ADDENDA

ANSI/ASHRAE Addendum m to
ANSI/ASHRAE Standard 135.1-2011

Method of Test for Conformance to BACnet®

Approved by the ASHRAE Standards Committee on October 2, 2012; by the ASHRAE Board of Directors on October 26, 2012; and by the American National Standards Institute on October 27, 2012.

This addendum was approved 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 (www.ashrae.org) or in paper form from the Manager of Standards.

The latest edition of an ASHRAE Standard may be purchased on the ASHRAE Web site (www.ashrae.org) or 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). For reprint permission, go to www.ashrae.org/permissions.

 ISSN 1041-2336

Approved American National Standard
ANSI
www.ansi.org

ASHRAE Standing Standard Project Committee 135
Cognizant TC: TC 1.4, Control Theory and Application
SPLS Liaison: Richard L. Hall

Carl Neilson, *Chair*
David Robin, *Chair* (2008–2012)*
Bernhard Isler, *Secretary**
Donald P. Alexander*
Chandrashekhar Appanna
Tomohino Asazuma
Dave Bohlmann
Barry B. Bridges*
Coleman L. Brumley, Jr.
Ernest C. Bryant
Steve Bushby
Jim Butler
Ryan Bykowski
Howard Coleman
Clifford H. Copass
Sharon E. Dinges*
Stuart G. Donaldson
Hu Dou
David Fisher
Rokuro Fuji
Fumio Fujimura
Noriaki Fujiwara
Craig Gemmill
Andrey Golovin, *International Liaison*
Nils-Gunnar Fritz
Rod Harruff
John Hartman
Teemu T Heikkil
David G. Holmberg
Masahiro Ishiyama
Hiroshi Ito
Kosuke Ito
Sudhir Jaiswal
John Rohde Jensen
Robert L. Johnson
Chris Jones
René Kälin
Stephen Karg*
Koji Kimura
Duane L. King
Bruno Kloubert
Daniel Kollodge
Thomas Kurowski
Roland Laird
Brett Leida
Rick Leinen
Simon Lemaire
Joe Lenart
J. Damian Ljungquist*
John Lundstedt
James G. Luth
John J. Lynch
Kerry Lynn
Graham Martin
Jerry Martocci
Hirotaka Masui
Konni Mergner
Brian D. Meyers
Charles Miltiades
Venkatesh Mohan
Tsuyoshi Momose
Hans-Joachim Mundt
Masaharu Nakamura
Mike Newman
Duffy O'Craven
Hideya Ochiai
Bob Old
Farhad Omar
Dave Oravetz
Mike Osborne
Bill Pienta
Dana Petersen
René Quirighetti
Suresh Ramachandran
Douglas T. Reindl, *SPLS Liaison*
David Ritter
William Roberts
Carl J. Ruther
Frank Schubert
Atsushi Shimadate
Brad Spencer
Gregory M. Spiro
Ted Sunderland
William O. Swan, III
Hans Symanczik
Bob Thomas
David B. Thompson*
Takeji Toyoda Jr., *International Liaison*
Stephen J. Treado*
Klaus Wächter, *International Liaison*
Klaus Wagner
Mark J. Weber, *Staff Liaison*
Bruce Westphal
J. Michael Whitcomb*
Grant N. Wichenko*
Cam Williams
Ove Wiuff
Christoph Zeller
Ming Zhu
Scott Ziegenfus
Rob Zivney

**Denotes members of voting status when the document was approved for publication*

SPECIAL NOTE

This American National Standard (ANS) is a national voluntary consensus standard developed under the auspices of 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, or
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

Addendum *m* to ANSI/ASHRAE Standard 135.1-2011 contains a number of changes to the current standard.These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135.1-2011*m*-1. Add Network Priority Test, p. 1.
135.1-2011*m*-2. Add Virtual Router Tests, p. 3.
135.1-2011*m*-3. Replace Time Master Tests, p. 22.
135.1-2011*m*-4. Add Backup and Restore Tests, p. 28.
135.1-2011*m*-5. Add APDU Retry Test, p. 41.
135.1-2011*m*-6. Add Workstation Schedule Interaction Tests, p. 42.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135.1-2011 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout.

135.1-2011*m*-1. Add Network Priority Test.

Rationale

Add a test to verify that responses are sent with the same network priority as the corresponding request.

[Change **Clause 10.1**, p. 365]
[The existing Clause 10.1 is renumbered 10.1.1 to allow for the new test to be added in the same general network section.]

10.1 General Network Layer Tests

10.1.*1* Processing Application Layer Messages Originating from Remote Networks

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 6.5.4.

Purpose: To verify that the IUT can respond to requests that originate from a remote network.

Test Concept: The TD transmits a ReadProperty-Request message that contains network layer information indicating that it originated from a remote network. The response from the IUT shall include correct DNET and DADR information so that the message can reach the original requester. The MAC layer destination address in the response can be either a broadcast, indicating that the IUT does not know the address of the router, or the MAC address of the appropriate router.

Test Steps:

1. TRANSMIT
   DESTINATION = IUT,
   SOURCE = TD,
   SNET = (any network number that is not the local network),
   SADR = (any valid MAC address consistent with the source network),
   ReadProperty-Request,
   'Object Identifier' = (any supported object),
   'Property Identifier' = (any required property of the specified object)
2. RECEIVE
   DESTINATION = LOCAL BROADCAST | (an appropriate router address),
   SOURCE = IUT,
   DNET = (the SNET specified in step 1),
   DADR = (the SADR specified in step 1),
   Hop Count = 255,
   ReadProperty-ACK,
   'Object Identifier' = (the object specified in step 1),
   'Property Identifier' = (the property specified in step 1),
   'Property Value' = (*any valid value*~~the value of the specified property as defined in the EPICS~~)

[Add new **Clause 10.1.2**, p. 310]

10.1.2 Network Layer Priority

Purpose: To verify that the IUT can process messages with all network priorities.

Test Concept: For each network layer priority, send a confirmed request to the IUT, and verify that the response has the same priority as the request.

Test Steps:

1. REPEAT PRIO = (Each valid network priority) DO {
   TRANSMIT
   SOURCE = TD,
   DESTINATION = IUT,
   'Network Priority' = PRIO,
   ReadProperty-Request,
   'Object Identifier' = (O1, any object in the target device),
   'Property Identifier' = (P1, any property of O1)
   RECEIVE
   SOURCE = IUT,
   DESTINATION = TD,
   'Network Priority' = PRIO,
   ReadProperty-Request,
   'Object Identifier' = O1
   'Property Identifier' = P1,
   'Property Value' = (any valid value)
   }

135.1-2011*m*-2. Add Virtual Router Tests.

Rationale

These tests were developed by the BTL for testing gateways that model data as a collection of virtual devices on a virtual network.

[Add new **Clause 10.8**, p. 406]

10.8 Virtual Routing Functionality Tests

Some gateway devices can simulate routers between a physical BACnet network and one or more virtual BACnet networks that contain one or more virtual BACnet devices. See Annex H in the BACnet standard for a description of virtual BACnet networks and virtual BACnet devices.

This clause defines the tests necessary to demonstrate routing functionality to/from virtual BACnet networks. The tests assume that the routing device has at least two ports, one connected to a virtual BACnet network containing one or more virtual BACnet devices, and one connected to a physical BACnet network. IUT Port 1 is directly connected to Network 1 (a virtual BACnet network), and Port 2 is directly connected to Network 2 (a physical BACnet network). The logical configuration of the internetwork used for these tests is shown in Figure 10.4. The test descriptions in this clause assume that the TD can physically connect to Network 2 and mimic all of the other devices. An acceptable alternative is to construct an internetwork with real devices as indicated. Logical network 3 shall use a network technology that has MAC addresses that are different in length from Network 2.

The logical devices included in the internetwork are:

IUT: implementation under test, a router between Networks 1 and 2
VD1A: virtual device on Network 1
VD1B: virtual device on Network 1
D2C: device on Network 2
D3D: device on Network 3
R2-3: router between Network 2 and Network 3

General Configuration Requirements: The IUT shall be configured with routing tables indicating that Network 1 is directly connected to Port 1 and that Network 2 is directly connected to Port 2 as shown in Figure 10-4. The tests assume that the IUT has exactly 2 ports. For IUTs that cannot be configured to have exactly 2 ports, the tester shall modify each test's expectations to take into account the extra ports. The routing device shall be configured to have one or more virtual devices (VD1A, VD1B, etc.), on Network 1. Although the network numbers 1-3 are used above and below, the tester may configure the network using any legal network numbers and modify the tests accordingly. Furthermore, the tester shall appropriately modify the tests for devices that route to multiple virtual networks simultaneously.

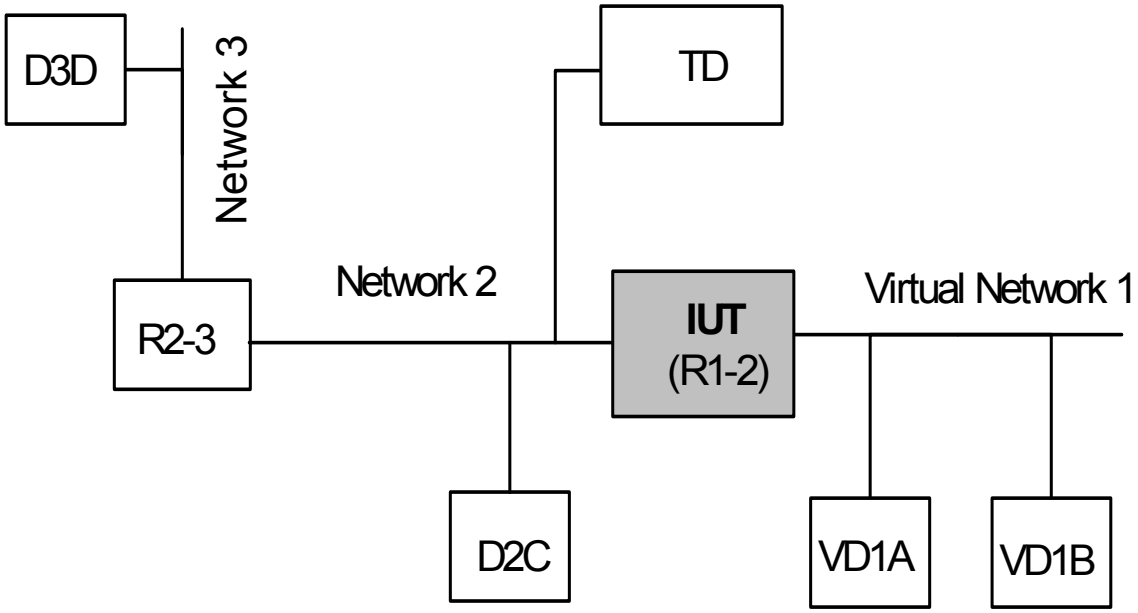

Figure 10-4. Logical internetwork configuration for virtual routing functionality tests

10.8.1 Startup

Purpose: To verify that the IUT will broadcast an appropriate I-Am-Router-To-Network message upon startup.

Test Steps:

1. MAKE (power cycle the router to make it reinitialize)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   I-Am-Router-To-Network,
   Network Numbers = 1

Notes to Tester: If the IUT routes to multiple virtual networks simultaneously, then all of their network numbers shall be reported in one or more I-Am-Router-To-Network messages.

10.8.2 Processing Network Layer Messages

10.8.2.1 Execution of Who-Is-Router-To-Network

10.8.2.1.1 No Specified Network Number

Purpose: To verify that the IUT will broadcast an I-Am-Router-To-Network message listing all downstream virtual networks when it receives a Who-Is-Router-To-Network message with no specified network number.

Test Steps:

1. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SOURCE = TD,
   Who-Is-Router-to-Network
2. RECEIVE
   DESTINATION = LOCAL BROADCAST,
   SOURCE = IUT,
   I-Am-Router-To-Network,
   Network Numbers = 1

Notes to Tester: If the IUT routes to multiple virtual networks simultaneously, then all of their network numbers shall be reported in one or more I-Am-Router-To-Network messages.

10.8.2.1.2 A Known Remote Network Number is Specified

Purpose: To verify that the IUT will broadcast an appropriate I-Am-Router-To-Network message when it receives a Who-Is-Router-To-Network message with a specified network number that is included in the routing table.

Test Steps:

1. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SOURCE = TD,
   Who-Is-Router-To-Network,
   Network Number = 1
2. RECEIVE
   DESTINATION = LOCAL BROADCAST,
   SOURCE = IUT,
   I-Am-Router-To-Network,
   Network Numbers = 1

10.8.2.1.3 A Network Number is Specified and the Router Does Not Respond

Purpose: To verify that the IUT does not respond if it receives a Who-Is-Router-To-Network message specifying a network number for a network that is known to be reachable through the same port through which the I-Am-Router-To-Network message was received.

Test Steps:

1. TRANSMIT
   DESTINATION = LOCAL BROADCAST,

SOURCE = TD,
Who-Is-Router-To-Network,
Network Number = 2

2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not transmit I-Am-Router-To-Network (Network Numbers = 2, ...) or Who-Is-Router-To-Network (Network Number = 2))
4. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SOURCE = R2-3,
   I-Am-Router-To-Network,
   Network Numbers = 3
5. WAIT **Internal Processing Fail Time**
6. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SOURCE = TD,
   Who-Is-Router-To-Network,
   Network Number = 3
7. WAIT **Internal Processing Fail Time**
8. CHECK (verify that the IUT does not transmit I-Am-Router-To-Network (Network Numbers = 3, ...) or Who-Is-Router-To-Network (Network Number = 3))

10.8.2.1.4 An Unknown Network Number is Specified

Purpose: To verify that, if the IUT receives a Who-Is-Router-To-Network message specifying an unknown network number, it will not transmit a Who-Is-Router-To-Network message on the same network.

