-identifier: ❑
-- priority: ❑
-- issue-confirmed-notifications: ❑
event-time-stamps: {❑,❑,❑}
profile-name: "❑"
}

[Change clause **7.3.1.10**, p. 36]

7.3.1.10 Event_Enable Tests

…

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. 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~~ *The* value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

…

[Change clause **7.3.1.11**, p. 38]

7.3.1.11 Acked_Transitions Tests

…

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. ~~If a Notification Class object is being used to configure recipient information the~~*The* value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

…

[Change clause **7.3.1.12**, p. 42]

7.3.1.12 Notify_Type Test

…

Configuration Requirements: The IUT shall be configured with two event-generation objects, E_1 and E_2. Object E1 shall be configured with a Notify_Type of ALARM and E2 shall be configured with a Notify_Type of EVENT. Both objects shall be in a NORMAL Event_State at the beginning of the test. 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). ~~If a Notification Class object is being used to configure recipient information the~~*The* value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE).

…

[Change clause **9.7.2.5**, p. 197]

9.7.2.5 Notification Class Filter

…

Configuration Requirements: If possible, the IUT shall be configured with one or more event-generating objects using each of two notification classes. ~~If Event Enrollment objects are used to establish this configuration the Recipient property shall have a value of NULL.~~

…

135.1*a*-5 – MS/TP slave proxy tests

Rationale
This change is a modification of various tests to account for the creation of a wildcard instance of the Device Object_Identifier and the ability to issue I-am responses on behalf of MS/TP slaves. These changes correspond to changes made in 135*a*-5 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-5

[Add new clause **9.16.2.6**, p.237]

9.16.2.6 Attempting to Create an Object with an instance of 4194303
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 Identifier' = (any object identifier representing a creatable object-type with an instance of 4194303)
2. RECEIVE BACnet-Reject-PDU,
   'Reject Reason' = PARAMETER_OUT_OF_RANGE

[Add new clause **9.18.1.3**, p.237]

9.18.1.3 Reading a Property From the Device Object using the Unknown Instance
Purpose: This test case verifies that the IUT can execute ReadProperty service requests when the requested object-identifier references a Device Object with an unknown instance (4194303). Let X be the instance number of the Device Object for the IUT. 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 ReadProperty-Request,
   'Object Identifier' = (Device, 4194303),
   'Property Identifier' = Object-Identifier
2. RECEIVE ReadProperty-ACK,
   'Object Identifier' = (Device, X),
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X)

Passing Result: The IUT shall respond as indicated conveying the value specified in the EPICS.

[Add new clause **9.20.1.11**, p.245]

9.20.1.11 Reading a Property From the Device Object using the Unknown Instance
Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests when the requested object-identifier references a Device Object with an unknown instance (4194303). Let X be the instance number of the Device Object for the IUT. 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 ReadPropertyMultiple-Request,
   'Object Identifier' = (Device, 4194303),
   'Property Identifier' = Object-Identifier

2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = (Device, X),
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X)

Passing Result: The IUT shall respond as indicated conveying the value specified in the EPICS.

[Change clause **4.5.9**, p. 8**]**

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: ❑↵
}↵

[Add new clause **13.5**, p. 427**]**

13.5 Slave Proxy Tests

These tests verify that an IUT can perform the function of a slave proxy. In order to be a slave proxy, a device must be capable of finding and confirming MS/TP slaves.

BACnet Reference Clauses: 12.11.39, 12.11.40, 12.11.41, 12.11.42, and 16.10.2.

13.5.1 Manual Slave Binding Test

Purpose: This test verifies that the IUT can find and confirm MS/TP slave devices listed in the Manual_Slave_Binding property in the IUT's device object. This test also verifies that the IUT correctly distinguishes between slave and master devices, and that it performs periodic confirmation of slave devices.

Test Concept: Configure the Manual_Slave_Binding property with the address of two MS/TP devices. Attach a slave at one of the addresses and a master that supports the Who-Is and I-Am services at the other address. Monitor the network to verify that the IUT confirms the devices and then verify that the slave device address is added into the Slave_Address_Binding property and that the master address is not. The slave is then removed, and once the IUT re-confirms the slave, it is verified that the slave is removed from the Slave_Address_Binding property.

Configuration Requirements: The MS/TP network shall contain a slave device at address A1 with a device identifier of D1 and a master device at address A2 with a device identifier of D2. The slave device shall not support the reading of its device object using the wildcard instance of 4194303. The master device shall execute the Who-Is service and initiate the I-Am service. The IUT shall be configured to perform slave proxying.

Test Steps:
1. BEFORE **Slave Proxy Confirm Interval**
   REPEAT addr=(A1, A2) DO {

RECEIVE DESTINATION=addr, SRC=IUT
ReadProperty-Request,
'Object Identifier' = (the correct value for the address being queried),
'Property Identifier' = Protocol_Services_Supported
RECEIVE DESTINATION=IUT, SRC=addr
BACnet-Complex-ACK,
'Object-Identifier' = (the correct value for the address being queried),
'Property Identifier' = Protocol_Services_Supported,
'Property Value' = (any valid value for this property)
}
2. VERIFY Slave_Address_Binding = ((A1, D1))
3. Remove the slave device from the MS/TP network
4. BEFORE **Slave Proxy Confirm Interval**
RECEIVE DESTINATION=A1, SRC=IUT
ReadProperty-Request,
'Object Identifier' = (DEVICE,the correct value for the address being queried),
'Property Identifier' = Protocol_Services_Supported
-- note that the slave will not reply to this request as it is no longer connected to the network.
5. WAIT (longer than it takes for the IUT to timeout on this request)
6. VERIFY Slave_Address_Binding = ()

Passing Result: The IUT shall read the Protocol_Services_Supported property from each MS/TP device to determine whether or not it supports Who-Is. The implementer may have chosen to read this property after the IUT has determined that a device exists at a given MAC address. In this case, any property shall be accepted in step 1, and the TD shall expect a subsequent read from the IUT for the Protocol_Services_Supported property for the two devices attached to the MS/TP segment. The IUT is also allowed to generate any other traffic it cares to during the test including, but not limited to, reading property values from the devices it finds.

13.5.2 Automatic Slave Discovery Test

Purpose: This test verifies that an IUT that contains an Auto_Slave_Discovery property is able to find and confirm MS/TP slaves that support reading with the wildcard instance of 4194303. This test also verifies that the IUT correctly distinguishes between slave and master devices, and that it performs periodic confirmation of slave devices.

