



International
Standard

ISO 16484-6

**Building automation and control
systems (BACS) —**

**Part 6:
Data communication
conformance testing**

*Systèmes d'automatisation et de gestion technique du
bâtiment —*

Partie 6: Essais de conformité de la communication de données

**Fifth edition
2024-12**

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2024



COPYRIGHT PROTECTED DOCUMENT

© ISO 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

CONTENTS

CLAUSE	PAGE
FOREWORD	vi
1. PURPOSE	1
2. SCOPE	1
3. DEFINITIONS	1
3.1 Terms Adopted from International Standardss	1
3.2 Abbreviations and Acronyms Used in the Standard	1
3.3 Common language used in tests	1
4. ELECTRONIC PICS FILE FORMAT	2
4.1 Character Encoding	2
4.2 Structure of EPICS Files	2
4.3 Character Strings	3
4.4 Notational Rules for Parameter Values	3
4.5 Sections of the EPICS File	5
5. EPICS CONSISTENCY TESTS	10
6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS	12
6.1 TCSL Components	12
6.2 TCSL Statements	13
6.3 Time Dependencies	18
6.4 BACnet References	19
6.5 TD Requirements	19
6.6 Test Execution Considerations	19
7. OBJECT SUPPORT TESTS	21
7.1 Read Support for Properties in the Test Database	21
7.2 Write Support for Properties in the Test Database	23
7.3 Object Functionality Tests	29
8. APPLICATION SERVICE INITIATION TESTS	332
8.1 AcknowledgeAlarm Service Initiation Tests	332
8.2 ConfirmedCOVNotification Service Initiation Tests	334
8.3 UnconfirmedCOVNotification Service Initiation Tests	348
8.4 ConfirmedEventNotification Service Initiation Tests	352
8.5 UnconfirmedEventNotification Service Initiation Tests	398
8.6 GetAlarmSummary Service Initiation Tests	422
8.7 GetEnrollmentSummary Service Initiation Tests	422
8.8 GetEventInformation Service Initiation Tests	424
8.9 LifeSafetyOperation Service Initiation Tests	425
8.10 SubscribeCOV Service Initiation Tests	426
8.11 SubscribeCOVProperty Service Initiation Tests	427
8.12 AtomicReadFile Service Initiation Tests	432
8.13 AtomicWriteFile Service Initiation Tests	432
8.14 AddListElement Service Initiation Tests	433
8.15 RemoveListElement Service Initiation Tests	433
8.16 CreateObject Service Initiation Tests	434
8.17 DeleteObject Service Initiation Tests	434
8.18 ReadProperty Service Initiation Tests	435
8.19 ReadPropertyConditional Service Initiation Tests	437
8.20 ReadPropertyMultiple Service Initiation Tests	437
8.21 ReadRange Service Initiation Tests	440
8.22 WriteProperty Service Initiation Tests	443
8.23 WritePropertyMultiple Service Initiation Tests	446
8.24 DeviceCommunicationControl Service Initiation Tests	448
8.25 ConfirmedPrivateTransfer Service Initiation Test	449
8.26 UnconfirmedPrivateTransfer Service Initiation Test	450
8.27 ReinitializeDevice Service Initiation Tests	450
8.28 ConfirmedTextMessage Service Initiation Tests	451
8.29 UnconfirmedTextMessage Service Initiation Tests	452
8.30 TimeSynchronization Service Initiation Tests	452
8.31 UTCTimeSynchronization Service Initiation Tests	452
8.32 Who-Has Service Initiation Tests	453