Test Steps:

1. TRANSMIT
   DESTINATION = LOCAL BROADCAST,
   SOURCE = TD,
   Who-Is-Router-To-Network,
   Network Number = 35001
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not transmit I-Am-Router-To-Network (Network Numbers = 35001, ...) or Who-Is-Router-To-Network (Network Number = 35001) on Network 2)

10.8.2.2 Reject-Message-To-Network

This clause tests some of the possible circumstances where a message should be rejected by the network layer.

10.8.2.2.1 Unknown Network

Purpose: To verify that the IUT will reject a message sent to the IUT if it is addressed to a device on an unknown and unreachable DNET.

Test Steps:

1. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 59001,
   DADR = (any valid MAC address),
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (any object identifier),

'Property Identifier' = (any property of the specified object)

2. RECEIVE
   DA = TD,
   SA = IUT,
   Reject-Message-To-Network,
   Reject Reason = 1 -- unknown destination network
   DNET = 59001
3. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 59001, ...) on Network 2)

10.8.2.2.2 Unknown Network Layer Message Type

Purpose: To verify that the IUT will reject a network layer message directed to the IUT if it contains an unknown message type that is in the range of message types reserved for use by ASHRAE.

Test Steps:

1. TRANSMIT
   DESTINATION = IUT,
   SOURCE = TD,
   Message Type = (any value in the range reserved for use by ASHRAE)
2. RECEIVE
   DESTINATION = TD,
   SOURCE = IUT,
   Reject-Message-To-Network,
   Reject Reason = 3 -- unknown network layer message type
   DNET = (any value)

10.8.3 Routing of Unicast APDUs

10.8.3.1 Route Request Message from a Local Device to a Virtual Device and Route Response Message from the Virtual Device to the Local Device

Purpose: To verify that the IUT can route a unicast request message from a local device to a virtual device and route the response from the virtual device to the local device.

Test Steps:

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (the object identifier of any object in the target device),
   'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)
2. RECEIVE
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   ReadProperty-ACK,
   'Object Identifier' = (the object identifier used in step 1),
   'Property Identifier' = (the property identifier used in step 1),

'Property Value' = (the contents of the specified property)

3. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (the object identifier of any object in the target device),
   'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented, but not the same property as in step 1)
4. RECEIVE
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   ReadProperty-ACK,
   'Object Identifier' = (the object identifier used in step 3),
   'Property Identifier' = (the property identifier used in step 3),
   'Property Value' = (the contents of the specified property)

10.8.3.2 Route Request Message from a Virtual Device to a Local Device

Purpose: To verify that the IUT can route a unicast request message from a virtual device to a local device.

Test Concept: Make one of the virtual devices generate a unicast request, and verify that the NPCI is correctly formed. This test shall be skipped if none of the IUT's virtual devices can issue a confirmed or unconfirmed request in a unicast message.

Test Steps:

1. MAKE (the virtual device generate a unicast APDU to a device on the IUT's local network)
2. RECEIVE
   DA = TD
   SA = IUT
   SNET = 1,
   SADR = (MAC address of the virtual device),
   BACnet-Confirmed-Request-PDU or BACnet-Unconfirmed-Request-PDU

Notes to Tester: During the test, the TD shall answer any requests that the IUT generates while attempting to locate the route to the target device.

10.8.3.3 Route Request Message from a Remote Device to a Virtual Device and Route Response Message from the Virtual Device to the Remote Device

Purpose: To verify that the IUT can route a unicast request message from a remote device to a virtual device and route the response from the virtual device to the remote device.

Test Steps:

1. TRANSMIT
   DA = IUT,
   SA = R2-3,
   DNET = 1,
   DADR = VD1A,
   SNET = 3,

SADR = D3D,
Hop Count = 254,
ReadProperty-Request,
'Object Identifier' = (the object identifier of any object in the target device),
'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)

2. RECEIVE
   DA = R2-3,
   SA = IUT,
   DNET = 3,
   DADR = D3D,
   SNET = 1,
   SADR = VD1A,
   Hop Count = (any integer x: 1 < x < 255),
   ReadProperty-Request,
   'Object Identifier' = (the object identifier used in step 1),
   'Property Identifier' = (the property identifier used in step 1),
   'Property Value' = (the contents of the specified property)

10.8.3.4 Route Request Message from a Virtual Device to a Remote Device

Purpose: To verify that the IUT can route a unicast message from a virtual device to a remote device.

Test Concept: While the IUT is unaware of the route to a remote network, make one of the virtual devices generate a unicast request that is destined for a remote network, and verify that the NPCI is correctly formed. This test shall be skipped if none of the IUT's virtual devices can issue a confirmed or unconfirmed request in a unicast message.

Configuration Requirements: The IUT shall be configured such that its routing table shall only contain entries for the directly connected networks (physical and virtual).

Test Steps:

1. MAKE (the virtual device generate a unicast APDU to a device on a remote network)
2. RECEIVE
   DA = R2-3,
   SA = IUT,
   DNET = 3,
   DADR = D3D,
   SNET = 1,
   SADR = (MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   BACnet-Confirmed-Request-PDU or BACnet-Unconfirmed-Request-PDU

Notes to Tester: During the test, the TD should be prepared to answer any requests that the IUT generates while attempting to locate the route to the target device.

10.8.3.5 Unicast Messages that Should Not Be Routed

10.8.3.5.1 Unknown Network

Purpose: To verify that the IUT will not attempt to route a message directed to a device on an unknown network if the message was transmitted using a local broadcast MAC address.

Test Concept: Direct at one of the virtual devices a ReadProperty request that is correct in all aspects, except for the network number. Ensure that the virtual device does not reply. The request is sent as a local broadcast so that the IUT will receive it and not attempt to re-route it via another router to the unknown network.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 59001,
   DADR = (the MAC address of the selected virtual device),
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (any object identifier of an object in the virtual device),
   'Property Identifier' = (any property of the specified object)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 59001, ...) or Reject-Message-To-Network (Network Number = 59001) or any message in response to the ReadProperty request on Network 2)

10.8.3.5.2 Network Reachable Through the Same Port

Purpose: To verify that the IUT will not attempt to route a message directed to a device on a known network reachable through the same port if the message was transmitted using a local broadcast MAC address.

Test Steps:

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3
   I-Am-Router-To-Network,
   Network Numbers = 3
2. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 3,
   DADR = D3D,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (any object identifier),
   'Property Identifier' = (any property of the specified object)
3. WAIT **Internal Processing Fail Time**
4. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 3, ...) or Reject-Message-To-Network (Network Number = 3) or any message in response to the ReadProperty request on Network 2)

10.8.4 Routing of Broadcast APDUs to Virtual Devices

10.8.4.1 Broadcasts that Should Be Ignored

Purpose: To verify that the IUT will not route APDUs which are locally broadcast on the directly connected physical BACnet network or remotely broadcast to a network that is reachable through the same port the message was received from.

Test Concept: In order to verify that a broadcast message is not routed, we need to look for some indication that the message was routed. Two commonly supported services that can be used for this test are Who-Is and Who-Has, but complications may arise because a device is allowed to transmit the expected responses (I-Am and I-Have, respectively) at any time. There are a few other services that may also be used, but special device configuration will most likely be required. The tester shall choose one of the following test options, with the requirement that the option chosen must use services supported by at least one of the IUT's virtual devices.

This test shall be skipped if the IUT's virtual devices are not capable of executing any application services that may be broadcast.

A. Who-Is Option

This test option is an alternative for IUTs having a virtual device that supports the execution of the Who-Is service. This test shall be run after all tests for the execution of the Who-Is service have been run with a passing result. One virtual device that supports the execution of the Who-Is service shall be selected for this test.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not transmit an I-Am message from the virtual device selected in step 1)
4. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   SNET = 3,
   SADR = D3D,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device)
5. WAIT **Internal Processing Fail Time**
6. CHECK (verify that the IUT does not transmit an I-Am message from the virtual device selected in step 4)
7. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 3,
   DLEN = 0,
   Hop Count = 255,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device)
8. WAIT **Internal Processing Fail Time**
9. CHECK (verify that the IUT does not transmit an I-Am message from the virtual device selected in step 7)

Notes to Tester: I-Am messages may be sent at any time by any device. If one or both of the CHECK steps fail, then repeat the test until you can determine with confidence whether the I-Am messages are in response to the Who-Is (a failing result) or are being transmitted for some other reason (a passing result).

B. Who-Has Option

This test option is an alternative for IUTs that have a virtual device that supports the execution of the Who-Has service. This test shall be run after all tests for the execution of the Who-Has service have been run with a passing result. One virtual device that supports the execution of the Who-Has service shall be selected for this test.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device),
   'Object Identifier' = (any object identifier of an object in the virtual device)

2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not transmit an I-Have message from the virtual device selected in step 1 that contains the object identifier used in step 1)
4. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   SNET = 3,
   SADR = D3D,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device),
   'Object Identifier' = (any object identifier of an object in the virtual device)
5. WAIT **Internal Processing Fail Time**
6. CHECK (verify that the IUT does not transmit an I-Have message from the virtual device selected in step 4 that contains the object identifier used in step 4)
7. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 3,
   DLEN = 0,
   Hop Count = 255,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device),
   'Object Identifier' = (any object identifier of an object in the virtual device)
8. WAIT **Internal Processing Fail Time**
9. CHECK (verify that the IUT does not transmit an I-Have message from the virtual device selected in step 4 that contains the object identifier used in step 7)

Notes to Tester: I-Have messages may be sent at any time by any device. If one or both of the CHECK steps fail, then repeat the test until you can determine with confidence whether the I-Have messages are in response to the Who-Has (a failing result) or are being transmitted for some other reason (a passing result).

C. Generic Test Option

This test option is intended for IUTs whose virtual devices do not support the execution of the Who-Has or Who-Is services. This option uses a service that may be broadcast and that is supported by one of the IUT's virtual devices. This test shall be run after all tests for the execution of the particular service have been run with a passing result.

Configuration requirements: Configure the IUT so that the receipt of a particular service request by a particular virtual device will cause some indication that is visible to the tester. Verify that the indication occurs if the service request is sent directly to the virtual device or remote broadcast on Network 1.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   BACnet-Unconfirmed-Request-PDU
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the indication does not occur)
4. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   SNET = 3,
   SADR = D3D,
   BACnet-Unconfirmed-Service-Request
5. WAIT **Internal Processing Fail Time**

6. CHECK (verify that the indication does not occur)
7. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 3,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Service-Request
8. WAIT **Internal Processing Fail Time**
9. CHECK (verify that the indication does not occur)

10.8.4.2 Route Global Broadcast from a Local Device to Virtual Devices

Purpose: To verify that the IUT properly forwards global broadcast messages originating on a local network to its virtual devices. This test shall be skipped if the IUT's virtual devices are not capable of executing any application services that may be broadcast.

Test Concept: A broadcast message of a type that the virtual device will execute is sent to the IUT. It is then verified that a correct response is generated.

The tester selects one of the following test options, depending on which services are supported by the virtual devices in the IUT.

A. Who-Is Test Option

Select one virtual device in the IUT that supports the execution of Who-Is requests.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = D2C,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Am-Request,
   'I Am Device Identifier' = (the Device object instance number of the virtual device),
   'Max APDU Length Accepted' = (any valid value),
   'Segmentation Supported' = (any valid value),
   'Vendor Identifier' = (any valid value)
   | (DA = LOCAL BROADCAST | D2C,
   SA = IUT,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   I-Am-Request,
   'I Am Device Identifier' = (the Device object instance number of the virtual device),
   'Max APDU Length Accepted' = (any valid value),

'Segmentation Supported' = (any valid value),
'Vendor Identifier' = (any valid value))

B. Who-Has Test Option

Select one virtual device in the IUT that supports the execution of Who-Has requests.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device),
   'Object Identifier' = (any object identifier of an object in the virtual device)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Have-Request,
   'Device Identifier' = (the Device object instance number of the virtual device),
   'Object Identifier' = (the object identifier specified in step 1),
   'Object Name' = (any valid value)
   | (DA = LOCAL BROADCAST,
   SA = IUT,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   I-Have-Request,
   'Device Identifier' = (the Device object instance number of the virtual device),
   'Object Identifier' = (the object identifier specified in step 1),
   'Object Name' = (any valid value))

C. Generic Test Option

This test option is intended for IUTs whose virtual devices do not support the execution of the Who-Has or Who-Is services. This option uses a service that may be broadcast and that is supported by one of the IUT's virtual devices. This test shall be run after all tests for the execution of the particular service have been run with a passing result.

Configuration requirements: Configure the IUT so that the receipt of a particular service request by a particular virtual device will cause some indication that is visible to the tester.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU
2. CHECK (verify that the indication occurs)

10.8.4.3 Route Global Broadcast from a Remote Device to Virtual Devices

Purpose: To verify that the IUT properly forwards global broadcast messages that originate on a remote network to its virtual devices. This test shall be skipped if the IUT's virtual devices are not capable of executing any application services that may be broadcast.

The tester shall select one of the following test options, depending on what services are supported by the virtual devices in the IUT.