Test Concept: Configure the Auto_Slave_Binding property to enable automatic detection of slaves. Connect a slave and a master to the MS/TP network. Monitor the network to verify that the IUT searches for and finds each device connected to the MS/TP segment. Verify that the IUT added the slave to its Slave_Address_Binding property and that it did not add the master to the list. A slave is then removed, and once the IUT re-confirms the slave, it is verified that the slave is removed from the Slave_Address_Binding property.

Configuration Requirements: The MS/TP network shall contain a slave device at address A1 with a device identifier of D1 and a master device at address A2 with a device identifier of D2. The slave device shall support the reading of its device object using the wildcard instance of 4194303. The master device shall execute the Who-Is service and initiate the I-Am service. The IUT shall be configured to perform automatic slave detection on all MS/TP addresses except its own and the broadcast address.

BACnet Reference Clauses: 12.11.39, 12.11.41, 12.11.42, and 15.5.2.

Test Steps:

1. MAKE(the IUT start its automatic slave detection)
2. BEFORE **Slave Proxy Confirm Interval**
REPEAT addr=(all MS/TP addresses excluding the IUT's MAC address) DO {
RECEIVE DESTINATION=addr, SRC=IUT
ReadProperty-Request,
'Object Identifier' = (DEVICE,4194303),

'Property Identifier' = Protocol_Services_Supported
RECEIVE DESTINATION=IUT, SRC=addr
BACnet-Complex-ACK,
'Object-Identifier' = (the correct value for the address being queried),
'Property Identifier' = Protocol_Services_Supported,
'Property Value' = (any valid value for this property)
}
3. VERIFY Slave_Address_Binding = ((A1, D1))
4. Remove the slave device from the MS/TP network
5. BEFORE **Slave Proxy Confirm Interval**
RECEIVE DESTINATION=A1, SRC=IUT
ReadProperty-Request,
'Object Identifier' = (DEVICE, the correct value for the address being queried),
'Property Identifier' = Protocol_Services_Supported
-- note that the slave will not reply to this request as it is no longer connected to the network.
6. WAIT (longer than it takes the IUT to timeout on this request)
7. VERIFY Slave_Address_Binding = ()

Passing Result: The IUT shall read the Protocol_Services_Supported property from each MS/TP device to determine whether or not it supports Who-Is. The implementer may have chosen to read this property after the IUT has determined that a device exists in a given slot. In this case, any property shall be accepted in step 1, and the TD shall expect a subsequent read from the IUT for the Protocol_Services_Supported property for the two devices attached to the MS/TP segment. The IUT is also allowed to generate any other traffic it cares to during the test including, but not limited to, reading property values from the devices it finds.

13.5.3 Proxy Test

Purpose: This test verifies that an IUT correctly proxies for slaves that are listed in its Slave_Address_Binding property.

Test Concept: Configure the IUT so that it will proxy for a slave device and wait for the IUT to find and confirm the slave. Issue Who-Is requests in all forms to ensure that the IUT correctly proxies the I-Am responses for the slave device. The test should be repeated with the TD connected to the MS/TP segment, and with the TD connected to a different BACnet network.

Configuration Requirements: The MS/TP network shall contain a slave device at address A1 with a device identifier of D1. The IUT shall be configured to perform slave proxying. The test starts after the IUT has successfully found and confirmed the slave device. This test shall be repeated once with the TD connected to the MS/TP network and once with the TD connected to a different BACnet network.

BACnet Reference Clauses: 12.11.39, 12.11.40, 12.11.41, 12.11.42, and 16.10.2.

Test Steps:

1. TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST
SOURCE = A1
I-Am-Request,
'I Am Device Identifier' = (the slave's Device object's Object_Identifer),
'Max APDU Length Accepted' = (the slave's value for this property),
'Segmentation Supported' = FALSE,
'Vendor Identifier' = (the slave's value for this property)
4. TRANSMIT DESTINATION = A1, Who-Is
5. WAIT **Internal Processing Fail Time**
6. RECEIVE

DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST
SOURCE = A1
I-Am-Request,
'I Am Device Identifier' = (the slave's Device object's Object_Identifer),
'Max APDU Length Accepted' = (the slave's value for this property),
'Segmentation Supported' = FALSE,
'Vendor Identifier' = (the slave's value for this property)

7. TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is
8. WAIT **Internal Processing Fail Time**
9. RECEIVE
   DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST
   SOURCE = A1
   I-Am-Request,
   'I Am Device Identifier' = (the slave's Device object's Object_Identifer),
   'Max APDU Length Accepted' = (the slave's value for this property),
   'Segmentation Supported' = FALSE,
   'Vendor Identifier' = (the slave's value for this property)

135.1*a*-6. Add a new silenced mode to the DeviceCommunicationControl service.

Rationale
This change is a modification of tests related to the addition of a new silenced mode to the DeviceCommunicationControl in Addendum 135*a*-6 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-6

[Add new clause **8.24.7**, p. 160]

8.24.7 Time Duration, Disable-Initiation, Password

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)

[Add new clause **9.24.1.6**, p. 258]

9.24.1.6 Indefinite Time Duration, Disable-Initiation, Restored by DeviceCommunicationControl

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when an indefinite time duration is specified, only initiation is disabled, and communication is restored using the DeviceCommunicationControl service. If the IUT does not initiate any services other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE-INITIATION,
   'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-Simple-ACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (the 'Property Identifier' specified in step 7),
   'Property Value' = (any valid value for the property)
9. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = ENABLE,
   'Password' = (any appropriate password as described in the Test Concept)
10. RECEIVE BACnet-Simple-ACK-PDU
11. MAKE (do something to cause the IUT to initiate a message)
12. WAIT **Internal Processing Fail Time**

13. CHECK (Verify that the IUT initiated a message)

[Add new clause **9.24.1.7**, p. 258]

9.24.1.7 Indefinite Time Duration, Disable-Initiation, Restored by ReinitializeDevice

Dependencies: ReinitializeDevice Service Execution Tests, 9.27.