8.33	I-Have Service Initiation Tests	454
8.34	Who-Is Service Initiation Tests	454
8.35	I-Am Service Initiation Tests	455
8.36	VT-Open Service Initiation Tests	455
8.37	VT-Close Service Initiation Tests	456
8.38	VT-Data Service Initiation Tests	457
8.39	RequestKey Service Initiation Tests	459
8.40	Authenticate Service Initiation Tests	459
8.41	WriteGroup Service Initiation Tests	462
8.42	SubscribeCOVPropertyMultiple Service Initiation Tests	463
8.43	AuditLogQuery Initiation Tests	467
9.	APPLICATION SERVICE EXECUTION TESTS	468
9.1	AcknowledgeAlarm Service Execution Tests	468
9.2	ConfirmedCOVNotification Service Execution Tests	493
9.3	UnconfirmedCOVNotification Service Execution Tests	498
9.4	ConfirmedEventNotification Service Execution Tests	501
9.5	UnconfirmedEventNotification Service Execution Tests	505
9.6	GetAlarmSummary Service Execution Tests	507
9.7	GetEnrollmentSummary Service Execution Tests	508
9.8	GetEventInformation Service Execution Tests	511
9.9	LifeSafetyOperation Service Execution Test	513
9.10	SubscribeCOV Service Execution Tests	517
9.11	SubscribeCOVProperty Service Execution Tests	527
9.12	AtomicReadFile Service Execution Tests	538
9.13	AtomicWriteFile Service Execution Tests	544
9.14	AddListElement Service Execution Tests	553
9.15	RemoveListElement Service Execution Tests	556
9.16	CreateObject Service Execution Tests	557
9.17	DeleteObject Service Execution Tests	562
9.18	ReadProperty Service Execution Tests	563
9.19	ReadPropertyConditional Service Execution Tests	568
9.20	ReadPropertyMultiple Service Execution Tests	568
9.21	ReadRange Service Execution Tests	577
9.22	WriteProperty Service Execution Tests	589
9.23	WritePropertyMultiple Service Execution Tests	596
9.24	DeviceCommunicationControl Service Execution Test	611
9.25	ConfirmedPrivateTransfer Service Execution Tests	618
9.26	UnconfirmedPrivateTransfer Service Execution Tests	619
9.27	ReinitializeDevice Service Execution Tests	619
9.28	ConfirmedTextMessage Service Execution Tests	623
9.29	UnconfirmedTextMessage Service Execution Tests	624
9.30	TimeSynchronization Service Execution Tests	625
9.31	UTCTimeSynchronization Service Execution Tests	625
9.32	Who-Has Service Execution Tests	626
9.33	Who-Is Service Execution Tests	633
9.34	VT-Open Service Execution Tests	636
9.35	VT-Close Service Execution Tests	637
9.36	VT-Data Service Execution Tests	639
9.37	RequestKey Service Execution Test	639
9.38	Authenticate Service Execution Tests	641
9.39	General Testing of Service Execution	644
9.40	AuditLogQuery Service Execution Tests	645
9.41	WriteGroup Tests	647
9.42	SubscribeCOVPropertyMultiple Service Execution Tests	651
10.	NETWORK LAYER PROTOCOL TESTS	665
10.1	General Network Layer Tests	665
10.2	Router Functionality Tests	666
10.3	Half-Router Functionality Tests	690
10.4	B/IP PAD Tests	697
10.5	Initiating Network Layer Messages	699
10.6	Non-Router Functionality Tests	700

ISO 16484-6:2024(en)

10.7	Route Binding Tests	703
10.8	Virtual Routing Functionality Tests	707
11.	LOGICAL LINK LAYER PROTOCOL TESTS	726
11.1	UI Command and Response	726
11.2	XID Command and Response	726
11.3	TEST Command and Response	727
12.	DATA LINK LAYER PROTOCOLS TESTS	728
12.1	MS/TP State Machine Tests	728
12.2	PTP State Machine Tests	786
12.3	BACnet/IP Functionality Tests	818
12.4	BACnet/IPv6 Functionality Tests	849
12.5	Secure Connect Functionality Tests	863
13.	SPECIAL FUNCTIONALITY TESTS	912
13.1	Segmentation	912
13.2	Time Master	920
13.3	Character Sets	925
13.4	Malformed PDUs	925
13.5	Slave Proxy Tests	926
13.6	Automatic Network Mapping	928
13.7	Automatic Device Mapping	929
13.8	Backup and Restore Procedure Tests	929
13.9	Application State Machine Tests	943
13.10	Workstation Scheduling Tests	943
14.	Reporting Test Results	961
ANNEX A – EXAMPLE EPICS (INFORMATIVE)		962
HISTORY OF REVISIONS		977