A. Who-Is Test Option

Select one virtual device in the IUT that supports the execution of Who-Is requests.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 3,
   SADR = D3D,
   Hop Count = 254,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Am-Request,
   'I Am Device Identifier' = (the Device object instance number of the virtual device),
   'Max APDU Length Accepted' = (any valid value),
   'Segmentation Supported' = (any valid value),
   'Vendor Identifier' = (any valid value)

   | (DA = R2-3,
   SA = IUT,
   DNET = 3,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Am-Request,
   'I Am Device Identifier' = (the Device object instance number of the virtual device),
   'Max APDU Length Accepted' = (any valid value),
   'Segmentation Supported' = (any valid value),
   'Vendor Identifier' = (any valid value))

   | (DA = R2-3,
   SA = IUT,
   DNET = 3,
   DADR = (the MAC address of D3D),
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),

I-Am-Request,
'I Am Device Identifier' = (the Device object instance number of the virtual device),
'Max APDU Length Accepted' = (any valid value),
'Segmentation Supported' = (any valid value),
'Vendor Identifier' = (any valid value))

B. Who-Has Test Option

Select one virtual device in the IUT that supports the execution of Who-Has requests.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 3,
   SADR = D3D,
   Hop Count = 254,
   Who-Has-Request,
   'Device Instance Range Low Limit' = (the Device object instance number of the virtual device),
   'Device Instance Range High Limit' = (the Device object instance number of the virtual device),
   'Object Identifier' = (any object identifier of an object in the virtual device)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Have-Request,
   'Device Identifier' = (the Device object instance number of the virtual device),
   'Object Identifier' = (the object identifier specified in step 1),
   'Object Name' = (any valid value)
   | (DA = R2-3,
   SA = IUT,
   DNET = 3,
   DLEN = 0,
   SNET = 1,
   SADR = (the MAC address of the virtual device),
   Hop Count = (any integer x: 1 < x < 255),
   I-Have-Request,
   'Device Identifier' = (the Device object instance number of the virtual device),
   'Object Identifier' = (the object identifier specified in step 1),
   'Object Name' = (any valid value))

C. Generic Test Option

This test option is intended for IUTs whose virtual devices do not support the execution of the Who-Has or Who-Is services. This option uses a service that may be broadcast and that is supported by one of the IUT's virtual devices. This test shall be run after all tests for the execution of the particular service have been run with a passing result.

Configuration requirements: Configure the IUT so that the receipt of a particular service request by a particular virtual device will cause some indication that is visible to the tester.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = R2-3,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 3,
   SADR = D3D,
   Hop Count = 254,
   BACnet-Unconfirmed-Request-PDU
2. CHECK (verify that the indication occurs)

10.8.4.4 Route Remote Broadcast from a Local Device to Virtual Devices

Purpose: To verify that the IUT properly forwards remote broadcast messages originating on a local network to its virtual devices.

Test Concept: Execute test 10.8.4.2 (Route Global Broadcast from a Local Device to Virtual Devices) modified as follows. Instead of transmitting a global broadcast message, the TD shall transmit a remote broadcast message from a device on Network 2 directed to Network 1.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 1,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU
2. Continue with Step 2 from test 10.8.4.2

This test shall be skipped if the IUT's virtual devices are not capable of executing any application services that may be broadcast.

10.8.4.5 Route Remote Broadcast from a Remote Device to Virtual Devices

Purpose: To verify that the IUT properly forwards remote broadcast messages that originate on a remote network to its virtual devices.

Test Concept: Execute test 10.8.4.3 (Route Global Broadcast from a Remote Device to Virtual Devices) modified as follows. Instead of transmitting a global broadcast message, the TD shall transmit a remote broadcast message from a device on Network 3 directed to Network 1.

1. TRANSMIT
   DA = IUT,
   SA = R2-3,
   DNET = 1,
   DLEN = 0,
   SNET = 3,
   SADR = D3D,
   Hop Count = 254,
   BACnet-Unconfirmed-Request-PDU
2. Continue with Step 2 from test 10.8.4.3

This test shall be skipped if the IUT's virtual devices are not capable of executing any application services that may be broadcast.

10.8.4.6 Route Global Broadcast Message from a Virtual Device

Purpose: To verify that the IUT can route a global broadcast message from one of its virtual devices to the local (physical) network.

Test Concept: Make one of the virtual devices generate a remote broadcast, and verify that it is correctly formulated. This test shall be skipped if none of the IUT's virtual devices can transmit a global broadcast message.

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   SNET = 1,
   SADR = (MAC address of a virtual device on Network 1),
   Hop Count = (any integer x: $1 < x < 255$),
   BACnet-Unconfirmed-Request-PDU

10.8.4.7 Route Remote Broadcast Message from a Virtual Device to a Local Network

Purpose: To verify that the IUT can route a remote broadcast message from a virtual device to a local physical network.

Test Concept: Make one of the virtual devices generate a remote broadcast directed to the non-virtual network that the IUT is connected to, and verify that it is correctly formulated. This test shall be skipped if none of the IUT's virtual devices can issue a remote broadcast message.

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message to the local network of the IUT)
2. RECEIVE
   DA = LOCAL BROADCAST,
   SA = IUT,
   SNET = 1,
   SADR = (MAC address of a virtual device on Network 1),
   BACnet-Unconfirmed-Request-PDU

10.8.4.8 Route Remote Broadcast Message from a Virtual Device to a Remote Network

Purpose: To verify that the IUT can route a remote broadcast message from a virtual device to a remote network.

Test Concept: The IUT shall be configured such that its routing table only contains entries for the directly connected networks (physical and virtual). One of the virtual devices is made to send a remote broadcast message to Network 3.

Configuration Requirements: The IUT shall be configured such that its routing table only contains entries for the directly connected networks (physical and virtual). This test shall be skipped if none of the IUT's virtual devices can issue a remote broadcast message.

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message to the network 3)
2. RECEIVE
   DA = R2-3,
   SA = IUT,

DNET = 3,
DLEN = 0,
SNET = 1,
SADR = (MAC address of a virtual device on Network 1),
Hop Count = (any integer x: 1 < x < 255),
BACnet-Unconfirmed-Request-PDU
| (DA = LOCAL_BROADCAST,
SA = IUT,
DNET = 3,
DLEN = 0,
SNET = 1,
SADR = (MAC address of a virtual device on Network 1),
Hop Count = (any integer x: 1 < x < 255),
BACnet-Unconfirmed-Request-PDU)

Notes to Tester: During the test, the TD shall answer any requests that the IUT generates while attempting to locate the route to network 3.

10.8.5 Hop Count Protection

Purpose: To verify that the IUT will discard a message if the Hop Count becomes zero.

BACnet Reference Clause: 6.5.4.

Configuration Requirements: The IUT shall be configured with routing tables indicating that Network 1 is directly connected to Port 1 and that Network 2 is directly connected to Port 2 as shown in Figure 10-1. The routing table shall contain no other entries.

Test Steps:

1. TRANSMIT,
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1D,
   SNET = 3,
   SADR = D3D,
   Hop Count = 1,
   ReadProperty-Request,
   'Object Identifier' = (any object identifier),
   'Property Identifier' = (any property of the specified object)
2. WAIT **Internal Processing Fail Time** * 2
3. CHECK (verify that the IUT did not transmit a response for the Read Property)
4. TRANSMIT,
   DA = LOCAL BROADCAST,
   SA = TD,
   DNET = 1,
   DLEN = 0,
   SNET = 3,
   SADR = D3D,
   Hop Count = 1,
   Who-Is-Request,
   'Device Instance Range Low Limit' = VD1A
   'Device Instance Range High Limit' = VD1A
5. WAIT **Internal Processing Fail Time** * 2

6. CHECK (verify that the IUT did not transmit an I-Am for VD1A)

10.8.6 Network Layer Priority

Purpose: To verify that the IUT can process messages with all network priorities.

Test Concept: Apply the test in Clause 10.1.2 to any virtual device behind the IUT.

10.8.7 Multiple Devices on a Single Virtual Network

Note: If only one virtual device may be configured, then VD1B may be any Device Identifier and MAC address not equal to those of VD1A.

10.8.7.1 Who-Is Specifying Different Device Identifier

Purpose: To verify that the IUT correctly associates MAC addresses with individual virtual Device Identifiers when the IUT contains multiple devices.

Test Steps:

1. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   Who-Is-Request,
   'Device Instance Range Low Limit' = (Device object instance of VD1B)
   'Device Instance Range High Limit' = (Device object instance of VD1B)
2. CHECK (verify that the IUT does not transmit an I-Am-Request-PDU)

10.8.7.2 Who-Has Specifying Different Device Identifier

Purpose: To verify that the IUT correctly associates MAC addresses with individual virtual Device Identifiers when the IUT contains multiple devices.

Test Steps:

1. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   Who-Has-Request,
   'Object Identifier' = (Device object identifier of VD1B)
2. CHECK (verify that the IUT does not transmit an I-Have-Request-PDU)

10.8.7.3 Read of Object Not Contained by Virtual Device

Purpose: To verify that the IUT will respond with an error for a read of an object contained in a different virtual device than the one addressed.

Test Steps:

1. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (Device object identifier of VD1B),
   'Property Identifier' = Object_Identifier
2. RECEIVE
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   BACnet-Error-PDU,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT
3. TRANSMIT
   DA = IUT,
   SA = TD,
   DNET = 1,
   DADR = VD1A,
   Hop Count = 255,
   ReadProperty-Request,
   'Object Identifier' = (any object in virtual device VD1B that does not also exist in VD1A),
   'Property Identifier' = (any property of the specified object)
4. RECEIVE
   DA = TD,
   SA = IUT,
   SNET = 1,
   SADR = VD1A,
   BACnet-Error-PDU,
   Error Class = OBJECT,
   Error Code = UNKNOWN_OBJECT

Notes to Tester: If all of the virtual devices contain the same set of object instances, then steps 3 and 4 shall be skipped.

135.1-2011*m*-3. Replace Time Master Tests.

Rationale

Changes to time master requirements were made in 135-2004*b* and 135-2008*l*, and these tests were created to address those changes.

[Replace **Clause 13.2** in its entirety, p. 511]

13.2 Time Master

The tests in this clause verify that an IUT can perform the function of a time master. To be a time master, a device is required to be capable of keeping time and of issuing TimeSynchronization and UTCTimeSynchronization service requests.

13.2.1 TimeSynchronization Recipients Test, Protocol_Revision < 7

Purpose: To verify that an IUT implemented to a Protocol_Revision less than 7 can perform the function of a time master.

Dependencies: None.

BACnet Reference Clause: 12.11.31 and 16.7.

Test Concept: The Time Master functionality requires that the device supports the Device object's Time_Synchronization_Recipients property. For these tests to be fully completed, the IUT's Time_Synchronization_Recipients property must list at least one valid recipient. Note that for Protocol_Revision < 7, the IUT is allowed to either send both TimeSynchronization and UTCTimeSynchronization requests to all recipients, or determine which service(s) is supported by the recipient and transmit accordingly. The tester should simply ignore any additional TimeSynchronization-Request messages to TD2, and ignore any additional UTCTimeSynchronization-Request messages to TD1. The order in which the IUT transmits the requests is not important as long as at least one of each type of transmission is observed and as long as at least one transmission of TimeSynchronization-Request to TD1 and at least one transmission to UTCTimeSynchronization-Request to TD2 are observed.

Test Configuration: The tester shall configure two TD devices for use with this test. TD1 shall support only Time Synchronization while TD2 shall support only UTC_TimeSynchronization. If the IUT claims a Protocol_Revision of 7 or higher, this test shall be skipped.

Test Steps:

1 WRITE Time_Synchronization_Recipients = ([TD1, TD2])
2. MAKE (the IUT issue time synchronization requests to all recipients)
3. RECEIVE UTCTimeSynchronization-Request
DESTINATION = TD2,
SOURCE = IUT,
'Time' = (the IUT's current UTC date and time)
4. RECEIVE TimeSynchronization-Request,
DESTINATION = TD1,
SOURCE = IUT,
'Time' = (the IUT's current local date and time)
5. WRITE Time_Synchronization_Recipients = BROADCAST
6. MAKE (the IUT issue time synchronization requests to all recipients)
7. RECEIVE UTCTimeSynchronization-Request,
DESTINATION = BROADCAST,
SOURCE = IUT,

'Time' = (the IUT's current UTC date and time)

8. RECEIVE TimeSynchronization-Request,
   DESTINATION = BROADCAST,
   SOURCE = IUT,
   'Time' = (the IUT's current local date and time)

Notes to Tester: The order in which the IUT transmits the requests is not important. Ignore an additional TimeSynchronization-Request message to TD2, if one is observed. Ignore an additional UTCTimeSynchronization-Request message to TD1, if one is observed. Test steps 5, 6, 7, and 8 shall be performed three times, using local broadcast, remote broadcast, and global broadcast forms of the recipient in step 5.

13.2.2 TimeSynchronization Recipients Test, Protocol_Revision • 7

Purpose: To verify that an IUT can issue TimeSynchronization requests to the recipients in the Time_Synchronization_Recipients property.

Dependencies: None.

BACnet Reference Clause: 12.11.23, 12.11.24, 12.11.31, and 16.7.

Test Concept: The Time Master functionality requires that the IUT be able to put the IUT's current date and time into a TimeSynchronization request. The IUT is not required to be in any particular time zone, nor is it required to know what time zone it is in, though it must know its own local date and time. The device must support the Time_Synchronization_Recipients property for this test to be performed. If the IUT claims a Protocol_Revision less than 7, this test shall be skipped.

Test Configuration: There are two devices, TD and OD, at two distinct addresses, both of which support TimeSynchronization execution.

Test Steps:

1. WRITE Time_Synchronization_Recipients = ([TD, OD])
2. MAKE (the IUT issue TimeSynchronization-Requests to all recipients)
3. REPEAT R = (Recipients in the IUT's Time_Synchronization_Recipients property) DO {
   RECEIVE TimeSynchronization-Request,
   DESTINATION = R,
   SOURCE = IUT,
   'Time' = (the IUT's current local date and time)
   }
4. WRITE Time_Synchronization_Recipients = (a valid non-empty list different from that used in step 1)
5. MAKE (the IUT issue TimeSynchronization_Requests to all recipients)
6. REPEAT R = (Recipients in the IUT's Time_Synchronization_Recipients property) DO {
   RECEIVE TimeSynchronization-Request,
   DESTINATION = R,
   SOURCE = IUT,
   'Time' = (the IUT's current local date and time)
   }

Notes to Tester: The order in which the IUT transmits the requests is not important. In test step 4, it is desirable to use a MAC address and directed or global BROADCAST for at least one BACnetRecipient.

13.2.3 UTC_TimeSynchronization_Recipients Test

Purpose: To verify that an IUT can initiate UTCTimeSynchronization requests to the recipients in the UTC_Time_Synchronization_Recipients property.

Dependencies: None.

BACnet Reference Clause: 12.11.47 and 16.8.