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when an indefinite time duration is specified, only initiation is disabled, and communication is restored using the ReinitializeDevice service. If the IUT does not initiate any services other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE-INITIATION,
   'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-Simple-ACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (the 'Property Identifier' specified in step 7),
   'Property Value' = (any valid value for the property)
9. TRANSMIT ReinitializeDevice-Request,
   Reinitialized State of Device' =WARMSTART,
   'Password' = (any appropriate password as described in the Test Concept)
10. RECEIVE BACnet-Simple-ACK-PDU
11. CHECK (Did the IUT perform a WARMSTART reboot?)
12. MAKE (do something to cause the IUT to initiate a message)
13. WAIT **Internal Processing Fail Time**
14. CHECK (Verify that the IUT initiated a message)

[Add new clause **9.24.1.8**, p. 258]

9.24.1.8 Finite Time Duration, Disable Initiation

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when finite time duration is specified and only initiation is disabled. If the IUT does not initiate any services, other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
   'Time Duration' = (a value T > 1, in minutes, selected by the tester).
   'Enable/Disable' = DISABLE-INITIATION,
   'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-Simple-ACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)

5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (the 'Property Identifier' specified in step 7),
   'Property Value' = (any valid value for the property)
9. WAIT (T)
10. MAKE (do something to cause the IUT to initiate a message)
11. WAIT **Internal Processing Fail Time**
12. CHECK (Verify that the IUT initiated a message)

135.1*a*-7. Addition of tests for Data Sharing BIBBs

Rationale
A review of Standard 135.1 by the BACnet Testing Labs working group revealed that there was insufficient test coverage for the Data Sharing BIBBs. These new tests meet this need.

Addendum 135.1*a*-7

[Add new clause **9.20.1.11**, p. 245]

9.20.1.11 Reading Maximum Multiple Properties

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be read using a single ReadPropertyMultiple request.

Test Concept: The object-identifier is read from the device object as many times as can be conveyed in the largest request accepted by the IUT or as can be returned in the largest response that the IUT can generate. The calculation of the maximum request/response size shall be based on the IUT's Max_APDU_Length_Accepted and maximum segments per request/response.

The procedure to determine the number of object-identifiers to use is:

MaxAPDU = IUT's Max_APDU_Length_Accepted
MaxRxSegs = IUT's maximum segments accepted per request
MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4
SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6
NonSegRespHdrSize = size of (non-segmented BACnet-ComplexACK-PDU header) = 3
SegRespHdrSize = size of (segmented BACnet-ComplexACK-PDU header) = 5
ObjIdSize = size of (an Object-Identifier) = 5
TagsSize = size of (an open and a close tag) = 2
PropIdSize = size of ('Object-Identifier' property Id) = 2

If the IUT does not support receiving segmented requests:
MaxPropsPerRqst =
(MaxAPDU - NonSegRqstHdrSize - ObjIdSize - TagsSize) / PropIdSize =
(MaxAPDU - 11) / 2

If the IUT does support receiving segmented requests:
MaxPropsPerRqst =
((MaxAPDU - SegRqstHdrSize) * MaxRxSegs - ObjIdSize - TagsSize) / PropIdSize =
((MaxAPDU - 6) * MaxRxSegs - 7) / 2

If the IUT does not support sending segmented responses:
MaxPropsPerResp =
(MaxAPDU - NonSegRespHdrSize - ObjIdSize - TagsSize) / (PropIdSize + TagsSize + ObjIdSize) =
(MaxAPDU - 10) / 9

If the IUT does support sending segmented responses:
MaxPropsPerResp =
((MaxAPDU - SegRespHdrSize) * MaxTxSegs - ObjIdSize - TagsSize) / (PropIdSize + TagsSize + ObjIdSize) =
((MaxAPDU - 5) * MaxTxSegs - 7) / 9

NumPropertiesToUse = min(MaxPropsPerRqst, MaxPropsPerResp)

NumPropertiesToUse = min(MaxPropsPerRqst, MaxPropsPerResp)

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = Object-Identifier,
   'Property Identifier' = Object-Identifier,
   'Property Identifier' = Object-Identifier,
   …
   'Property Identifier' = Object-Identifier
2. RECEIVE ReadPropertyMultiple-ACK,
   'Object Identifier' = (Device ,X),
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X),
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X),
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X),
   …
   'Property Identifier' = Object-Identifier,
   'Property Value' = (Device, X)

[Add new clause **9.23.1.7**, p. 254]

9.23.1.7 Writing Maximum Multiple Properties

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be written to it using a single WritePropertyMultiple request.

Test Concept: A writable property is written to an object in the IUT as many times as can be conveyed in the largest request accepted by the IUT. The calculation of the maximum request size shall be based on the IUT's Max_APDU_Length_Accepted and maximum segments per request.

The procedure to determine the number of values to use is:

MaxAPDU = IUT's Max_APDU_Length_Accepted
MaxRxSegs = IUT's maximum segments accepted per request
MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4
SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6
ObjIdSize = size of (an Object-Identifier) = 5
TagsSize = size of (an open and a close tag) = 2

PropIdSize = size of (chosen property Id) = depends on property ID and includes array index size if required
ValueSize = size of (chosen property value) = depends on property and value chosen

If the IUT does not support receiving segmented requests:
NumPropertiesToWrite =
(MaxAPDU – NonSegRqstHdrSize – ObjIdSize – TagsSize) / (PropIdSize + TagsSize + ValueSize) =
(MaxAPDU – 11) / (PropIdSize + 2 + ValueSize)

If the IUT does support receiving segmented requests:
NumPropertiesToWrite =
((MaxAPDU – SegRqstHdrSize) * MaxRxSegs – ObjIdSize – TagsSize) / PropIdSize =

((MaxAPDU – 6) * MaxRxSegs – 7) / 2

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = P1,
   ‘Array Index’ = A1, -- only if required
   'Property Value' = V1,
   …
   'Property Identifier' = P1,
   ‘Array Index’ = A1, -- only if required
   'Property Value' = V1
2. RECEIVE Simple-ACK
3. VERIFY (P1 = V1)

135.1*a*-8. Specify the behavior of a BACnetARRAY when its size is changed.

Rationale
This change is the addition of several tests related to new specification for the behavior of arrays when they are resized. The new array behavior was added in Addendum 135*a*-8 to Standard 135-2001, now incorporated in Standard 135-2004, and incorporated in (proposed) Addendum 135*b*-2 to Standard 135-2004.

Note: The last group of tests described by this rationale (see Clause 7.3.2.13, Global Group Object Tests) verify the correct operation of features that might not be part of a BACnet device being tested. These features will be added to BACnet by Addendum b to Standard 135-2004, which has not yet been published.

Addendum 135.1*a*-8

[Add new clause **7.3.2.9.8**, p. 58]

7.3.2.9.8 Action Size Changes Action_Text Size Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Action, 12.10.8, and Action_Text, 12.10.9

Purpose: This test case verifies that when the size of the Action array is changed, the size of the Action_Text array is changed accordingly to the same size. If the size of the Action and Action_Text arrays cannot be changed, then this test shall not be performed. If Protocol_Revision is not present, or has a value less than 4, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Command object with resizable Action and Action_Text arrays.