FOREWORD

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 205, *Building environmental design*, in collaboration with the European Committee for Standardization (CEN) Technical Committee CEN/TC 247, *Building Automation, Controls and Building Management*, in accordance with the Agreement on technical cooperation between ISO and CEN (Vienna Agreement) and with the American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc. (ASHRAE).

This fifth edition cancels and replaces the fourth edition (ISO 16484-6:2020), which has been technically revised.

The main changes are as follows:

— See the detailed list of changes on pages 977 to 981.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Building automation and control systems (BACS) —

Part 6:

Data communication conformance testing

1. PURPOSE

To define a standard method for verifying that an implementation of the BACnet protocol provides each capability claimed in its Protocol Implementation Conformance Statement (PICS) in conformance with the BACnet standard.

2. SCOPE

This standard provides a comprehensive set of procedures for verifying the correct implementation of each capability claimed on a BACnet PICS including:

- (a) support of each claimed BACnet service, either as an initiator, executor, or both,
- (b) support of each claimed BACnet object-type, including both required properties and each claimed optional property,
- (c) support of the BACnet network layer protocol,
- (d) support of each claimed data link option, and
- (e) support of all claimed special functionality.

3. DEFINITIONS

All definitions from ANSI/ASHRAE Standard 135-2020 also apply to this addendum.

3.1 Terms Adopted from International Standards

local network: the network to which a BACnet device is directly connected.

remote network: a network that is accessible from a BACnet device only by passing through one or more routers.

test database: a database of BACnet functionality and objects created by reading the contents of an EPICS.

3.2 Abbreviations and Acronyms Used in the Standard

BNF	Backus-Naur Form syntax
EPICS	electronic protocol implementation conformance statement
IUT	implementation under test
TCSL	testing and conformance scripting language
TD	testing device
TPI	text protocol information

3.3 Common language used in tests

'any valid value': Any valid value refers to any value of the correct data type and within the vendor's range specified for the property this is applied to.

'any appropriate password': Any password that meets the Configuration Requirements specified in the test or test section. Passwords when required by the vendor are required to be no more than 20 characters.

'reset': Some tests require to reset the IUT. Reset includes power cycle via switch, power cycle via loss of power, and reinitializeDevice WARMSTART. As defined by the BACnet standard, "WARMSTART shall mean to reboot the device and start over, retaining all data and programs that would normally be retained during a brief power outage."

4. ELECTRONIC PICS FILE FORMAT

An electronic protocol implementation conformance statement (EPICS) file contains a BACnet protocol implementation conformance statement expressed in a standardized text form. EPICS files are machine and human readable representations of the implementation of BACnet objects and services within a given device. EPICS files shall use the extension ".TPI" (text protocol information) and contain normal editable text lines consisting of text character codes ending in carriage return/linefeed pairs (X'0D', X'0A').

EPICS files are used by software testing tools to conduct and interpret the results of tests defined in this standard. An EPICS file shall accompany any device tested according to the procedures of this standard.

4.1 Character Encoding

BACnet provides for a variety of possible character encodings. The character encodings in BACnet fall into three groups: octet streams, double octet streams and quad octet streams. Octet streams represent characters as single octet values. In some cases, such as Microsoft DBCS and JIS C 6226, certain octet values signal that the second octet which follows should be viewed along with the leading octet as a single value, thus extending the range to greater than 256 possible characters. In contrast, double octet streams view pairs of octets as representing single characters. The ISO 10646 UCS-2 encoding is an example. The first or leading octet of the pair is the most significant part of the value. Quad octet streams, such as ISO 10646 UCS-4, treat tuples of four octets at a time as single characters with the first or leading octet being the most significant.