Test Concept: DM-UTC-A functionality requires only that the IUT be able to put the current UTC date and time into a UTCTimeSynchronization request. The IUT is not required to be in any particular time zone, nor is it required to know what time zone it is in or even to know its own local date and/or time. It could, for example, obtain the current UTC date and time dynamically from some external provider.

Test Configuration: There are two devices, TD and OD, at two distinct addresses, both of which support UTCTimeSynchronization execution.

Test Steps:

1. WRITE UTC_Time_Synchronization_Recipients = ([TD, OD])
2. MAKE (the IUT issue UTCTimeSynchronization-Requests to all recipients)
3. REPEAT R = (Recipients in the IUT's UTC_Time_Synchronization_Recipients property) DO {
   RECEIVE UTCTimeSynchronization-Request,
   DESTINATION = R,
   SOURCE = IUT,
   'Time' = (the IUT's current UTC date and time)
   }
4. WRITE UTC_Time_Synchronization_Recipients = (a valid non-empty list different from that used in step 1)
5. MAKE (the IUT issue UTCTime_Synchronization_Requests to all recipients)
6. REPEAT R = (Recipients in the IUT's UTC_Time_Synchronization_Recipients property) DO {
   RECEIVE UTC_TimeSynchronization-Request,
   DESTINATION = R,
   SOURCE = IUT,
   'Time' = (the IUT's current UTC date and time)
   }

Notes to Tester: The order in which the IUT transmits the requests is not important. In test step 4, it is desirable to use a MAC address and directed or global BROADCAST for at least one BACnetRecipient.

13.2.4 Time_Synchronization_Interval Test

Dependencies: None.

BACnet Reference Clause: 12.11.48.

Purpose: To verify that when Time_Synchronization_Interval is non-zero, the IUT initiates TimeSynchronization requests at the specified interval. The test also verifies that when the property is zero, the IUT does not initiate TimeSynchronization requests.

Test Concept: The Time_Synchronization_Recipients property is made to contain a single recipient. Time_Synchronization_Interval is set to a non-zero value, X1, and the interval between TimeSynchronization requests is measured. Then Time_Synchronization_Interval is set to zero, and it is verified that TimeSynchronization requests are not issued by monitoring network traffic for 2*X1 minutes. If the Time_Synchronization_Recipients property is not present, then this test shall be skipped.

Configuration Requirements: The Time_Synchronization_Recipients property is configured to contain a single recipient, the TD. Align_Intervals is set to FALSE. Time_Synchronization_Interval is set to zero.

Test Steps:

1. WRITE Time_Synchronization_Interval = X1
2. BEFORE X1+1 minutes
   RECEIVE TimeSynchronization-Request,
   'Time' = T1
3. BEFORE X1+1 minutes
   RECEIVE TimeSynchronization-Request,
   'Time' = T2
4. CHECK (T2 - T1 = X1 to a precision of ±1 minute)
5. WRITE Time_Synchronization_Interval = 0
6. WAIT 2 * X1 minutes
7. CHECK (that the IUT did not issue any more TimeSynchronization-Requests)

Notes to Tester: The tolerance of 1 minute in steps 3 and 4 is in accord with the granularity used in the language in 135-2010, Clauses 12.11.48 and 12.11.50

13.2.5 UTC_Time_Synchronization_Interval Test

Dependencies: None.

BACnet Reference Clause: 12.11.48.

Purpose: To verify that when Time_Synchronization_Interval is non-zero, the IUT initiates UTCTimeSynchronization requests at the specified interval. The test also verifies that when the property is zero, the IUT does not initiate TimeSynchronization requests.

Test Concept: The UTC_Time_Synchronization_Recipients property is made to contain a single recipient. Time_Synchronization_Interval is configured to a non-zero value, X1, and the interval between UTCTimeSynchronization requests is measured. Then Time_Synchronization_Interval is set to zero, and it is verified that UTCTimeSynchronization requests are not issued by monitoring network traffic for 2*X1 minutes. If the UTC_Time_Synchronization_Recipients property is not present, then this test shall be skipped.

Configuration Requirements: The UTC_Time_Synchronization_Recipients property is configured to contain a single recipient, the TD. Align_Intervals is set to FALSE. Time_Synchronization_Interval is set to zero.

Test Steps:

1. WRITE Time_Synchronization_Interval = X1
2. BEFORE X1+1 minutes
   RECEIVE UTCTimeSynchronization-Request,
   'Time' = T1
3. BEFORE X1+1 minutes
   RECEIVE UTCTimeSynchronization-Request,
   'Time' = T2
4. CHECK (T2 - T1 = X1 to a precision of ±1 minute)
5. WRITE Time_Synchronization_Interval = zero
6. WAIT 2 * X1 minutes
7. CHECK (that the IUT did not issue any more UTCTimeSynchronization-Requests)

Notes to Tester: The tolerance of 1 minute in steps 3 and 4 is in accord with the granularity used in the language in 135-2010, Clauses 12.11.48 and 12.11.50

13.2.6 Align_Intervals and Interval_Offset TimeSynchronization Test

Dependencies: TimeSynchronization Recipients Test, Protocol_Revision • 7,13.2.2.

BACnet Reference Clause: 12.11.49 and 12.11.50.

Purpose: To verify that when Align_Intervals is TRUE and Time_Synchronization_Interval is a factor of (i.e, it divides without a remainder) an hour or day, time synchronization requests are aligned to the hour or day.

Test Concept: The Interval_Offset is set to zero and Align_Intervals is set to TRUE so that alignment of Time_Synchronization_Requests to the clock will occur; this requires a Time_Synchronization_Interval that is greater than 1 minute and less than or equal to 60 minutes and also that the interval is an evenly divisible factor of 60 minutes or 1440 minutes (i.e., Time_Synchronization_Interval is 2, 3, 4, 5, 6, 10, 12, 15, 16, 18, 20, 24, 30, 32, 36, 45, or 48 minutes). For this test, select two such intervals, one for step 1 and one for step 4. Prefer two small values from the list that are acceptable to the IUT, as indicated in its EPICS. There is no need to run this test for long durations. A TimeSynchronization-Request is received and checked that it is aligned to the clock; then the Time_Synchronization_Interval is changed and verified. Finally, Interval_Offset is set to a non-zero value less than the Time_Synchronization_Interval and checked, and then to a value greater than Interval_Offset and checked.

Configuration Requirements: The Time_Synchronization_Recipients property is configured to contain a single recipient. Time_Synchronization_Interval and Interval_Offset are set to zero; Align_Intervals is set to TRUE. If the Time_Synchronization_Recipients property is not present, then this test shall be skipped.

Test Steps:

1. WRITE Time_Synchronization_Interval = (X1, one of 4, 5, 6, 10, or 12)
2. BEFORE 2 * X1 minutes
   RECEIVE TimeSynchronization-Request,
   'Time' = T1
3. CHECK (T1 'minutes' is a multiple of Time_Synchronization_Interval, ±1 minute)
4. WRITE Time_Synchronization_Interval = (X2, any of the values not chosen in step 1)
5. BEFORE 2 * X2 minutes
   RECEIVE TimeSynchronization-Request,
   'Time' = T2
6. CHECK (T2 'minutes' is a multiple of Time_Synchronization_Interval, ±1 minute)
7. WRITE Interval_Offset = (any value from 2 to Time_Synchronization_Interval-1)
8. BEFORE 2 * X2 minutes
   RECEIVE TimeSynchronization-Request,
   'Time' = T3
9. CHECK (T3 'minutes' modulo Time_Synchronization_Interval = Interval_Offset, ±1 minute)
10. WRITE Interval_Offset = (any value from Time_Synchronization_Interval+1 to (2 * Time_Synchronization_Interval)-1)
11. BEFORE 2 * X2 minutes
    RECEIVE TimeSynchronization-Request,
    'Time' = T4
12. CHECK (T4 'minutes' modulo Time_Synchronization_Interval = (Interval_Offset - Time_Synchronization_Interval), ±1 minute)

13.2.7 Align_Intervals and Interval_Offset UTCTimeSynchronization Test

Dependencies: UTC_Time_Synchronization_Recipients Test, 13.2.3.

BACnet Reference Clause: 12.11.49 and 12.11.50.

Purpose: To verify that when Align_Intervals is TRUE and Interval_Offset is a factor of (i.e, it divides without a remainder) an hour or day, UTCTimeSynchronization requests are aligned to the hour or day.

Test Concept: Interval_Offset is set to zero and Align_Intervals is set to TRUE so that alignment of UTCTime_Synchronization_Requests to the clock will occur; this requires a Time_Synchronization_Interval that is greater than 1 minute and less than or equal to 60 minutes and also that the interval is an evenly divisible factor of 60 minutes or 1440 minutes (i.e., Time_Synchronization_Interval is 2, 3, 4, 5, 6, 10, 12, 15, 16, 18, 20, 24, 30, 32, 36, 45, or 48 minutes). For this test, select two such intervals, one for step 1 and one for step 4. Prefer two small values from the list that are acceptable to the IUT, as indicated in its EPICS. There is no need to run this test for long durations. A UTCTimeSynchronization-Request is received and checked that it is aligned to the clock; then the Time_Synchronization_Interval is changed and verified. Finally, Interval_Offset is set to a non-zero value less than the Time_Synchronization_Interval and checked, and then to a value greater than the Interval_Offset and checked.

Configuration Requirements: The UTC_Time_Synchronization_Recipients property is configured to contain a single recipient. The Time_Synchronization_Interval and Interval_Offset are set to zero; Align_Intervals is set to TRUE. If the UTC_Time_Synchronization_Recipients property is not present, then this test shall be skipped.

Test Steps:

1. WRITE Time_Synchronization_Interval = (X1, one of 4, 5, 6, 10, or 12)
2. BEFORE 2 * X1 minutes
   RECEIVE UTCTimeSynchronization-Request,
   'Time' = T1
3. CHECK (T1 'minutes' is a multiple of Time_Synchronization_Interval, ±1 minute)
4. WRITE Time_Synchronization_Interval = (X2, any of the values not chosen in step 1)
5. BEFORE 2 * X2 minutes
   RECEIVE UTCTimeSynchronization-Request,
   'Time' = T2
6. CHECK (T2 'minutes' is a multiple of Time_Synchronization_Interval, ±1 minute)
7. WRITE Interval_Offset = (any value from 2 to Time_Synchronization_Interval-1)
8. BEFORE 2 * X2 minutes
   RECEIVE UTCTimeSynchronization-Request,
   'Time' = T3
9. CHECK (T3 'minutes' modulo Time_Synchronization_Interval = Interval_Offset, ±1 minute)
10. WRITE Interval_Offset = (any value from Time_Synchronization_Interval+1 to (2 * Time_Synchronization_Interval)-1)
11. BEFORE 2 * X2 minutes
    RECEIVE UTCTimeSynchronization-Request,
    'Time' = T4
12. CHECK (T4 'minutes' modulo Time_Synchronization_Interval = (Interval_Offset - Time_Synchronization_Interval), ±1 minute)

135.1-2011*m*-4. Add Backup and Restore Tests.

Rationale

This section adds tests for the Backup and Restore procedure. There are currently no tests for this functionality.

[Add new **Clause 13.8**, p. 516]

13.8 Backup and Restore Procedure Tests

13.8.1 Backup and Restore Execution Tests

The tests in this section verify that a device that can be backed up and restored has implemented the Backup and Restore procedure correctly.

13.8.1.1 Execution of Full Backup and Restore Procedure

Purpose: This test case verifies that the IUT can execute a full Backup and Restore procedure.

Test Concept: This test takes the IUT through a successful Backup and then a successful Restore procedure. The Database_Revision and Last_Restore_Time properties are noted before the procedure begins for later comparison. The IUT is then commanded to enter the Backup state; all the files are read, and the IUT is commanded to end the backup. If the Database_Revision property can be changed by means other than the restore procedure, it is modified and checked to ensure that it incremented correctly; then the IUT is commanded to enter the Restore state. If the file objects do not exist on the IUT, the TD will create them in the IUT. The files are then truncated to size 0, the file contents are written to the IUT, and the IUT is commanded to end the restore. The Database_Revision and Last_Restore_Time properties are checked to ensure that they incremented or advanced correctly.

For IUTs that use Stream Access when performing the AtomicReadFile and AtomicWriteFile services, a Maximum Requested Octet Count (MROC) and a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max_APDU_Length_Accepted and the IUT's maximum transmittable APDU length. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max_APDU_Length_Accepted.