Test Concept: The Action and Action_Text arrays are set to a certain size. They are then increased by writing the Action array element 0, decreased by writing the Action array, increased by writing the Action array and decreased by writing the Action array element 0.

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action,
   'Property Array Index' = 0,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Action = 2, ARRAY INDEX = 0
4. VERIFY Action_Text = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
6. RECEIVE Simple-ACK-PDU
7. VERIFY Action = (the value written in step 5), ARRAY INDEX = 0
8. VERIFY Action_Text = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action,
   'Property Value' = (Action array of length 2)
10. RECEIVE Simple-ACK-PDU
11. VERIFY Action = 2, ARRAY INDEX = 0

12. VERIFY Action_Text = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Command object being tested),
    'Property Identifier' = Action,
    'Property Value' = (Action array of length greater than 2)
14. RECEIVE Simple-ACK-PDU
15. VERIFY Action = (the length of the array written in step 13), ARRAY INDEX = 0
16. VERIFY Action_Text = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Command object being tested),
    'Property Identifier' = Action,
    'Property Array Index' = 0,
    'Property Value' = 2
18. RECEIVE Simple-ACK-PDU
19. VERIFY Action = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Action_Text = 2, ARRAY INDEX = 0

[Add new clause **7.3.2.9.9**, p. 58]

7.3.2.9.9 Action_Text Size Changes Action Size Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Action, 12.10.8, and Action_Text, 12.10.9

Purpose: This test case verifies that when the size of the Action_Text array is changed, the size of the Action array is changed accordingly to the same size. If the size of the Action and Action_Text arrays cannot be changed, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Command object with resizable Action and Action_Text arrays.

Test Concept: The Action and Action_Text arrays are set to a certain size. They are then increased by writing the Action_Text array element 0, decreased by writing the Action_Text array, increased by writing the Action_Text array and decreased by writing the Action_Text array element 0.

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action_Text,
   'Property Array Index' = 0,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Action_Text = 2, ARRAY INDEX = 0
4. VERIFY Action = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action_Text,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
6. RECEIVE Simple-ACK-PDU
7. VERIFY Action_Text = (the value written in step 5), ARRAY INDEX = 0
8. VERIFY Action = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Command object being tested),
   'Property Identifier' = Action_Text,
   'Property Value' = (Action_Text array of length 2)

10. RECEIVE Simple-ACK-PDU
11. VERIFY Action_Text = 2, ARRAY INDEX = 0
12. VERIFY Action = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Command object being tested),
    'Property Identifier' = Action_Text,
    'Property Value' = (Action_Text array of length greater than 2)
14. RECEIVE Simple-ACK-PDU
15. VERIFY Action_Text = (the length of the array written in step 13), ARRAY INDEX = 0
16. VERIFY Action = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Command object being tested),
    'Property Identifier' = Action_Text,
    'Property Array Index' = 0,
    'Property Value' = 2
18. RECEIVE Simple-ACK-PDU
19. VERIFY Action_Text = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Action = 2, ARRAY INDEX = 0

[Add new clause **7.3.2.17.5**, p. 63]

7.3.2.17.5 Number_Of_States and State_Text Size Change Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Number_Of_States, 12.18.11, and State_Text, 12.18.12

Purpose: This test case verifies that when the value of the Number_Of_States property is changed, the size of the State_Text array is changed accordingly to the same size. If the Number_Of_States and the size of the State_Text arrays cannot be changed, then this test shall not be performed. If Protocol_Revision is not present, or has a value less than 4, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Multi-state Input object with writable Number_Of_States and resizable State_Text arrays.

Test Concept: Number_Of_States and the State_Text array are set to a certain size. They are then increased by writing the Number_Of_States, decreased by writing the State_Text array, increased by writing the State_Text array and decreased by writing Number_Of_States.

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Multi-state Input object being tested),
   'Property Identifier' = Number_Of_States,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Number_Of_States = 2
4. VERIFY State_Text = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Multi-state Input object being tested),
   'Property Identifier' = Number_Of_States,
   'Property Value' = (some value greater than 2)
6. RECEIVE Simple-ACK-PDU
7. VERIFY Number_Of_States = (the value written in step 5)
8. VERIFY State_Text = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Multi-state Input object being tested),
   'Property Identifier' = State_Text,

'Property Value' = (State_Text array of length 2)

10. RECEIVE Simple-ACK-PDU
11. VERIFY Number_Of_States = 2
12. VERIFY State_Text = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Multi-state Input object being tested),
    'Property Identifier' = State_Text,
    'Property Value' = (State_Text array of length greater than 2)
14. RECEIVE Simple-ACK-PDU
15. VERIFY Number_Of_States = (the length of the array written in step 13)
16. VERIFY State_Text = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Multi-state Input object being tested),
    'Property Identifier' = Number_Of_States,
    'Property Value' = 2
18. RECEIVE Simple-ACK-PDU
19. VERIFY State_Text = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Number_Of_States = 2

[Add new clause **7.3.2.18.6**, p. 64]

7.3.2.18.6 Number_Of_States and State_Text Size Change Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Number_Of_States, 12.19.11, and State_Text, 12.19.12

Test Steps:

Tests to verify the Number_Of_States value and State_Text array size of Multi-state Output objects are defined in 7.3.2.15.5. Run the tests using a Multi-state Output object.

[Add new clause **7.3.2.19.5**, p. 64]

7.3.2.19.5 Number_Of_States and State_Text Size Change Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Number_Of_States, 12.20.10, and State_Text, 12.20.11

Test Steps:

Tests to verify the Number_Of_States value and State_Text array size of Multi-state Value objects are defined in 7.3.2.15.5. Run the tests using a Multi-state Value object.

[Add new clause **7.3.2.22.9**, p. 84]

7.3.2.22.9 Exception_Schedule Size Change Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Exception_Schedule, 12.24.8

Purpose: This test case verifies that when the size of the Exception_Schedule is changed by writing to the array index the size of the array changes accordingly and any new entries contain an empty List of BACnetTimeValue. If the size of the Exception_Schedule array cannot be changed, then this test shall not be performed. If Protocol_Revision is not present, or has a value less than 4, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Schedule object with a resizable Exception_Schedule array.

Test Concept: The Exception_Schedule array is set to a certain size. It is then increased by writing the its array size, decreased by writing the array, increased by writing the array and decreased by writing the array size.