To accommodate the various encodings that may be used with BACnet device descriptions, EPICS files begin with a header that serves both to identify the file as an EPICS file, and to identify the particular encoding used. The header begins with the string "PICS #" where # is replaced by a numeral representing the character set as shown in Table 4-1.

Table 4-1. Character Set Codes

code	character set
0	ANSI X3.4
1	Microsoft DBCS
2	JIS C 6226
3	ISO 10646 (UCS-4)
4	ISO 10646 (UCS-2)
5	ISO 8859-1

An octet stream format can be recognized by examining the first eight octets of the EPICS file. Using ANSI X3.4 encoding as an example these eight octets will contain: X'50' X'49' X'43' X'53' X'20' X'30' X'0D' X'0A'. This represents the text "PICS 0" followed by carriage return and linefeed.

A double octet stream format can be recognized by examining the first 16 octets of the EPICS file. Using ISO 10646 UCS-2 encoding as an example these 16 octets will contain:

X'00' X'50' X'00' X'49' X'00' X'43' X'00' X'53'
X'00' X'20' X'00' X'34' X'00' X'0D' X'00' X'0A'

This represents the text "PICS 4" followed by carriage return and linefeed.

A quad octet stream format can be recognized by examining the first 32 octets of the EPICS file. Using ISO 10646 UCS-4 as an example these 32 octets will contain:

X'00' X'00' X'00' X'50' X'00' X'00' X'00' X'49'
X'00' X'00' X'00' X'43' X'00' X'00' X'00' X'53'
X'00' X'00' X'00' X'20' X'00' X'00' X'00' X'33'
X'00' X'00' X'00' X'0D' X'00' X'00' X'00' X'0A'

This represents the text "PICS 3" followed by carriage return and linefeed.

4.2 Structure of EPICS Files

EPICS files consist of text lines ending in carriage return/linefeed pairs (X'0D', X'0A') encoded as octet, double octet or quad octet streams as defined in 4.1. In the rest of this standard, the term "character" will be used to mean one symbol

encoded as one, two, or four octets based on the character encoding used in the EPICS file header. For example, the character space may be encoded as X'20' or X'0020' or X'00000020'. In this standard all characters will be shown in their single octet form.

The special symbol ↵ is used in this Clause to signify the presence of a carriage return/linefeed pair (X'0D0A'). Except within character strings, the character codes tab (X'09'), space (X'20'), carriage return (X'0D') and linefeed (X'0A') shall be considered to be white space. Any sequence of 1 or more white space characters shall be equivalent to a single white space character. Except within a character string, a sequence of two dashes (X'2D') shall signify the beginning of a comment which shall end with the next carriage return/linefeed pair, i.e., the end of the line upon which the -- appears. Comments shall be considered to be white space, and may thus be inserted freely.

EPICS files shall have, as their first line following the header, the literal text:

BACnet Protocol Implementation Conformance Statement ↵

This text serves as a signature identifying the EPICS file format.

Lines that define the sections of the EPICS (see 4.5) and the particular implementation data for a given device follow the signature line.

The EPICS file ends with a line containing the following literal text:

End of BACnet Protocol Implementation Conformance Statement ↵

4.3 Character Strings

The occurrence of a double quote (X'22'), single quote (X'27') or accent grave (X'60') shall signify character strings. For double quotes, the end of the string shall be signified by the next occurrence of a double quote, or the end of the line. For single quote or accent grave, the end of the string shall be signified by the next occurrence of a single quote (X'27'), or the end of the line. Thus strings which need to include a single quote or accent grave as a literal character in the string shall use the double quote quoting method, while strings which need to include double quote shall use the single quote or accent grave quoting method.