Test Steps:

1. READ DR1 = Database_Revision
2. READ LRT1 = Last_Restore_Time
3. READ OL1 = Object_List
4. REPEAT X = (1 through length of OL1) DO {
   READ NAMES[X] = (OL1[X]), Object_Name
   }
5. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ BPT = Backup_Preparation_Time
   READ RPT = Restore_Preparation_Time
   READ RCT = Restore_Completion_Time
   VERIFY Backup_And_Restore_State = IDLE
6. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Password' = (any valid password)
7. RECEIVE BACnet-Simple-ACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT BPT
   READ BRSTATE = Backup_And_Restore_State
   READ CF = Configuration_Files

```
        WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
            WAIT 1 second
            READ BRSTATE = Backup_And_Restore_State
            IF CF is an empty list THEN
                READ CF = Configuration_Files
            IF CF is a non-empty list THEN
                READ X = (the file referenced by Configuration_Files[1]).Name
        }
        CHECK (BRSTATE = PERFORMING_A_BACKUP)
9.  READ CF = Configuration_Files
10. CHECK (CF is a non-empty array of BACnetObjectIdentifiers referring to File objects)
11. REPEAT X = (each entry in CF) DO {
        READ Y = X, File_Access_Method
        IF (Y = RECORD_ACCESS)
            WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
                    TRANSMIT AtomicReadFile-Request,
                            'Object Identifier' = X,
                            'File Start Record' = (the next unread record),
                            'Requested Record Count' = 1
                    RECEIVE AtomicReadFile-ACK,
                            'End Of File' = TRUE | FALSE,
                            'File Start Record' = Z,
                            'Requested Record Count' = 1
                            'Returned Data' = (File contents)
                    | Error-PDU -- only acceptable for the first record and only when there are no records in the file
                            'Error Class' =         SERVICES,
                            'Error Code' =          INVALID_FILE_START_POSITION
            }
        ELSE
            WHILE (the last read did not indicate 'End Of File') DO {
                    TRANSMIT AtomicReadFile-Request,
                        'Object Identifier' = X,
                        'File Start Position' = (the next unread octet),
                        'Requested Octet Count' = MROC
                    RECEIVE AtomicReadFile-ACK,
                        'End Of File' = TRUE | FALSE,
                        'File Start Position' = (the next unread octet)
                        'File Data' = (File contents of length MROC if 'End Of File' is FALSE
                                    or of length MROC or less if 'End Of File' is TRUE)
                    | Error-PDU -- only acceptable for the first record and only when there are no records in the file
                            'Error Class' =         SERVICES,
                            'Error Code' =          INVALID_FILE_START_POSITION
            }
    }
12. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = ENDBACKUP,
        'Password' = (any valid password)
13. RECEIVE BACnet-Simple-ACK-PDU
14. VERIFY System_Status ! = BACKUP_IN_PROGRESS
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
        VERIFY Backup_And_Restore_State = IDLE
16. IF (Database_Revision is changeable) THEN
        MAKE (the configuration in the IUT different, such that the Database_Revision property increments)
        VERIFY Database_Revision <> DR1
        READ DR2 = Database_Revision
        CHECK (DR1 <> DR2)
```

17. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = STARTRESTORE,
    'Password' = (any valid password)
18. RECEIVE BACnet-Simple-ACK-PDU
19. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    READ BRSTATE = Backup_And_Restore_State
    WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
        WAIT 1 second
        READ BRSTATE = Backup_And_Restore_State
    }
    CHECK (BRSTATE = PERFORMING_A_RESTORE)
20. READ OL2 = Object_List
21. REPEAT X = (entry in CF) DO {
    IF (X is not in OL2)
        TRANSMIT CreateObject-Request
            'Object Identifier' = X
        RECEIVE CreateObject-ACK
            'Object Identifier' = X
    READ FS = X, File_Size
    IF (File_Size is not equal to the size of the backed up file)
        WRITE X, File_Size = 0
    IF (Y = RECORD_ACCESS)
        TRANSMIT AtomicWriteFile-Request
            'File Identifier' = X
            'File Start Record' = 0
            'Record Data' = (file content for first record obtained in step 11)
        RECEIVE AtomicWriteFile-ACK
            'File Start Record' = 0
        REPEAT REC = (each subsequent record in the backup of this file) {
            TRANSMIT AtomicWriteFile-Request
                'File Identifier' = X
                'File Start Record' = -1
                'Record Count' = 1
                'Record Data' = REC
            RECEIVE AtomicWriteFile-ACK
                'File Start Record' = (the record number)
        }
    ELSE
        REPEAT Z = (0 through the file size, in increments of MWDL) DO {
            TRANSMIT AtomicWriteFile-Request
                'File Identifier' = X
                'File Start Position' = Z
                'Record Data' = (file contents obtained from the backup, the number of octets
                    being the lesser of (file size - Z) and MWDL)
            RECEIVE AtomicWriteFile-ACK
                'File Start Position' = Z
        }
    }
22. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    VERIFY Backup_And_Restore_State = RESTORE_IN_PROGRESS
23. TRANSMIT ReinitializeDevice-Request,
        'Reinitialize State Of Device' = ENDRESTORE,
        'Password' = (any valid password)
24. RECEIVE BACnet-Simple-ACK-PDU
25. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN

```
        WAIT RCT
        VERIFY Backup_And_Restore_State = IDLE
26. READ DR3 = Database_Revision
27. CHECK (DR3 <> DR1)
28. IF (Database_Revision was changed in step 16) THEN
        CHECK (DR3 <> DR2)
29. VERIFY Last_Restore_Time > LRT1
30. READ OL3 = Object_List
31. CHECK (that OL1 and OL3 contain the same set of objects)
32. REPEAT X = (1 through length of OL1) DO {
        VERIFY (OL1[X]), Object_Name = NAMES[X]
    }
```

13.8.1.2 Attempting a Backup Procedure While Already Performing a Backup Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Backup Procedure from one client and then is commanded to start a Backup Procedure from a different client.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ BPT = Backup_Preparation_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT BPT
5. TRANSMIT
   SNET = (N, any remote network number),
   SADR = (M, any MAC address valid for the specified network),
   ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Property Identifier' = (any valid password),
6. RECEIVE
   DNET = N,
   DADR = M,
   BACnet-Error PDU,
   Error Class = DEVICE,
   Error Code = CONFIGURATION_IN_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ENDBACKUP,
   'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU

13.8.1.3 Attempting a Backup Procedure While Already Performing a Restore Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Backup Procedure when already performing a Restore Procedure.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN

READ RPT = Restore_Preparation_Time
READ RCT = Restore_Completion_Time

2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,
   Error Class = DEVICE,
   Error Code = CONFIGURATION_IN_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ABORTRESTORE,
   'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RCT

Note to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.

13.8.1.4 Attempting a Restore Procedure While Already Performing a Backup Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Backup Procedure.

Test Concept: The IUT is commanded to start a Restore Procedure from one client and then is commanded to start a Restore Procedure from a different client.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ BPT = Backup_Preparation_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT BPT
5. TRANSMIT
   SNET = (N, any remote network number),
   SADR = (M, any MAC address valid for the specified network),
   ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Property Identifier' = (any valid password),
6. RECEIVE
   DNET = N,
   DADR = M,
   BACnet-Error PDU,
   Error Class = DEVICE,
   Error Code = CONFIGURATION_IN_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ENDBACKUP,

'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU

13.8.1.5 Attempting a Restore Procedure While Already Performing a Restore Procedure

Purpose: To verify that the IUT correctly rejects a command to start a Restore Procedure when already performing a Restore Procedure.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ RPT = Restore_Preparation_Time
   READ RCT = Restore_Completion_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Property Identifier' = (any valid password)
3. RECEIVE Simple-ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Property Identifier' = (any valid password),
6. RECEIVE BACnet-Error PDU,
   Error Class = DEVICE,
   Error Code = CONFIGURATION_IN_PROGRESS
7. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ABORTRESTORE,
   'Property Identifier' = (any valid password)
8. RECEIVE Simple-ACK-PDU
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RCT

Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.

13.8.1.6 Ending Backup and Restore Procedures via Timeout

Purpose: This test case verifies that the IUT will end Backup and Restore procedures after not receiving any messages related to the backup or restore for longer than Backup_Failure_Timeout and that the Backup_Failure_Timeout property is writeable.

Test Steps:

1. WRITE Backup_Failure_Timeout = (A value T1 greater than Backup_Preparation_Timeout)
2. VERIFY Backup_Failure_Timeout = T1
3. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ BPT = Backup_Preparation_Time
4. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Property Identifier' = (any valid password)
5. RECEIVE Simple-ACK-PDU
6. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT BPT
   READ BRSTATE = Backup_And_Restore_State
   WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
   WAIT 1 second

READ BRSTATE = Backup_And_Restore_State
}
CHECK (BRSTATE = PERFORMING_A_BACKUP)

7. WAIT (T1 + 10 seconds)
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   VERIFY Backup_And_Restore_State = IDLE
9. VERIFY System_Status ! = BACKUP_IN_PROGRESS
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    READ RPT = Restore_Preparation_Time
    READ RCT = Restore_Completion_Time
11. TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = STARTRESTORE,
    'Password' = (any valid password)
12. RECEIVE BACnet-Simple ACK-PDU
13. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    READ BRSTATE = Backup_And_Restore_State
    WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
    WAIT 1 second
    READ BRSTATE = Backup_And_Restore_State
    }
    CHECK (BRSTATE = PERFORMING_A_RESTORE)
14. WAIT (40 seconds)
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    VERIFY Backup_And_Restore_State = IDLE
16. VERIFY System_Status ! = DOWNLOAD_IN_PROGRESS

Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.

13.8.1.7 Ending Backup and Restore Procedures via Abort

Purpose: This test case verifies that the IUT will leave the BACKUP_IN_PROGRESS and DOWNLOAD_IN_PROGRESS states upon a command to abort.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ BPT = Backup_Preparation_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT BPT
5. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ENDBACKUP,
   'Password' = (any valid password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   VERIFY Backup_And_Restore_State = IDLE
8. VERIFY System_Status != BACKUP_IN_PROGRESS
9. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ RPT = Restore_Preparation_Time
   READ RCT = Restore_Completion_Time

10. TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = STARTRESTORE,
    'Password' = (any valid password)
11. RECEIVE BACnet-Simple ACK-PDU
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
13. TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = ABORTRESTORE,
    'Password' = (any valid password)
14. RECEIVE BACnet-Simple ACK-PDU
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    VERIFY Backup_And_Restore_State = IDLE
16. VERIFY System_Status ! = DOWNLOAD_IN_PROGRESS

Notes to Tester: After an incomplete restore attempt, the IUT may revert to a default configuration or another state that is different from the IUT state when this test was started.

13.8.1.8 Attempting a Backup Procedure with an Invalid Password

Purpose: To verify the correct execution of the Backup procedure when an invalid password is provided. If the IUT cannot be made to deny a ReinitializeDevice <STARTBACKUP> service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTBACKUP,
   'Password' = (any invalid password)
2. RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE

13.8.1.9 Attempting a Restore Procedure with an Invalid Password

Purpose: To verify the correct execution of the Restore procedure when an invalid password is provided. If the IUT cannot be made to deny a ReinitializeDevice <STARTRESTORE > service request that does not contain a valid password, then this test shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Password' = (any invalid password)
2. RECEIVE BACnet-Error-PDU,
   Error Class = SECURITY,
   Error Code = PASSWORD_FAILURE

13.8.1.10 Starting and Ending a Backup Procedure when a Password is not Required

Purpose: This test case verifies that the IUT ignores the password. If the IUT cannot be made to accept a ReinitializeDevice service request that contains any or no password, then this test shall be omitted.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN

READ BPT = Backup_Preparation_Time
2. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = STARTBACKUP,
'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
WAIT BPT
5. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = ENDBACKUP,
'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
VERIFY Backup_And_Restore_State = IDLE
8. VERIFY System_Status ! = BACKUP_IN_PROGRESS

13.8.1.11 Starting and Ending a Restore Procedure when a Password is not Required

Purpose: This test case verifies that the IUT ignores the password. If the IUT cannot be made to accept a ReinitializeDevice service request that contains any or no password, then this test shall be omitted.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
READ RPT = Restore_Preparation_Time
READ RCT = Restore_Completion_Time
2. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = STARTRESTORE,
'Password' = (any non-zero length password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
WAIT RPT
5. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = ENDRESTORE,
'Password' = (any non-zero length password)
6. RECEIVE BACnet-Simple ACK-PDU
7. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
WAIT RCT
8. VERIFY System_Status ! = DOWNLOAD_IN_PROGRESS

13.8.1.12 System_Status during a Backup Procedure

Purpose: This test case verifies that the IUT correctly sets its System_Status during a Backup procedure. If the IUT does not change its operational behavior during a Backup Procedure, then this test shall be omitted.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
READ BPT = Backup_Preparation_Time
2. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = STARTBACKUP,
'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
WAIT BPT
5. VERIFY System_Status = BACKUP_IN_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,

'Reinitialized State of Device' = ENDBACKUP,
'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. WAIT a vendor specified period of time for the device to complete the backup operation
9. VERIFY System_Status != BACKUP_IN_PROGRESS

13.8.1.13 System_Status during a Restore Procedure

Purpose: This test case verifies that the IUT correctly sets its System_Status during a Restore procedure. If the IUT does not change its operational behavior during a Restore Procedure, this test shall be omitted.

Test Steps:

1. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   READ RPT = Restore_Preparation_Time
   READ RCT = Restore_Completion_Time
2. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = STARTRESTORE,
   'Password' = (any valid password)
3. RECEIVE BACnet-Simple ACK-PDU
4. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RPT
5. VERIFY System_Status = DOWNLOAD_IN_PROGRESS
6. TRANSMIT ReinitializeDevice-Request,
   'Reinitialized State of Device' = ABORTRESTORE,
   'Password' = (any valid password)
7. RECEIVE BACnet-Simple ACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
   WAIT RCT
9. VERIFY System_Status != DOWNLOAD_IN_PROGRESS

13.8.2 Backup and Restore Initiation Tests

The tests in this section verify that a device which can back up and restore other devices has implemented the initiation of the Backup and Restore procedure correctly.

The tests in this section rely on the TD being able to emulate a device that can be backed up and restored. The ability to configure the characteristics of the TD with respect to the Backup and Restore operations is important to ensure adequate testing of the IUT.

13.8.2.1 Initiate a Full Backup and Restore

Purpose: To verify that the IUT can perform a Backup and Restore on a BACnet server device.

Test Concept: The IUT is first made to initiate a Backup and then a Restore of the TD device. This test verifies that the IUT performs the Backup procedure correctly by comparing the resulting restored file with the original. The TD is made to respond appropriately such that the Backup and Restore procedures are completed normally. The final check can be accomplished using a file compare of the original files to the files restored or by comparing the network traffic during the backup to the network traffic during the restore. The number of files, the order of the files, and the file content should be the same. The test is to be executed multiple times with the TD configured with different sets of backup and restore characteristics.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. The TD is configured with some of the following characteristics:

Backup Characteristics:

1. The TD is configured to contain an APDU size that is smaller than the APDU size of the IUT. If the TD and the IUT support segmentation, the TD is configured to support a smaller window size than the IUT.
2. The TD is configured to contain a configuration file of size zero.
3. The TD is configured to contain some configuration files that are STREAM_ACCESS and some that are RECORD_ACCESS.
4. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
5. The TD is configured to require the same password for all of the reinitialize device requests.
6. The TD is configured to contain file names that would not be accepted by the OS which the IUT is running on.
7. The TD is configured with a Protocol_Revision < 10.
8. The TD is configured with a Protocol_Revision • 10. This is only used if the IUT claims Protocol_Revision • 10.