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Schedule object being tested),
   'Property Identifier' = Exception_Schedule,
   'Property Value' = (Exception_Schedule array of length 2)
2. RECEIVE Simple-ACK-PDU
3. VERIFY Exception_Schedule = (the value written in step 1)
4. VERIFY Exception_Schedule = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Schedule object being tested),
   'Property Identifier' = Exception_Schedule,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
6. RECEIVE Simple-ACK-PDU
7. VERIFY Exception_Schedule = (the value written in step 1 with new entries containing empty Lists of BACnetTimeValue))
8. VERIFY Exception_Schedule = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Schedule object being tested),
   'Property Identifier' = Exception_Schedule,
   'Property Value' = (Exception_Schedule array of length 2)
10. RECEIVE Simple-ACK-PDU
11. VERIFY Exception_Schedule = (the value written in step 9)
12. VERIFY Exception_Schedule = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Schedule object being tested),
    'Property Identifier' = Exception_Schedule
    'Property Value' = (Exception_Schedule array of length greater than 2)
14. RECEIVE Simple-ACK-PDU
15. VERIFY Exception_Schedule = (the value written in step 13)
16. VERIFY Exception_Schedule = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Schedule object being tested),
    'Property Identifier' = Exception_Schedule,
    'Property Array Index' = 0,
    'Property Value' = 2
18. RECEIVE Simple-ACK-PDU
19. VERIFY Exception_Schedule = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Exception_Schedule = 2, ARRAY INDEX = 0

[Insert new clause **7.3.2.13**, p. 58, and renumber existing **7.3.2.13** through **7.3.2.23** accordingly]

[Note: This new clause does not introduce additional Global Group object tests, which are outside the scope of this addendum. The Global Group object is defined in Addendum 135-2004*b*-2.]

7.3.2.13 Global Group Object Tests

7.3.2.13.1 Resizing Group_Member_Names by Writing Group_Members Property Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses (Addendum 135-2004b-2): 12.14.5.1, 12.14.5.3, 12.14.6.3, and 12.14.7.1

Purpose: This test case verifies that when the size of the Group_Members array is changed by writing to it, the size of the Group_Member_Names and Present_Value arrays change accordingly and any new entries contain the specified initialized values. If the Group_Members array cannot be written, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Global Group object with a writable Group_Members property.

Test Concept: The Group_Members array is set to a certain size. It is then increased by writing the array size, decreased by writing the array, increased by writing the array and decreased by writing the array size. At each step the size of the Group_Member_Names and Present_Value arrays are verified and the initialized values of the new elements, if any, are checked.

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Members,
   'Property Array Index' = 0,
   'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Group_Members = 2, ARRAY INDEX = 0
4. VERIFY Group_Member_Names = 2, ARRAY INDEX = 0
5. VERIFY Present_Value = 2, ARRAY INDEX = 0
6. TRANSMIT WriteProperty-Request,
   'Object Identifier' = (the Global Group object being tested),
   'Property Identifier' = Group_Members,
   'Property Array Index' = 0,
   'Property Value' = (some value greater than 2)
7. RECEIVE Simple-ACK-PDU
8. VERIFY Group_Members = (the value written in step 6), ARRAY INDEX = 0
9. VERIFY Group_Member_Names = (the value written in step 6), ARRAY INDEX = 0
10. VERIFY Present_Value = (the value written in step 6), ARRAY INDEX = 0
11. VERIFY Group_Members = (a BACnetDeviceObjectPropertyReference containing
    (Device, Instance number 4194303)),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
12. VERIFY Group_Member_Names = (an empty string),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
13. VERIFY Present_Value = 'Access_Result' = PropertyAccessError (PROPERTY, VALUE_NOT_INITIALIZED),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
14. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Value' = (a one-element array containing any valid
    BACnetDeviceObjectPropertyReference)
15. RECEIVE Simple-ACK-PDU
16. VERIFY Group_Members = 1, ARRAY INDEX = 0
17. VERIFY Group_Member_Names = 1, ARRAY INDEX = 0
18. VERIFY Present_Value = 1, ARRAY INDEX = 0
19. VERIFY Group_Members = (the array written in step 14)
20. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Value' = (an array of two or more valid BACnetDeviceObjectPropertyReference values)

21. RECEIVE Simple-ACK-PDU
22. VERIFY Group_Members = (the size of the array written in step 20), ARRAY INDEX = 0
23. VERIFY Group_Member_Names = (the size of the array written in step 20), ARRAY INDEX = 0
24. VERIFY Present_Value = (the size of the array written in step 20), ARRAY INDEX = 0
25. VERIFY Group_Members = (the array written in step 20)
26. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Members,
    'Property Array Index' = 0,
    'Property Value' = (some value between 0 and the size of the array written in step 20)
27. RECEIVE Simple-ACK-PDU
28. VERIFY Group_Members = (the size of the array written in step 26), ARRAY INDEX = 0
29. VERIFY Group_Member_Names = (the size of the array written in step 26), ARRAY INDEX = 0
30. VERIFY Present_Value = (the size of the array written in step 26), ARRAY INDEX = 0

7.3.2.13.2 Resizing Group_Members by Writing Group_Member_Names Property Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses (Addendum 135-2004b-2): 12.14.5.3, 12.14.6.1, 12.14.6.3, and 12.14.7.1

Purpose: This test case verifies that when the size of the Group_Member_Names array is changed by writing to it, the size of the Group_Members and Present_Value arrays change accordingly and any new entries contain the specified initialized values. If the Group_Member_Names array cannot be written, then this test shall not be performed.

Configuration Requirements: The IUT shall be configured with a Global Group object with a writable Group_Member_Names property.

Test Concept: The Group_Member_Names array is set to a certain size. It is then increased by writing the array size, decreased by writing the array, increased by writing the array and decreased by writing the array size. At each step the size of the Group_Members and Present_Value arrays are verified and the initialized values of the new elements, if any, are checked.

1. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Array Index' = 0,
    'Property Value' = 2
2. RECEIVE Simple-ACK-PDU
3. VERIFY Group_Member_Names = 2, ARRAY INDEX = 0
4. VERIFY Group_Members = 2, ARRAY INDEX = 0
5. VERIFY Present_Value = 2, ARRAY INDEX = 0
6. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Array Index' = 0,
    'Property Value' = (some value greater than 2)
7. RECEIVE Simple-ACK-PDU
8. VERIFY Group_Member_Names = (the value written in step 6), ARRAY INDEX = 0
9. VERIFY Group_Members = (the value written in step 6), ARRAY INDEX = 0
10. VERIFY Present_Value = (the value written in step 6), ARRAY INDEX = 0
11. VERIFY Group_Member_Names = (an empty string),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
12. VERIFY Group_Members = (Device, Instance number 4194303),
    ARRAY INDEX = (some value from 3 through the value written in step 6)
13. VERIFY Present_Value = 'Access_Result' = PropertyAccessError (PROPERTY, VALUE_NOT_INITIALIZED),

ARRAY INDEX = (some value from 3 through the value written in step 6)

14. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Value' = (an array of one Character String)
15. RECEIVE Simple-ACK-PDU
16. VERIFY Group_Member_Names = 1, ARRAY INDEX = 0
17. VERIFY Group_Members = 1, ARRAY INDEX = 0
18. VERIFY Present_Value = 1, ARRAY INDEX = 0
19. VERIFY Group_Member_Names = (the array written in step 14)
20. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Value' = (an array of two or more Character Strings)
21. RECEIVE Simple-ACK-PDU
22. VERIFY Group_Member_Names = (the size of the array written in step 20), ARRAY INDEX = 0
23. VERIFY Group_Members = (the size of the array written in step 20), ARRAY INDEX = 0
24. VERIFY Present_Value = (the size of the array written in step 20), ARRAY INDEX = 0
25. VERIFY Group_Member_Names = (the array of Character Strings written in step 20)
26. TRANSMIT WriteProperty-Request,
    'Object Identifier' = (the Global Group object being tested),
    'Property Identifier' = Group_Member_Names,
    'Property Array Index' = 0,
    'Property Value' = (some value between 0 and the size of the array written in step 20)
27. RECEIVE Simple-ACK-PDU
28. VERIFY Group_Member_Names = (the size of the array written in step 26), ARRAY INDEX = 0
29. VERIFY Group_Members = (the size of the array written in step 26), ARRAY INDEX = 0
30. VERIFY Present_Value = (the size of the array written in step 26), ARRAY INDEX = 0

135.1*a*-9. Clarifying the behavior of a BACnet router when it receives an unknown network message type.

Rationale
This change is a modification of tests related to router behavior to account for changes made in Addendum 135*a*-9 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-9

[Add new clause **10.2.2.7.3**, p.298]

10.2.2.7.3 Unknown Network Layer Message Type For Someone Else

Purpose: This test case verifies that the IUT will not reject a network layer message with an unknown message type when it is destined elsewhere. This test shall not be run if the IUT does not have a Protocol_Revision property or its value is less than 4.

BACnet Reference Clause: 6.6.3.5

Test Steps:

1. TRANSMIT PORT A, DA = IUT, SA = D1A,
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   Message Type = (any value in the range reserved for use by ASHRAE)
2. RECEIVE Port B, DA = D2C, SA = IUT,
   SNET = 1,
   SADR = D1A,
   Message Type = (value from step 1)
3. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Message Type = (any value in the range reserved for use by ASHRAE)
4. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = D1A,
   Hop Count = (any value greater than 1 and less than 255),
   Message Type = (value from step 3)
5. TRANSMIT PORT A, DA = IUT, SA = D1A,
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   Message Type = (any value in the range available for vendor proprietary messages),
   Vendor ID = (any value, when paired with Message Type, that is not supported by the IUT)
6. RECEIVE Port B, DA = D2C, SA = IUT,
   SNET = 1,
   SADR = D1A,
   Message Type = (value from step 1)
7. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Message Type = (any value in the range available for vendor proprietary messages),
   Vendor ID = (any value, when paired with Message Type, that is not supported by the IUT)

8. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = D1A,
   Hop Count = (any value greater than 1 and less than 255),
   Message Type = (value from step 7)

135.1*a*-10. Testing unsupported service request execution.

Rationale
Tests are needed to verify that a device properly handles service requests that the device does not support. A duplicate test is also deleted.

Addendum 135.1*a*-10

[Add new clause **9.39**, p. 284]

9.39 General Testing of Service Execution

This subclause defines the tests necessary to demonstrate that a device can peacefully coexist on a BACnet internetwork. These are general tests that are not associated with any particular network service.

9.39.1 Unsupported Confirmed Services Test

Dependencies: None

BACnet Reference Clause: UNRECOGNIZED_SERVICE, 18.8.9

Purpose: This test case verifies that the IUT will reject any confirmed services that it does not support.

Test Steps:

1. REPEAT X = (all confirmed services that the IUT does not execute) DO {
   TRANSMIT X
   RECEIVE BACnet-Reject-PDU,
   'Reject Reason' = UNRECOGNIZED_SERVICE
   }
2. TRANSMIT (a currently undefined confirmed service)
3. RECEIVE BACnet-Reject-PDU,
   'Reject Reason' = UNRECOGNIZED_SERVICE

Passing Result: The device responds correctly for each unsupported confirmed service.

9.39.2 Unsupported Unconfirmed Services Test

Dependencies: None

Purpose: This test case verifies that the IUT will quietly accept and discard any unconfirmed services that it does not support. When determining the set of services to send to the IUT, the UnconfirmedPrivateTransfer service should be included regardless of whether the IUT supports it or not. The UnconfirmedPrivateTransfer service shall be sent with a vendor ID/Service Number pair not supported by the device.

Configuration Requirements: This test requires that the IUT be placed into a normal operating state in which it will not initiate any requests.

Test Steps:

1. VERIFY System_Status == OPERATIONAL | OPERATIONAL_READ_ONLY
2. REPEAT X = (all unconfirmed services that the IUT does not execute) DO {
   TRANSMIT X
   BEFORE **Internal Processing Fail Time**
   CHECK (verify that the IUT did not reset and that the IUT did not send any packets)

VERIFY System_Status = (the value of System_Status read in step 1)
}

Passing Result: The IUT does not reset and sends no packets in response to the services.

[Delete Clause **13.4.6**, p. 427.]

135.1*a*-11. Reading Entire Arrays

Rationale
A test is needed in order to verify that an entire array in a device can be read using a single ReadProperty request, or if the array cannot be read, then to verify that an appropriate Abort message is transmitted.

Addendum 135.1*a*-11

[Add new clause **9.18.1.3**, p. 239]

9.18.1.3 Reading Entire Arrays

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array and the entire array is requested.

BACnet Reference Clause: 5.4.5.3

Test Steps:

1. VERIFY (Device, X), Object_List = (the entire Object_List array)

Passing Result: The IUT shall respond as indicated, conveying values specified in the EPICS. If the object list is too long to return given the APDU and segmentation limitations of the IUT and TD, an Abort message indicating "segmentation not supported" or “buffer overflow” is a passing result. If an Abort message is received and the IUT has another array that is small enough to read in its entirety without segmentation, then this test shall be repeated using that array. A passing result in this case is that the entire array is returned in response to the ReadProperty request.