4.4 Notational Rules for Parameter Values

Within each section, parameters may need to be expressed in one of several forms. The following rules govern the format for parameters:

- (a) key words are case insensitive so that X'41' through X'5A' are equivalent to X'61' through X'7A';
- (b) null values are shown by the string "NULL";
- (c) Boolean values are shown by the strings "T" or "TRUE" if the value is true, or "F" or "FALSE" if the value is false;
- (d) integer values are shown as strings of digits, possibly with a leading minus (-): 12345 or -111;
- (e) real values are shown with a decimal point, which may not be the first or last character: 1.23, 0.02, 1.0 but not .02;
- (f) octet strings are shown as pairs of hex digits enclosed in either single quotes (X'2D') or accent graves (X'60'), and preceded by the letter "X": X'001122';
- (g) character strings are represented as one or more characters enclosed in double, single or accent grave quotes as defined in 4.3: 'text' or 'text' or "text";
- (h) bitstrings are shown as a list, enclosed by curly brackets ({ } or X'7B' and X'7D'), of true and false values: {T,T,F} or {TRUE, TRUE, FALSE}. When the actual value of a bit does not matter, a question mark is used: {T,T,?};
- (i) enumerated values are represented as named, rather than numeric, values. Enumeration names are case insensitive so that X'41' through X'5A' are equivalent to X'61' through X'7A'. The underscore (X'5F') and dash (X'2D') are considered equivalent in enumeration names. Proprietary values are shown as a named text with no whitespace and ending in a non-negative decimal numeric. Each must start with the word "proprietary": Object_Type, proprietary-object-type-653;
- (j) dates are represented enclosed in parenthesis: (Monday, 24-January-1998). Any "wild card" or unspecified field is shown by an asterisk (X'2A'): (Monday, *-January-1998). The omission of day of week implies that the day is unspecified: (24-January-1998);
- (k) times are represented as hours, minutes, seconds, hundredths in the format hh:mm:ss.xx: 2:05:44.00, 16:54:59.99. Any "wild card" field is shown by an asterisk (X'2A'): 16:54:*.;*;
- (l) object identifiers are shown enclosed by parentheses, with commas separating the object type and the instance number: (analog-input, 56). Proprietary object types replace the object type enumeration with the word "proprietary" followed by the numeric value of the object type: (proprietary 700,1);

- (m) constructed data items are represented enclosed by curly brackets ({ } or X'7B' and X'7D'), with elements separated by commas. If an element is itself a constructed value, then that element shall be enclosed in curly brackets.

4.4.1 Complex Parameter Values

Some parameter values, notably property values for constructed or CHOICE types of encoded values, need to use a more complex notation to represent their values. This notation is tied to the ASN.1 encoding for those property values and may appear obscure out of context. These additional rules govern the presentation of those types of parameter values:

- (a) values which are a CHOICE of application-tagged values are represented by the value of the chosen item encoded as described in 4.4;
- (b) values which are a CHOICE of context-tagged values are represented by the context tag number enclosed in square brackets, followed by the representation of the value of the chosen item;
- (c) list values (ASN.1 "SEQUENCE OF") are represented enclosed in parenthesis, with the elements of the list separated by commas. If an element is itself a constructed value, then that element shall be enclosed in curly brackets;
- (d) array values are represented enclosed in curly brackets, with the elements of the array separated by commas. If an element is itself a constructed value, then that element shall be enclosed in curly brackets.

4.4.2 Specifying Limits on Parameter Values

Some properties may have restrictions on the range or resolution of their values. In order to correctly interpret the results of tests in which the value of a property is changed using WriteProperty, WritePropertyMultiple, or AddListElement then read back using ReadProperty or ReadPropertyMultiple, it is necessary to know what these restrictions are. The test database may contain restriction statements that define these constraints. The permissible restrictions and the datatypes they apply to are:

- (a) **minimum** - the minimum value for Unsigned, Integer, Real, or Double datatypes. The earliest date for the Date datatype;
- (b) **maximum** - the maximum value for Unsigned, Integer, Real, or Double datatypes. The latest date for the Date datatype;
- (c) **resolution** - the minimum guaranteed resolution for Real and Double datatypes. The minimum time resolution in seconds for the Time datatype;
- (d) **maximum length string** - the maximum length of a CharacterString or OctetString;
- (e) **maximum length list** - the maximum number of elements guaranteed to fit in a list;
- (f) **maximum length array** - the maximum number of elements in an array;
- (g) **allowed values** - a comma-delimited list of supported enumerations for an Enumerated datatype. A comma-delimited list of object types for properties that reference an external object identifier.