Restore Characteristics:

1. The TD is configured to support CreateObject service, and some of the configuration files exist while others do not.
2. The TD is configured such that some of the configuration file File objects exist, but the file size is different from that of the file to be restored.
3. The TD is configured to not support the CreateObject service.
4. The TD is configured to contain some configuration files that are STREAM_ACCESS and some that are RECORD_ACCESS.
5. The TD is configured to only allow access to File and Device objects during the Backup and Restore procedures. All other attempts shall result in an error from the TD.
6. The TD is configured to require the same password for all of the reinitialize device requests.
7. The TD is configured with a Protocol_Revision < 10.
8. The TD is configured with a Protocol_Revision • 10. This is only used if the IUT claims Protocol_Revision • 10.

Test Steps:

1. MAKE (IUT initiate a backup on the TD device)
2. WAIT (for backup to complete)
3. MAKE (changes required in TD to meet restore characteristics for this test)
4. MAKE (IUT initiate a restore on the TD device)
5. WAIT (for restore to complete)
6. CHECK (that the file content restored is the same as the file content that was backed up)

Notes to Tester: Other items to ensure were correct during execution of the test:

1. Verify the order the IUT read the configuration files was the same as the order returned by the Configuration_Files property.
2. Verify that any file with a File_Size of zero was restored.
3. Verify that each file read is in byte order if STREAM_ACCESS and in record order if RECORD_ACCESS.

13.8.2.2 Can Abort Backup if Error Received from TD

Purpose: To verify that the IUT can abort the Backup procedure when an error is received from the TD during a backup attempt.

Test Concept: The IUT is made to initiate a backup of the TD device. Before the backup is completed, the TD shall return an error. This error will be a BACnetAbortPDU, BACnetErrorPDU, or BACnetRejectPDU response to one of the confirmed requests initiated by the IUT during the backup process. When the IUT receives this error, it should abort the backup process.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. For this test to be performed, the TD shall be able to return an error for one of the confirmed requests initiated by the IUT during a backup.

Test Steps:

1. MAKE (IUT start backup on TD)
2. WAIT (until a tester chosen point in the backup)
3. WHILE (a ReiniutializeDevice-Request is not received) DO {
   IF (confirmed request received) THEN
   TRANSMIT
   BACnet-AbortPDU,
   AbortReason = (any value)
   | (BACnet-ErrorPDU,
   Error Class = OBJECT | PROPERTY,
   Error Code = (any of the error codes for an OBJECT or PROPERTY class)
   | (BACnet-Reject PDU,
   Reject Reason = (any value))
   }
4. RECEIVE ReinitializeDevice-Request,
   'Reinitialize State Of Device' = ENDBACKUP,
   'Password' = (any valid password)
5. TRANSMIT BACnet-Simple-ACK-PDU

Notes to Tester: An IUT that never stops the Backup procedure after several attempts by the TD to return an error shall fail this test.

13.8.2.3 Can Abort Restore if Error Received from TD

Purpose: To verify that the IUT can abort the restore procedure when an error is received from the TD during a restore attempt.

Test Concept: The IUT is made to initiate a restore of the TD device. Before the restore is completed, the TD returns an error. This error will be a BACnet Abort PDU or BACnet Reject PDU response to one of the confirmed requests initiated by the IUT during the backup process. When the IUT receives this error, it should abort the restore process.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. For this test to be performed, the TD shall be able to return an error for one of the confirmed requests initiated by the IUT during a restore.

Test Steps:

1. MAKE (IUT start restore on TD)
2. WAIT (until a tester selected time during the restore procedure)
3. WHILE (a ReiniutializeDevice-Request is not received) DO {
   IF (confirmed request received) THEN
   TRANSMIT
   BACnet-AbortPDU,
   AbortReason = (any value)
   | (BACnet-ErrorPDU,
   Error Class = OBJECT | PROPERTY,
   Error Code = (any of the error codes for an OBJECT or PROPERTY class))
   | (BACnet-Reject PDU,
   Reject Reason = (any value))
4. RECEIVE ReinitializeDevice-Request,
   'Reinitialize State Of Device' = ABORTRESTORE,

'Password' = (any valid password)
5. TRANSMIT BACnet-Simple-ACK-PDU

Notes to Tester: An IUT that never stops the restore procedure after several attempts by the TD to return an error shall fail this test.

13.8.2.5 Initiate an Abort Backup

Purpose: To verify that the IUT can abort the Backup procedure.

Test Concept: The IUT is made to initiate a backup of the TD device. Before the backup is completed, the IUT requests an abort of the backup.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device. A TD that takes a long time to backup is required.

Test Steps:

1. MAKE (IUT start backup on TD)
2. BEFORE (backup is complete)
   MAKE (IUT end backup on TD)
3. RECEIVE ReinitializeDevice-Request,
   'Reinitialize State Of Device' = ENDBACKUP,
   'Password' = (any valid password)
4. TRANSMIT BACnet-Simple-ACK-PDU

13.8.2.6 Initiate an Abort Restore

Purpose: To verify that the IUT can abort the restore procedure.

Test Concept: The IUT is made to initiate a restore of the TD device. Before the restore is completed, the IUT requests an abort of the restore procedure.

Configuration Requirements: The IUT is configured to already contain a device binding for the TD device.

Test Steps:

1. MAKE (IUT start restore on TD)
2. BEFORE (restore is complete)
   MAKE (IUT abort restore on TD)
3. RECEIVE ReinitializeDevice-Request,
   'Reinitialize State Of Device' = ABORTRESTORE,
   'Password' = (any valid password)
4. TRANSMIT BACnet-Simple-ACK-PDU

135.1-2011*m*-5. Add APDU Retry Test.

Rationale

Add a test to verify that the IUT's client application state machine correctly recognizes timeouts and performs retries.

[Add new **Clause 13.9**, p. 516]

13.9 Application State Machine Tests

13.9.1 APDU Retry and Timeout Test

Purpose: Verify that the IUT will re-send confirmed requests for which no response is received.

Test Concept: Make the IUT initiate a confirmed request to a non-responsive device, and verify that the request is retried after the APDU timeout.

Configuration Requirements: The network address of the TD is known to the IUT before the start of the test. The TD is configured to not respond to any requests after it has been found by the IUT and before the execution of the test. The IUT shall be configured with a non-zero value in its Number_Of_APDU_Retries property and a non-zero value in its APDU_Timeout property.

Test Steps: Steps 1-3 require that D1 does not answer the confirmed request.

1. MAKE (A condition that will cause the IUT to generate a confirmed request to D1)
2. RECEIVE
   SOURCE = IUT,
   DESTINATION = TD
   'Invoke Id' = (I, any valid value)
   BACnet-Confirmed-Request-PDU,
3. REPEAT (Number_Of_APDU_Retries) DO {
   WAIT (APDU_Timeout)
   RECEIVE Confirmed Request
   SOURCE = IUT,
   DESTINATION = TD,
   'Invoke ID' = I
   }
4. CHECK (that the IUT stopped its attempts using that invoke ID)

135.1-2011*m*-6. Add Workstation Schedule Interaction Tests.

Rationale

Add a set of tests to verify that IUTs which implement client-side scheduling functionality also implement the scheduling BIBBs correctly.

[Add new **Clause 13.10**, p. 516]

13.10 Workstation Scheduling Tests

Purpose: This group of tests verifies that the IUT is capable of viewing and modifying existing schedules.

Test Concept: This test consists of high-level MAKE and CHECK steps that are expected to be manually executed while monitoring BACnet communications using a BACnet network analyzer.

Test Configuration: The reference device shall be configured to indicate that it supports only the ReadProperty-Request and WriteProperty-Request services in the Protocol_Services_Supported property of its Device object. (Service clients that can use services more complex than ReadProperty-Request, such as ReadPropertyMultiple-Request, shall be able to adapt to a server device that does not support these more complex services.) The reference device is configured to contain Schedules and Calendars as specified by S1, S2, C1, and C2. The datatype of the 'value' portion of BACnetTimeValue can be varied to test scheduling of different datatypes, but all of the BACnetTimeValue elements shall be consistent within the same Schedule object, and the Present_Value and properties referenced by List_Of_Object_Property_References shall also be of the same datatype. The reference objects S1, S2, C1, and C2 represent the standard test data, but the tester is free to use additional test data for this test.

Note: The reference Schedule and Calendars contain some data that requires support for Protocol_Revision 4 or later. To convert the Schedule to conform to Protocol_Revision 3 or older, make the following changes:

1. Change month 13 to month 3 in BACnetSpecialEvent[5].
2. Remove the Schedule_Default property.
3. Change month 14 to month 2 in BACnetSpecialEvent[6].
4. Change dayOfWeek from 32 to 28 in BACnetSpecialEvent[6].
5. Change the last two CalendarEntries in the Date_List of Calendar C1 to remove the use of special values indicating all even months, all odd months, and the Last Day of the month.

Reference Schedule S1:

Effective_Period = ((January 5, 2007, Friday)-(December 31, 2009, Thursday))
Schedule_Default: NULL -- Applicable only to IUTs claiming support for Protocol_Revision 4 or greater.