135.1*a*-12. Update Negative Tests

Rationale
Negative tests need to be updated as a result of changes to the BACnet standard that clarify the use of error classes and error codes in specific situations. The changes were made in Addendum 135c-8 to Standard 135-2001, now incorporated in Standard 135-2004.

Addendum 135.1*a*-12

[Delete existing clause **9.16.2.4**, pp. 236-237, and replace it with new clause **9.16.2.4**, pp. 236-237]

9.16.2.4 Attempting to Create an Object with an Object Type Object Specifier and an Error in the Initial Values

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 Steps:

1. TRANSMIT CreateObject-Request,
   'Object Type' = (any creatable object type),
   'List Of Initial Values' = (a list of two or more properties and their initial values 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 | OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED | 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 Type' = (any creatable object type),
   'List Of Initial Values' = (a list of two or more properties and their initial values with one of the values being an inappropriate datatype)
5. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE CreateObject-Error PDU,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE
   '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 | INVALID_DATATYPE | OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED | OTHER
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
6. CHECK(Verify that the new object was not created)

[Delete existing clause **9.16.2.5**, p. 238, and replace it with new clause **9.16.2.5**, p.238]

9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values

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 Steps:

1. TRANSMIT CreateObject-Request,
   'Object Identifier' = (any unique object identifier of a type that is creatable),
   'List Of Initial Values' = (a list of two or more properties and their initial values 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 | INVALID_DATATYPE
   'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
3. TRANSMIT CreateObject-Request,
   'Object Identifier' = (any unique object identifier of a type that is creatable),
   'List Of Initial Values' = (a list of two or more properties and their initial values with one of the values being an inappropriate datatype)
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE CreateObject-Error PDU,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE
   '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 = 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)
5. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (the 'Object Identifier' used in step 1),
   'Property Identifier' = (any required property of the specified object)
6. IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
   RECEIVE BACnet-Error PDU,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT
   ELSE
   RECEIVE BACnet-Error PDU
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT | NO_OBJECTS_OF_SPECIFIED_TYPE | OTHER

[Change clause **9.22.2.1**, p. 250]

9.22.2.1 Writing Non-Array Properties with an Array Index

Purpose: This test case verifies 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. 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 verified 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~~)~~ |
~~(BACnet-Reject-PDU,~~
~~Reject Reason = INCONSISTENT_PARAMETERS)~~
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

[Change clause **9.22.2.3,** p. 251]

9.22.2.3 Writing with a Property Value Having the Wrong Datatype

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 in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

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 with an invalid datatype)
3. *IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN*
   RECEIVE BACnet-Error PDU,
   Error Class = *PROPERTY,*
   Error Code = *INVALID_DATATYPE*
   ELSE
   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 = (the value defined for this property in the EPICS)

[Change clause **9.22.2.4**, p. 252]

9.22.2.4 Writing with a Property Value that is Out of Range

Purpose: This test case verifies 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.

Test Steps:

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 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 = (the value defined for this property in the EPICS)

[Add new clause **9.22.2.5**, p. 252]

9.22.2.5 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the object specified in the service request does not exist. 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, that does not exist in the IUT. Object1 shall be of a type supported by the IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE BACnet-Error PDU,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT

Passing Result: While OBJECT::UNKNOWN_OBJECT is the desired error for this condition, in some implementations other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are:
PROPERTY::UNKNOWN_PROPERTY,
PROPERTY::WRITE_ACCESS_DENIED,
PROPERTY::INVALID_DATATYPE,
PROPERTY::VALUE_OUT_OF_RANGE and
RESOURCES::NO_SPACE_TO_WRITE_PROPERTY.

[Add new clause **9.22.2.6**, p. 252]

9.22.2.6 Writing To Non-Existent Properties

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the property specified in the service request is not supported by the object specified 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 in the IUT, designated Object1. The TD shall select a property, designated P1, that is not supported by the specific object instance. An attempt will be made to write to this property.

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any valid value for this property)
2. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = UNKNOWN_PROPERTY

[Add new clause **9.22.2.7**, p. 252]

9.22.2.7 Writing To Non-Writable Properties

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the property specified in the service request is not writable. 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 non-writable property designated P1. An attempt will be made to write to this property. If no object supports non-writable 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)
2. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = WRITE_ACCESS_DENIED

[Add new clause **9.22.2.8**, p. 252]

9.22.2.8 Writing An Object_Name With A Value That Is Already In Use

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when an Object_Name property is written to with a value already in use by a different object in the device. 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 Object_Name property. An attempt will be made to write to this property with a value that is in use by the Object_Name property of another object in the device. If no object supports writable Object_Name properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WriteProperty-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = Object_Name,
   'Property Value' = (an Object_Name value already in use by another object)
2. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = DUPLICATE_NAME

[Add new clause **9.23.2.4**, p. 256]

9.23.2.4 Writing Non-Array Properties with an Array Index

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property value is not an array but an array index is included in the service request. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable scalar property designated P1. An attempt will be made to write to this property using an array index. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. 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 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 verified in step 1),
   `Property Array Index' = (any positive integer)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = PROPERTY_IS_NOT_AN_ARRAY,
   objectIdentifier = Object1,
   propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

[Add new clause **9.23.2.5**, p. 256]

9.23.2.5 Writing Array Properties with an Array Index that is Out of Range

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the requested property value is an array but the array index is out of range. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an array index that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. 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 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 verified in step 1),
   `Property Array Index' = (any positive integer that is larger that the supported size of the array)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_ARRAY_INDEX,
   objectIdentifier = Object1,
   propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

[Add new clause **9.23.2.6**, p. 256]

9.23.2.6 Writing with a Property Value Having the Wrong Datatype

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype. 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. An attempt will be made to write to this property using an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

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 value with an invalid datatype)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE,
   objectIdentifier = Object1,
   propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

[Add new clause **9.23.2.7**, p. 256]

9.23.2.7 Writing with a Property Value that is Out of Range

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 this property using a value that is outside of the supported range.

Test Steps:

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, of the correct datatype, that is outside the supported range)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = VALUE_OUT_OF_RANGE,
   objectIdentifier = Object1,
   propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

[Add new clause **9.23.2.8**, p. 256]

9.23.2.8 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the object specified in the service request does not exist. 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, that does not exist in the IUT. Object1 shall be of a object type supported by IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE WritePropertyMultiple-Error,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT,
   objectIdentifier = Object1,
   propertyIdentifier = P1

Passing Result: While OBJECT::UNKNOWN_OBJECT is the desired error for this condition, in some implementations, other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are:
PROPERTY::UNKNOWN_PROPERTY,
PROPERTY::WRITE_ACCESS_DENIED,
PROPERTY::INVALID_DATATYPE,
PROPERTY::VALUE_OUT_OF_RANGE and
RESOURCES::NO_SPACE_TO_WRITE_PROPERTY.