Restriction statements shall be listed within pointed brackets (< and >) following the default value. If there are multiple restrictions within a single set of angle brackets, then the restrictions shall be separated by a semicolon (;). A restriction statement consists of the restriction name followed by a colon (:) followed by the restriction value or, where appropriate, a comma-delimited list of possible values.

Here are some examples of property values with restriction statements as they could appear in the test database.

```
present-value: 13.4 <minimum: 0.0; maximum: 20.0; resolution: 0.1>
description: "this is a description" <maximum length string: 30>
units: milliamperes <allowed values: milliamperes, amperes>
object-property-reference: (analog input, 12) <allowed values: analog input, analog value>
```

The Units property is a special case, because changing the units can change the value of the Present_Value property as well as any restrictions on its value. Therefore, minimum, maximum, and resolution restrictions are only valid for the default value of the Units property.

It is possible to specify default restrictions for most datatypes as described in 4.5.8. Restriction statements in the test database override the default restrictions for the individual property that contains the restriction statement.

4.5 Sections of the EPICS File

Each section of the EPICS file begins with a section name followed by a colon (: or X'3A'). After the colon is a set of one or more parameters delimited by a set of curly braces ({ } or X'7B' X'7D').

The following symbols are used as placeholders to indicate the presence of parameter information:

- (a) the open box symbol inside quotation marks, "□", is used to indicate that a character string parameter shall be present;
- (b) the open box symbol with no quotation marks, □, is used to indicate that a parameter with a datatype other than a character string shall be present;
- (c) a question mark, ?, is used in the test database to indicate that the property is present but the value is unknown because it depends on hardware input or is being changed by an internal algorithm.

An example EPICS file may be found in Annex A.

4.5.1 General Information Sections

These sections provide general information about the BACnet device. The syntax for these sections is shown below.

```
Vendor Name: "□"↵
Product Name: "□"↵
Product Model Number: "□"↵
Product Description: "□"↵
```

4.5.2 Conformance Sections

These sections provide information about the BACnet functionality that the device claims to support.

4.5.2.1 BIBBs Supported

This section indicates which BIBBs are supported. The syntax is shown below. Each BIBB shall be listed, one per line between the curly braces. An empty list indicates that no BIBBs are supported.

```
BIBBs Supported: ↵
{↵
  □↵
}↵
```

The BIBBs may be any of those BIBBs described in Annex K. The format in the EPICS shall be to use the short acronym described in the title for each BIBB. For example: A device which supports 'Data Sharing - ReadProperty - B' should include 'DS-RP-B' in this section.

For example:

```
BIBBs Supported: ↵
{↵
  DS-RP-B↵
  DS-WP-B↵
  DS-RPM-B↵
  DM-DOB-B↵
  DM-DDB-B↵
  DM-DDB-A↵
}↵
```

4.5.3 Application Services Supported

This section indicates which standard application services are supported. The syntax is shown below. Each supported service shall be listed between curly braces one service per line, followed by the words "Initiate" or "Execute" to indicate whether the service can be initiated, executed, or both.

```
BACnet Standard Application Services Supported: ↵
{↵
```



```

❑ Initiate↵
❑ Execute↵
❑ Initiate Execute↵
}

```

The standard services may be any of the services listed in the Clause 21 production BACnetServicesSupported.

The format should match the title of each corresponding services section in the standard minus the text 'Service'. For example, if the device supports the AcknowledgeAlarm service, the text 'AcknowledgeAlarm' should be included in this section of the EPICS.