-- Each day of the week contains a slightly different BACnetDailySchedule to test that the IUT assigns the correct
-- day of the week to the array indexes.
Weekly_Schedule = {
((00:00:00.00, <value1>), -- Monday
(01:00:00.00, <value2>),
(01:30:00.00, <value3>),
(08:00:00.00, <value4>),
(17:00:17.17, <value5>),
(23:59:59.99, <value6>)),
((00:00:00.00, <value1>), -- Tuesday
(02:00:00.00, <value2>),
(02:30:00.00, <value3>),
(08:00:00.00, <value4>),
(17:00:17.17, <value5>),

```
        (23:59:59.99, <value6>)),
        ((02:00:00.00, <value1>),  -- Wednesday
        (03:00:00.00, <value2>),   -- First 2 hours of Wednesday and Thursday are unspecified
        (03:30:00.00, <value3>),
        (08:00:00.00, <value4>),
        (17:00:17.17, <value5>),
        (21:59:59.99, NULL)),      -- Last 2 hours of Wednesday and Thursday are set to NULL
        ((02:00:00.00, <value1>),  -- Thursday
        (04:00:00.00, <value2>),
        (04:30:00.00, <value3>),
        (08:00:00.00, <value4>),
        (17:00:17.17, <value5>),
        (21:59:59.99, NULL)),
        ((00:00:00.00, <value1>),  -- Friday
        (05:00:00.00, <value2>),
        (05:30:00.00, <value3>),
        (08:00:00.00, <value4>),
        (17:00:17.17, <value5>),
        (23:59:59.99, <value6>)),
        ((00:00:00.00, NULL),      -- Saturday, most of the day is set to NULL
        (06:00:00.00, <value2>),
        (06:30:00.00, NULL),
        (08:00:00.00, <value4>),
        (12:00:00.00, <value5>),
        (13:00:00.00, NULL)),
        ((00:00:00.00, NULL),      -- Sunday, most of the day is set to NULL
        (07:00:00.00, <value2>),
        (07:30:00.00, NULL),
        (12:00:00.00, <value4>),
        (17:00:00.00, <value5>),
        (18:00:00.00, NULL)) }

Exception_Schedule = {
    -- 255 BACnetSpecialEvents, most with six entries in the listOfTimeValues.
    -- The first several BACnetSpecialEvents are designed to interact with the Effective_Period.

    (((January 1, 2007, Monday)-(January 2, 2007, Tuesday)),-- [1] calendarEntry, BACnetDateRange,
                                                            -- outside effective period
        ((01:00:00.00, <value1>),          -- first hour is unspecified
        (06:00:00.00, NULL),               -- 6:00-20:00 is relinquished
        (20:00:00.00, <value3>),
        (21:00:00.00, <value4>),
        (21:05:00.00,<value5>),
        (21:10:00.00,<value6>),
        (21:15:00.00,<value7>),
        (21:20:00.00,<value8>),
        (21:25:00.00,<value9>),
        (21:30:00.00,<value10>),
        (22:00:00.00, <value11>),
        (22:59:59.99, NULL)),              -- last hour is relinquished
        16),                               -- eventPriority
    (((January 3, 2007, Wednesday)-(January 6, 2007, Saturday)), -- [2] calendarEntry, BACnetDateRange,
                                                            -- period straddling start of effective period
        (<same as above>),                 -- listOfTimeValues same as in [1] above
        16),                               -- eventPriority
```

```
(((January 8, 2007, Monday)-(January 9, 2007, Tuesday)),-- [3] calendarEntry, BACnetDateRange,
                                                        -- period inside effective period
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

((January 11, 2007, Thursday),                          -- [4] calendarEntry, Date,
                                                        -- period inside effective period
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

-- The next section of BACnetSpecialEvents are designed to test variations
-- of the WeekNDay choice of BACnetCalendarEntry

((Odd months, days numbered 15-21, Monday),             -- [5] calendarEntry, BACnetWeekNDay,
                                                        -- period is 3rd Monday of each odd month
                                                        -- Odd months (13) only supported if
                                                        -- Protocol_Revision ≥ 4
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

((Even months, *, Sunday),                              -- [6] calendarEntry, BACnetWeekNDay,
                                                        -- every Sunday in every even month
                                                        -- Even months (14) only supported if
                                                        -- Protocol_Revision ≥ 4
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

((*, last 7 days of this month, Tuesday),               -- [7] calendarEntry, BACnetWeekNDay,
                                                        -- last Tuesday of each month
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

((February, *, Friday),                                 -- [8] calendarEntry, BACnetWeekNDay,
                                                        -- every Friday in February
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority

((*, days numbered 1-7, *),                             -- [9] calendarEntry, BACnetWeekNDay,
                                                        -- first 7 days of every month
    (<same as above>),              -- listOfTimeValues same as in [1] above
    16),                            -- eventPriority


-- The next section of BACnetSpecialEvents is designed to test CalendarReferences.
-- There are two references to (Calendar, 1) and one reference to (Calendar, 2))

((Calendar, 1),                     -- [10] calendarReference,
    ((08:15:00.00, <value1>),       -- listOfTimeValues
    (16:45:00.00, NULL)),
    15),                            -- eventPriority


-- The second CalendarReference to the same Calendar has a higher event priority
-- than the previous CalendarReference and a different listOfTimeValues.

((Calendar, 1),                     -- [11] calendarReference,
```

```
        ((12:00:00.00, <value2>),          -- listOfTimeValues
        (13:00:00.00, NULL)),
        14),                               -- eventPriority


        -- The following BACnetSpecialEvent references a different Calendar.

((Calendar, 2),                            -- [12] calendarReference,
        ((14:00:00.00, <value1>),          -- listOfTimeValues
        (15:00:00.00, NULL)),
        15),                               -- eventPriority


-- The next set of BACnetSpecialEvents are designed to test the partial day exception feature added in
-- Protocol_Revision 4.  All of the events are specified to occur between 8:00 AM and 5:00 PM, and the
-- higher the priority of the event, the shorter the duration of its period.  These data structures are allowed in a
-- device with Protocol_Revision less than 4, but the net effect of the schedule will be different because only
-- the BACnetSpecialEvent with the highest priority will be in effect on the specified date and all of the other
-- BACnetSpecialEvent records shall be ignored.

((January 12, 2007, Friday),                                -- [13] calendarEntry, Date,
                                                            -- Partial day exception, priority 15
        ((08:15:00.00, <value1>),          -- listOfTimeValues
        (16:45:00.00, NULL)),
        15),                               -- eventPriority

((January 12, 2007, Friday),                                -- [14] calendarEntry, Date,
                                                            -- Partial day exception, priority 14
        ((08:30:00.00, <value2>),          -- listOfTimeValues
        (16:30:00.00, NULL)),
        14),                               -- eventPriority

((January 12, 2007, Friday),                                -- [15] calendarEntry, Date,
                                                            -- Partial day exception, priority 13
        ((08:45:00.00, <value3>),          -- listOfTimeValues
        (16:15:00.00, NULL)),
        13),                               -- eventPriority

((January 12, 2007, Friday),                                -- [16] calendarEntry, Date,
                                                            -- Partial day exception, priority 12
        ((09:00:00.00, <value4>),          -- listOfTimeValues
        (16:00:00.00, NULL)),
        12),                               -- eventPriority

((January 12, 2007, Friday),                                -- [17] calendarEntry, Date,
                                                            -- Partial day exception, priority 11
        ((09:15:00.00, <value5>),          -- listOfTimeValues
        (15:45:00.00, NULL)),
        11),                               -- eventPriority

((January 12, 2007, Friday),                                -- [18] calendarEntry, Date,
                                                            -- Partial day exception, priority 10
        ((09:30:00.00, <value6>),          -- listOfTimeValues
        (15:30:00.00, NULL)),
        10),                               -- eventPriority
```

```
((January 12, 2007, Friday),                                    -- [19] calendarEntry, Date,
                                                                -- Partial day exception, priority 9
    ((09:45:00.00, <value7>),          -- listOfTimeValues
    (15:15:00.00, NULL)),
    9),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [20] calendarEntry, Date,
                                                                -- Partial day exception, priority 8
    ((10:00:00.00, <value8>),          -- listOfTimeValues
    (15:00:00.00, NULL)),
    8),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [21] calendarEntry, Date,
                                                                -- Partial day exception, priority 7
    ((10:15:00.00, <value9>),          -- listOfTimeValues
    (14:45:00.00, NULL)),
    7),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [22] calendarEntry, Date,
                                                                -- Partial day exception, priority 6
    ((10:30:00.00, <value10>),         -- listOfTimeValues
    (14:30:00.00, NULL)),
    6),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [23] calendarEntry, Date,
                                                                -- Partial day exception, priority 5
    ((10:45:00.00, <value11>),         -- listOfTimeValues
    (14:15:00.00, NULL)),
    5),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [24] calendarEntry, Date,
                                                                -- Partial day exception, priority 4
    ((11:00:00.00, <value12>),         -- listOfTimeValues
    (14:00:00.00, NULL)),
    4),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [25] calendarEntry, Date,
                                                                -- Partial day exception, priority 3
    ((11:15:00.00, <value13>),         -- listOfTimeValues
    (13:45:00.00, NULL)),
    3),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [26] calendarEntry, Date,
                                                                -- Partial day exception, priority 2
    ((11:30:00.00, <value14>),         -- listOfTimeValues
    (13:30:00.00, NULL)),
    2),                                -- eventPriority

((January 12, 2007, Friday),                                    -- [27] calendarEntry, Date,
                                                                -- Partial day exception, priority 1
    ((11:45:00.00, <value15>),         -- listOfTimeValues
    (13:15:00.00, NULL)),
    1),                                -- eventPriority
```

```
-- The next BACnetSpecialEvent is a test to see if the IUT can handle an event where the times
-- in the listOfTimeValue are not in chronological order.

((January 14, 2007, Sunday),                          -- [28] calendarEntry, Date,
     ((22:00:00.00, <value5>),        -- listOfTimeValues is not in chronological order
     (21:00:00.00, <value4>),
     (06:00:00.00, NULL),
     (20:00:00.00, <value3>),
     (01:00:00.00, <value1>),
     (22:59:59.99, NULL)),
     16),                             -- eventPriority


-- The next BACnetSpecialEvent is one that is deleted and then added back during the test steps.
-- The event times and values have no significance.

((January 20, 2007, Saturday),                        -- [29] calendarEntry, Date,
     ((00:00:00.00, NULL),            -- listOfTimeValues
     (13:00:00.00, <value1>),
     (15:00:00.00, <value2>)),
     16),                             -- eventPriority

-- The remaining BACnetSpecialEvents are "filler" to create a schedule large enough to test
-- for the SCH-VM-A and SCH-AVM-A capacity requirements.
-- Each BACnetSpecialEvent is a simple "date" choice of
-- BACnetCalendarEntry, with each date being different.  It is suggested that all of the dates in this
-- section be set to a different year than the previous BACnetSpecialEvents to avoid confusion.

((January 1, 2008, Tuesday),                          -- [30] calendarEntry, Date,
     ((00:00:00.00, NULL),            -- listOfTimeValues
     (11:00:00.00, <value2>),
     (11:30:00.00, NULL),
     (21:00:00.00, <value4>),
     (21:05:00.00, <value5>),
     (21:10:00.00, <value6>),
     (21:15:00.00, <value7>),
     (21:20:00.00, <value8>),
     (21:25:00.00, <value9>),
     (21:30:00.00, <value10>),
     (22:00:00.00, <value11>),
     (23:00:00.00, NULL)),
     16),                             -- eventPriority


-- [31 through 255] are the same as [30], with the date of the calendarEntry changed so that
-- each BACnetSpecialEvent applies to a different date, i.e., January 2, 2008, January 3, 2008, etc.

}
```

Reference Schedule S2:

Identical to schedule S1, except that the Exception_Schedule property is absent.

Reference Schedule S3:

A self-inconsistent schedule.

```
Effective_Period = ((January 5, 2007, Friday)-(December 31, 2009, Thursday))
Schedule_Default:  70.0 --- (REAL)

Weekly_Schedule = {
        ((00:00:00.00, 71.0),           -- Monday contains REAL values and a NULL
        (01:00:00.00, 72.0),
        (01:30:00.00, 73.0),
        (08:00:00.00, 74.0),
        (17:00:17.17, 75.0),
        (23:59:59.99, NULL)),
        ((00:00:00.00, 1),              -- Tuesday contains ENUMERATED values and a NULL
        (02:00:00.00, 2),
        (02:30:00.00, 3),
        (08:00:00.00, 4),
        (17:00:17.17, 5),
        (23:59:59.99, NULL)),
        ((02:00:00.00, 4294967201),     -- Wednesday contains Unsigned32 values and a NULL
        (03:00:00.00, 4294967202),
        (03:30:00.00, 4294967203),
        (08:00:00.00, 4294967204),
        (17:00:17.17, 4294967205),
        (21:59:59.99, NULL)),
        (),                             -- Thursday is an empty list
        (),                             -- Friday is an empty list
        (),                             -- Saturday is an empty list
        () }                            -- Sunday is an empty list

Exception_Schedule = {

    ((January 11, 2007, Thursday),      -- [1] calendarEntry, Date,
        ((01:00:00.00, 61.0),           -- Jan 11 contains REAL values
        (06:00:00.00, NULL),            -- 6:00-20:00 is relinquished
        (20:00:00.00, 62.0),
        (21:00:00.00, 63.0),
        (21:05:00.00, 64.0),
        (21:10:00.00, 65.0),
        (21:15:00.00, 66.0),
        (21:20:00.00, 67.0),
        (21:25:00.00, 68.0),
        (21:30:00.00, 69.0),
        (22:00:00.00, 70.0),
        (22:59:59.99, NULL)),           -- last hour is relinquished
        16),                            -- eventPriority

    ((January 12, 2007, Friday),        -- [2] calendarEntry, Date,
        ((01:00:00.00, 1),              -- Jan 12 contains ENUMERATED values
        (06:00:00.00, NULL),            -- 6:00-20:00 is relinquished
        (20:00:00.00, 2),
        (21:00:00.00, 3),
        (21:05:00.00, 4),
        (21:10:00.00, 5),
        (21:15:00.00, 6),
        (21:20:00.00, 7),
        (21:25:00.00, 8),
```

```
            (21:30:00.00, 9),
            (22:00:00.00, 10),
            (22:59:59.99, NULL)),          -- last hour is relinquished
            16),                           -- eventPriority

        ((January 13, 2007, Saturday),     -- [3] calendarEntry, Date,
            ((01:00:00.00, 4294967201),    -- Jan 13 contains Unsigned32 values
            (06:00:00.00, NULL),           -- 6:00-20:00 is relinquished
            (20:00:00.00, 4294967202),
            (21:00:00.00, 4294967203),
            (21:05:00.00, 4294967204),
            (21:10:00.00, 4294967205),
            (21:15:00.00, 4294967206),
            (21:20:00.00, 4294967207),
            (21:25:00.00, 4294967208),
            (21:30:00.00, 4294967209),
            (22:00:00.00, 4294967210),
            (22:59:59.99, NULL)),          -- last hour is relinquished
            16),                           -- eventPriority
}

-- The List_Of_Object_Property_References contains references to properties of differing datatypes.
List_Of_Object_Property_References = {
    ((Analog Output, Instance 1), Present_Value),   -- REAL
    ((Binary Output, Instance 1), Present_Value),    -- BACnetBinary PV
    ((Multi-state Output, Instance 1), Present_Value)   -- Unsigned
}
```

Reference Calendar C1:

```
-- The Date_List of this Calendar contains 32 CalendarEntries to test for the capacity requirements
-- specified by SCH-SVM-A.  All of the CalendarEntries are Date entries except for one DateRange.
-- The entries for 2009 & 2010 are "filler" to test the BIBB capacity limits and to test some
-- wildcard Date entries.
Object_Identifier = (Calendar, 1)
Object_Name = "2007 Holidays"
Description = "Holidays for 2007"
Date_List =                                -- 32 entries
            ((January 1, 2007, Monday),
            (April 6, 2007, Friday),
            (May 28, 2007, Monday),
            (July 4, 2007, Wednesday),
            (September 3, 2007, Monday),
            ((November 22, 2007, Thursday) – (November 23, 2007, Friday)),
            (December 24, 2007, Monday),
            (December 31, 2007, Monday),
            (October, 1, 2009, Thursday),
            (October, 2, 2009, Friday),
            (October, 3, 2009, Saturday),
            (October, 4, 2009, Sunday),
            (October, 5, 2009, Monday),
            (October, 6, 2009, Tuesday),
            (October, 7, 2009, Wednesday),
            (October, 8, 2009, Thursday),
            (October, 9, 2009, Friday),
```

(October, 10, 2009, Saturday),
Date: October, 11, 2009, Sunday
Date: October, 12, 2009, Monday
Date: October, 13, 2009, Tuesday
Date: October, 14, 2009, Wednesday
Date: October, 15, 2009, Thursday
Date: October, 16, 2009, Friday
Date: October, 17, 2009, Saturday
Date: October, 18, 2009, Sunday
Date: October, 19, 2009, Monday
Date: (3rd of every month in 2010)
{year: 110,
month: X'FF,
day of month: 3,
day of week: X'FF}
Date: (Every Sunday in January 2010)
{year: 110,
month: 1,
day of month: X'FF',
day of week: 7}
Date: (Leap year day, February 29, every year that it occurs)
{year: X'FF',
month: 2,
day of month: 29,
day of week: X'FF'}
Date: (The last day of every odd month in 2010, only supported if Protocol_Revision ≥ 4)
{year: 110,
month: 13,
day of month: 32,
day of week: X'FF'}
Date: (The 30th of every even month in 2010, only supported if Protocol_Revision ≥ 4)
{year: 110,
month: 14,
day of month: 30,
day of week: X'FF'}

Reference Calendar C2:

Object_Identifier: (Calendar, 2)
Object_Name: "Pax Romanus"
Description: "Dates of Roman significance, for the most part…"
Date_List (18 entries):
WeekNDay:
month: X'FF'
weekOfMonth: 1
dayOfWeek: 1
WeekNDay:
month: 3
weekOfMonth: 3
dayOfWeek: 1
WeekNDay:
month: 5
weekOfMonth: 3
dayOfWeek: 1
WeekNDay:

```
        month: 7
        weekOfMonth: 3
        dayOfWeek: 1
    WeekNDay:
        month: 10
        weekOfMonth: 3
        dayOfWeek: 1
    WeekNDay:
        month: 1
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 2
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 4
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 6
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 8
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 9
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 11
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 12
        weekOfMonth: 2
        dayOfWeek: 6
    WeekNDay:
        month: 4
        weekOfMonth: X'FF'
        dayOfWeek: 1
    WeekNDay:
        month: 12
        weekOfMonth: 3
        dayOfWeek: X'FF'
    WeekNDay:
        month:  X'FF'
        weekOfMonth: X'FF'
        dayOfWeek: 3
    WeekNDay:
        month: X'FF'
        weekOfMonth: 6
        dayOfWeek: X'FF'
    WeekNDay:
```

month: 11
weekOfMonth: X'FF'
dayOfWeek: X'FF'
WeekNDay:
month: 13
weekOfMonth: X'FF'
dayOfWeek: X'FF'
WeekNDay:
month: 14
weekOfMonth: X'FF'
dayOfWeek: X'FF'

13.10.1 Read and Present a Weekly_Schedule

Purpose: Demonstrate that the IUT reads and presents the Weekly_Schedule property of a Schedule object.