[Add new clause **9.23.2.9**, p. 256]

9.23.2.9 Writing To Non-Existent Properties

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property specified in the service request is not supported by the object specified 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 in the IUT, designated Object1. The TD shall select a property, designated P1, that is not supported by the object. An attempt will be made to write to this property.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value of the correct datatype for the property),
2. RECEIVE BACnet-Error PDU,
   Error Class = PROPERTY,
   Error Code = UNKNOWN_PROPERTY,
   objectIdentifier = Object1,
   propertyIdentifier = P1

[Add new clause **9.23.2.10**, p. 256]

9.23.2.10 Writing To Non-Writable Properties

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property specified in the service request is not writable. 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 Object 1, in the IUT that contains a non-writable property designated P1. An attempt will be made to write to this property. If no object supports non-writable properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = WRITE_ACCESS_DENIED,
   objectIdentifier = Object1,
   propertyIdentifier = P1

[Add new clause **9.23.2.11**, p. 256]

9.23.2.11 Writing An Object_Name With A Value That Is Already In Use

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an Object_Name property is written to with a value already in use by a different object in the device. 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 Object_Name property. An attempt will be made to write to this property with a value that is in use by the Object_Name property of another object in the device. If no object supports writable Object_Name properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = Object_Name,
   'Property Value' = (an Object_Name value already in use by another object)
2. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = DUPLICATE_NAME

[Change clause **9.18.2.1**, p. 239]

9.18.2.1 Reading Non-Array Properties with an Array Index

Purpose: This test case verifies that the IUT can execute ReadProperty service requests when the requested property value is not an array but an array index is included in the service request.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = Vendor_Name,
   'Array Index' = 1
2. *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-Reject-PDU,~~
   ~~'Reject Reason' = INCONSISTENT_PARAMETERS) |~~
   (BACnet-Error-PDU,
   Error Class = PROPERTY,
   Error Code = *PROPERTY_IS_NOT_AN_ARRAY* |
   INVALID_ARRAY_INDEX) |
   (BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = INCONSISTENT_PARAMETERS)

[Add new clause **9.20.2.3**, p. 246]

9.20.2.3 Reading a Single Non-Array Property with an Array Index

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests when the requested 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 Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
   ‘Object Identifier’ = (Device, X),
   ‘Property Identifier’ = Vendor_Name,
   ‘Array Index’ = 1
2. RECEIVE
   (BACnet-Error-PDU,
   ‘Error Class’ = PROPERTY,

‘Error Code’ = PROPERTY_IS_NOT_AN_ARRAY) |
(ReadPropertyMultiple-ACK,
‘Object Identifier’ = (Device, X),
‘Property Identifier’ = Vendor_Name,
‘Array Index’ = 1,
‘Error Class’ = PROPERTY,
‘Error Code’ = PROPERTY_IS_NOT_AN_ARRAY)

[Change clause **13.4.3**, p. 426]

13.4.3 Invalid Tag

Purpose: This test case verifies that the IUT correctly responds to a message containing an invalid data tag.

Test Concept: The TD transmits a ReadProperty service request that has an invalid tag for the 'Property_Identifier' parameter.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object_Identifier' = (any object in the IUT's database),
   'Property_Identifier' = (any valid property for the object, but the tag shall have a number x: 2 < x < 254)
2. RECEIVE BACnet-Reject-PDU,
   Reject Reason = INVALID_TAG |
   INCONSISTENT_PARAMETERS |
   INVALID_PARAMETER_DATA_TYPE |
   MISSING_REQUIRED_PARAMETER |
   TOO_MANY_ARGUMENTS

[Change clause **13.4.4**, p. 426]

13.4.4 Missing Required Parameter

Purpose: This test case verifies that the IUT correctly responds to a message that is missing a required parameter.

Test Concept: The TD transmits a ReadProperty service request that does not include a 'Property Identifier' parameter.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object_Identifier' = (any object in the IUT's database),
2. RECEIVE BACnet-Reject-PDU,
   Reject Reason = MISSING_REQUIRED_PARAMETER |
   INVALID_TAG

[Change clause **13.4.5**, p. 427]

13.4.5 Too Many Arguments

Purpose: This test case verifies that the IUT correctly responds to a message that conveys too many arguments.

Test Concept: The TD transmits a ReadProperty service request that conveys an extra property identifier.

Test Steps:

1. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (any supported object),
   'Property Identifier = (any valid property identifier for the specified object),
   'Property Identifier' = (any valid property identifier for the specified object but not used in the previous parameter)
2. RECEIVE BACnet-Reject-PDU,
   Reject Reason = TOO_MANY_ARGUMENTS | *INVALID_TAG*

POLICY STATEMENT DEFINING ASHRAE'S CONCERN FOR THE ENVIRONMENTAL IMPACT OF ITS ACTIVITIES

ASHRAE is concerned with the impact of its members' activities on both the indoor and outdoor environment. ASHRAE's members will strive to minimize any possible deleterious effect on the indoor and outdoor environment of the systems and components in their responsibility while maximizing the beneficial effects these systems provide, consistent with accepted standards and the practical state of the art.

ASHRAE's short-range goal is to ensure that the systems and components within its scope do not impact the indoor and outdoor environment to a greater extent than specified by the standards and guidelines as established by itself and other responsible bodies.

As an ongoing goal, ASHRAE will, through its Standards Committee and extensive technical committee structure, continue to generate up-to-date standards and guidelines where appropriate and adopt, recommend, and promote those new and revised standards developed by other responsible organizations.

Through its *Handbook*, appropriate chapters will contain up-to-date standards and design considerations as the material is systematically revised.

ASHRAE will take the lead with respect to dissemination of environmental information of its primary interest and will seek out and disseminate information from other responsible organizations that is pertinent, as guides to updating standards and guidelines.

The effects of the design and selection of equipment and systems will be considered within the scope of the system's intended use and expected misuse. The disposal of hazardous materials, if any, will also be considered.

ASHRAE's primary concern for environmental impact will be at the site where equipment within ASHRAE's scope operates. However, energy source selection and the possible environmental impact due to the energy source and energy transportation will be considered where possible. Recommendations concerning energy source selection should be made by its members.