For example:

```

BACnet Standard Application Services Supported: ↵
{↵
Who-Is                Initiate Execute↵
I-Am                  Initiate Execute↵
Who-Has               Execute↵
I-Have                Initiate ↵
ReadProperty           Execute↵
ReadPropertyMultiple  Execute↵
WriteProperty          Execute↵
}

```

4.5.4 Object Types Supported

This section indicates which standard object types are supported. The syntax is shown below. Each supported object type shall be listed between curly braces one object type per line, optionally followed by the words "Createable", "Deleteable", or both to indicate that dynamic creation or deletion is supported.

```

Standard Object Types Supported: ↵
{↵
❑↵
❑ Createable↵
❑ Deleteable↵
❑ Createable Deleteable↵
}↵

```

The standard objects may be any of the objects listed in the Clause 21 production, BACnetObjectTypesSupported.

The format should be the title of each corresponding object section in the standard minus the text Object Type. For example, if the device supports the object access door, the text 'Access Door' should be included in this section.

For example:

```

BACnet Standard Application Services Supported: ↵
{↵
Analog Value Createable Deleteable↵
Analog Input↵
Device↵
}

```

4.5.5 Data Link Layer Options

This section indicates which standard data link layer options are supported. The syntax is shown below. Each supported data link layer type shall be listed between the curly braces one per line. MS/TP and Point-To-Point data links shall also specify supported baud rate(s).

```

Data Link Layer Option: ↵
{↵

```


ISO 8802-3, 10BASE5 ↵
 ISO 8802-3, 10BASE2 ↵
 ISO 8802-3, 10BASET ↵
 ISO 8802-3, fiber ↵
 ARCNET, coax star ↵
 ARCNET, coax bus ↵
 ARCNET, twisted pair star ↵
 ARCNET, twisted pair bus ↵
 ARCNET, fiber star ↵
 ARCNET, twisted pair, EIA-485, Baud rate(s): □ ↵
 MS/TP master. Baud rate(s): 9600, □ ↵
 MS/TP slave. Baud rate(s): 9600, □ ↵
 Point-To-Point. EIA 232, Baud rate(s): □ ↵
 Point-To-Point. Modem, Baud rate(s): □ ↵
 Point-To-Point. Modem, Autobaud range: □ to □ ↵
 BACnet/IP, 'DIX' Ethernet ↵
 BACnet/IP, Other ↵
 Other ↵
 } ↵

4.5.6 Character Sets

This section indicates which BACnet character sets are supported. The syntax is shown below. Each supported character set shall be listed one per line between the curly braces.

Character Sets Supported: ↵
 { ↵
 ANSI X3.4 ↵
 IBM/Microsoft DBCS ↵
 JIS C 6226 ↵
 ISO 8859-1 ↵
 ISO 10646 (UCS-4) ↵
 ISO 10646 (UCS2) ↵
 } ↵

4.5.7 Special Functionality

This section indicates which BACnet special functionalities are supported. The syntax is shown below. Each special functionality supported shall be listed one per line between the curly braces. The maximum APDU size and window sizes shall be specified as integers.

Special Functionality: ↵
 { ↵
 Maximum APDU size in octets: □ ↵
 Segmented Requests Supported, window size: □ ↵
 Segmented Responses Supported, window size: □ ↵
 Router ↵
 BACnet/IP BBMD ↵
 } ↵

4.5.8 Property Value Restrictions

This section defines default restrictions on the values of writable properties. Restrictions listed for a particular datatype apply to every writable property or component of a writable property of that datatype. The restriction may be overridden for a particular property by adding a new restriction specifically for that property in the test database section of the EPICS. See 4.4.2. Only those datatypes for which default restrictions are being defined should be listed, one datatype per line. An empty list indicates that no default restrictions apply.