Test Configuration: A reference device contains Schedule object S1.

Test Steps:

1. MAKE (the IUT read and present the data represented by the Weekly_Schedule of S1)
2. CHECK (Did the IUT properly present the data represented by the Weekly_Schedule?)

13.10.2 Modify a Weekly_Schedule

This clause is used to verify that the IUT allows the user to modify any Weekly_Schedule located within a server device and does so appropriately.

13.10.2.1 Modify a Weekly_Schedule by Changing the Time of a BACnetTimeValue

Purpose: Demonstrate that the IUT can modify a Weekly_Schedule by changing the Time of a BACnetTimeValue that already exists in the schedule.

Test Configuration: A reference device contains Schedule object S1.

Test Steps:

1. MAKE (the IUT modify the Weekly_Schedule of S1 by changing the time of a BACnetTimeValue without changing the value)
2. CHECK (Did the IUT write the change to the Weekly_Schedule correctly?)

Notes to Tester: For example, modify the Friday 5:30:00.00 entry to Friday 5:45:00.00 with the same value, <value3>.

13.10.2.2 Modify a Weekly_Schedule by Changing the Value of a BACnetTimeValue

Purpose: Demonstrate that the IUT can modify a Weekly_Schedule by changing the Value of a BACnetTimeValue that already exists in the schedule.

Test Configuration: A reference device contains Schedule object S1.

Test Steps:

1. MAKE (the IUT modify the Weekly_Schedule of S1 by changing the value of a BACnetTimeValue without changing the time)
2. CHECK (Did the IUT write the change to the Weekly_Schedule correctly?)

13.10.2.3 Modify a Weekly_Schedule by Deleting a BACnetTimeValue

Purpose: Demonstrate that the IUT can modify a Weekly_Schedule by deleting an existing BACnetTimeValue.

Test Configuration: A reference device contains Schedule object S1.

Test Steps:

1. MAKE (the IUT delete a BACnetTimeValue from one of the days in the Weekly_Schedule of S1)
2. CHECK (Did the IUT write the modified Weekly_Schedule correctly?)

13.10.2.4 Modify a Weekly_Schedule by Adding a BACnetTimeValue

Purpose: Demonstrate that the IUT can modify a Weekly_Schedule by adding a new BACnetTimeValue to the schedule.

Test Configuration: A reference device contains Schedule object S1.

Test Steps:

1. MAKE (the IUT **add** a BACnetTimeValue to one of the days in the Weekly_Schedule of S1)
2. CHECK (Did the IUT write the modified Weekly_Schedule correctly?)

Notes to Tester: For example, add an entry of Friday 5:30:00.00 with a value of NULL.

13.10.3 Read and Present a Complex Schedule

Purpose: Demonstrate that the IUT reads and presents a complex reference schedule intended to test the capacity limits of the IUT as well as support for all choices of data structures.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT read and present the data represented by S1, C1, and C2)
2. CHECK (Did the IUT properly present the data represented by the reference objects, including all aspects of the properties?)

13.10.4 Modify an Exception_Schedule

This clause is used to verify that the IUT allows the user to modify any Exception_Schedule located within a server device and does so appropriately.

13.10.4.1 Modify an Exception_Schedule by Changing the Time of a BACnetTimeValue in the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by changing the time of a BACnetTimeValue pair in the BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT modify the time of a BACnetTimeValue of a calendarEntry in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, modify the January 11, 2007, calendarEntry such that an event at 20:00:00.00 is changed to occur at 19:00:00.00.

13.10.4.2 Modify an Exception_Schedule by Changing the Value of a BACnetTimeValue in the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by changing the value of a BACnetTimeValue pair in the BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT modify the value of a BACnetTimeValue of a calendarEntry in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, modify the January 11, 2007, calendarEntry such that an event at 22:00:00.00 is set for a value of NULL instead of <value11>.

13.10.4.3 Modify an Exception_Schedule by Deleting a BACnetTimeValue from the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by deleting a BACnetTimeValue pair in the BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT delete a BACnetTimeValue from the listOfTimeValues of a BACnetSpecialEvent with period of choice calendarEntry in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, delete the 21:00:00.00 event from the January 11, 2007, calendarEntry of the Exception_Schedule of S1.

13.10.4.4 Modify an Exception_Schedule by Adding a BACnetTimeValue to the listOfTimeValues of a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetTimeValue pair to the BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (The IUT add a BACnetTimeValue to the listOfTimeValues of a BACnetSpecialEvent with period of choice calendarEntry in the Exception_Schedule of S1.)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.5 Modify an Exception_Schedule by Changing the eventPriority of a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by changing the eventPriority of the BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT modify the eventPriority of a BACnetSpecialEvent with period of choice calendarEntry)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, try changing the priority of some of the BACnetSpecialEvents on January 12, 2007.

13.10.4.6 Modify an Exception_Schedule by Deleting a BACnetSpecialEvent with Period of Choice calendarEntry

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by deleting a BACnetSpecialEvent of the schedule with a period of choice calendarEntry.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT delete an entire BACnetSpecialEvent with period of choice calendarEntry from the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.7 Modify an Exception_Schedule by Adding a BACnetSpecialEvent with Period of Choice calendarEntry of choice Date

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetSpecialEvent of the schedule with the calendarEntry of type Date.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT add an entire BACnetSpecialEvent with period of choice calendarEntry of choice Date to the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, try different Date representations that leave one or more of each of the following fields unspecified: year, month, dayOfMonth, dayOfWeek**.** If the IUT supports devices with Protocol_Revision ≥ 4, try using the special values for month 13 (all odd) and 14 (all even) and the special value of 32 (last day) for the dayOfMonth.

13.10.4.8 Modify an Exception_Schedule by Adding a BACnetSpecialEvent with Period of Choice calendarEntry of Choice DateRange

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetSpecialEvent of the schedule with the calendarEntry of type DateRange.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT add an entire BACnetSpecialEvent with period of choice calendarEntry of choice DateRange to the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: It is not required that the IUT be capable of creating DateRanges with wildcard fields with the exception of unspecified dates (all fields equal to 255 in either one or both of the date fields of the BACnetDateRange).

13.10.4.9 Modify an Exception_Schedule by Adding a BACnetSpecialEvent with Period of Choice calendarEntry of Choice WeekNDay

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetSpecialEvent of the schedule with the calendarEntry of type WeekNDay.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT add an entire BACnetSpecialEvent with period of choice calendarEntry of choice WeekNDay to the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, try different WeekNDay representations that leave one or more of each of the following fields unspecified: month, weekOfMonth, dayOfWeek. If the IUT supports devices with Protocol_Revision ≥ 4, try using the special values for month 13 (all odd) and 14 (all even).

13.10.4.10 Modify an Exception_Schedule by Adding a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetSpecialEvent of the schedule with the calendarEntry of type calendarReference.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT add an entire BACnetSpecialEvent with period of choice calendarReference to the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

Notes to Tester: For example, add a BACnetSpecialEvent that is another CalendarReference to C2.

13.10.4.11 Modify an Exception_Schedule by Changing the Time of a BACnetTimeValue in the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by changing the time of a BACnetTimeValue of a BACnetSpecialEvent of the schedule.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT modify the time of a BACnetTimeValue of a calendarReference in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.12 Modify an Exception_Schedule by Changing the Value of a BACnetTimeValue in the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by changing the value of a BACnetTimeValue of a BACnetSpecialEvent of the schedule.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT modify the value of a BACnetTimeValue of a calendarReference in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.13 Modify an Exception_Schedule by Deleting a BACnetTimeValue from the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by deleting a BACnetTimeValue of a BACnetSpecialEvent of the schedule.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT delete a BACnetTimeValue from the listOfTimeValues of a calendarReference in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.14 Modify an Exception_Schedule by Adding a BACnetTimeValue to the listofTimeValues of a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by adding a BACnetTimeValue of a BACnetSpecialEvent of the schedule with period of choice calendarReference.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT add a BACnetTimeValue to the listOfTimeValues of a calendarReference in the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.4.15 Modify an Exception_Schedule by Deleting a BACnetSpecialEvent with Period of Choice calendarReference

Purpose: Demonstrate that the IUT can accept user input and modify the Exception_Schedule by deleting a BACnetSpecialEvent of the schedule with a period of choice calendarReference.

Test Configuration: A reference device contains reference objects S1, C1, and C2.

Test Steps:

1. MAKE (the IUT delete an entire BACnetSpecialEvent with period of choice calendarReference from the Exception_Schedule of S1)
2. CHECK (Did the IUT write the modified Exception_Schedule correctly?)

13.10.5 Modify a Calendar Object

This clause is used to verify that the IUT can modify a Calendar object in a server device appropriately.

13.10.5.1 Modify a Calendar by Deleting a BACnetCalendarEntry from the Date_List

Purpose: Demonstrate that the IUT can accept user input and use it to delete a BACnetCalendarEntry from the Date_List.

Test Configuration: A reference device contains reference object C1.

Test Steps:

1. MAKE (the IUT delete a BACnetCalendarEntry from the Date_List of Calendar C1)
2. CHECK (Did the IUT write the modified Date_List correctly?)

13.10.5.2 Modify a Calendar by Adding a BACnetCalendarEntry of Choice Date to the Date_List

Purpose: Demonstrate that the IUT can accept user input and use it to add a BACnetCalendarEntry of choice Date to the Date_List.

Test Configuration: A reference device contains reference object C1.

Test Steps:

1. MAKE (the IUT add a BACnetCalendarEntry of type Date to the Date_List of C1)
2. CHECK (Did the IUT write the modified Date_List correctly?)

Notes to Tester: For example, try different Date representations that leave one or more of each of the following fields unspecified: year, month, dayOfMonth, dayOfWeek. If the IUT supports devices with Protocol_Revision ≥ 4, try using the special values for month 13 (all odd) and 14 (all even) and the special value of 32 (last day) for the dayOfMonth.

13.10.5.3 Modify a Calendar by Adding a BACnetCalendarEntry of Choice DateRange to the Date_List

Purpose: Demonstrate that the IUT can accept user input and use it to add a BACnetCalendarEntry of choice DateRange to the Date_List.

Test Configuration: A reference device contains reference object C1.

Test Steps:

1. MAKE (the IUT add a BACnetCalendarEntry of type DateRange to the Date_List of C1)
2. CHECK (Did the IUT write the modified Date_List correctly?)

Notes to Tester: It is not required that the IUT be capable of creating DateRanges with wildcard fields.

13.10.5.4 Modify a Calendar by Adding a BACnetCalendarEntry of Choice WeekNDay to the Date_List

Purpose: Demonstrate that the IUT can accept user input and use it to add a BACnetCalendarEntry of choice WeekNDay to the Date_List.

Test Configuration: A reference device contains reference object C1.

Test Steps:

1. MAKE (the IUT add a BACnetCalendarEntry of type WeekNDay to the Date_List of C1)
2. CHECK (Did the IUT write the modified Date_List correctly?)

Notes to Tester: For example, try different WeekNDay representations that leave one or more of each of the following fields unspecified: month, weekOfMonth, dayOfWeek. If the IUT supports devices with Protocol_Revision ≥ 4, try using the special month values 13 (all odd) and 14 (all even).

13.10.6 Modify a Self-inconsistent Schedule to be Consistent

Purpose: Demonstrate that the IUT can read a Schedule that is self-inconsistent with regard to the scheduled datatype, and modify it to be consistent. This capability is required in order to be able to fix Schedule objects that may be left in a self-inconsistent state if the process of changing the scheduled datatype of a Schedule object is interrupted.

Test Configuration: A reference device contains reference object S3.

Test Steps:

1. MAKE (the IUT modify reference schedule S3 so that the scheduled datatype is consistent)
2. CHECK (Did the IUT write a consistent schedule?)

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

1. All of the values in the BACnetTimeValue pairs within the Weekly_Schedule and the Exception_Schedule properties shall be of the same datatype or NULL.
2. The value of the Schedule_Default property shall be of the same datatype as the non-NULL values in the BACnetTimeValue pairs within the Weekly_Schedule and Exception_Schedule properties, or it shall be NULL.
3. All of the standard properties referenced in the List_Of_Object_Property_References shall be of the same datatype as the non-NULL values in the BACnetTimeValue pairs.

The IUT is not required to present schedule S3 while it is self-inconsistent, only that the IUT write out a consistent schedule after the modification. Ideally, the IUT shall involve the user in the process of modifying the schedule to be consistent, presenting the full data of the schedule and allowing the user to edit the data to correct the inconsistencies. If the IUT modifies the schedule automatically to make it consistent, it should at least notify the user that the schedule has been modified. It is not acceptable for the IUT to modify the schedule without any indication to the user that this was done.

13.10.7 Change the Datatype that a Schedule Object Schedules

Purpose: Verify that the IUT can alter the scheduled datatype of an existing Schedule object.

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

Test Steps:

1. MAKE (the IUT modify the reference schedule so that the scheduled datatype is different than the original scheduled datatype)
2. CHECK (Did the IUT write the modified properties correctly?)

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

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.