Default Property Value Restrictions: ↵
 { ↵
 unsigned-integer: <minimum: □; maximum: □> ↵

```

signed-integer:    <minimum: □; maximum: □>↵
real:              <minimum: □; maximum: □; resolution: □>↵
double:           <minimum: □; maximum: □; resolution: □>↵
date:             <minimum: □; maximum: □>↵
octet-string:     <maximum length string: □>↵
character-string: <maximum length string: □>↵
list:             <maximum length list: □>↵
variable-length-array: <maximum length array: □>↵
}↵

```

4.5.9 Timers

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

Fail Times: ↵

```

{↵
    Notification Fail Time: □↵
    Internal Processing Fail Time: □↵
    Minimum ON/OFF Time: □↵
    Schedule Evaluation Fail Time: □↵
    External Command Fail Time: □↵
    Program Object State Change Fail Time: □↵
    Acknowledgement Fail Time: □↵
    Slave Proxy Confirm Interval: □↵
    Unconfirmed Response Fail Time: □↵
    Channel Write Fail Time: □↵
    Auto Negotiation Fail Time: □↵
    Activate Changes Fail Time: □↵
    Foreign Device Registration Fail Time: □↵
}↵

```

4.5.10 Test Database

The last section of the EPICS file defines the contents of the device's test database of objects and their properties. The syntax for this section is described below.

List of Objects in Test Device: ↵

```

{↵
    object1↵
    object2↵
    ...
    objectN↵
}↵

```

Each of the objects is defined by a collection of object property values contained within curly braces. The first property to appear within the curly braces shall always be the Object_Identifier which specifies the tuple of (object type, instance). The second property shall always be Object_Name and the third property shall always be Object_Type with a value matching the object type portion of the object-identifier tuple:

```

{
    object-identifier: (object-type, instance)
    object-name: "□"
    object-type: object-type
    other properties...
}

```

Definitions of nonstandard objects shall contain only the three properties required by the BACnet standard, as shown below:

```

{
    object-identifier: (proprietary □, instance)
    object-name: "□"
}

```

```
object-type: proprietary □  
}
```

Properties in the test database that are writable shall have a "W" following the property value, as shown in the example below:

```
{  
  object-identifier: (analog-value, 6)  
  object-name: "□"  
  object-type: analog-value  
  present-value: 23.4 W  
  other properties...  
}
```

Properties in the test database that are conditionally writable shall have a "C" following the property value, as shown in the example below. It is recommended that the governing mechanism be identified in a comment:

```
{  
  object-identifier: (analog-input, 6)  
  object-name: "□"  
  object-type: analog-input  
  present-value: 12.3 C      -- Writable when Out_Of_Service is TRUE  
  other properties...  
}
```

5. EPICS CONSISTENCY TESTS

These tests are static tests of the EPICS and do not involve interrogating the IUT. There are no Configuration Requirements or Test Step sections with TCSL in these tests because the tests are static tests of the EPICS and not tests of the IUT itself.

Each implementation shall be tested to ensure consistency among interrelated data elements. These tests shall include:

- (a) All object types required by the specified BIBBs shall be indicated as supported in the Standard Object Types Supported section of the EPICS.
- (b) A minimum of one instance of each object type required by the specified BIBBs shall be included in the test database.
- (c) The Protocol_Object_Types_Supported property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.
- (d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.
- (e) The Protocol_Services_Supported property of the Device object in the test database shall indicate support for each application service for which the supported BIBBs requires support for execution of the service.
- (f) The object types listed in the Standard Object Types Supported section of the EPICS shall have a one-to-one correspondence with object types listed in the Protocol_Object_Types_Supported property of the Device object contained in the test database. An object type is supported if it can be made to exist in the IUT's database.
- (g) For each object type listed in the Standard Object Types Supported* there shall be at least one object of that type in the test database. It is permissible for there to be no instance of the File object type if File objects are dynamically creatable and come into existence only temporarily during Backup and Restore.
* An object type is supported if it can be made to exist in the IUT's database.
- (h) There shall be a one-to-one correspondence between the objects listed in the Object_List property of the Device object and the objects included in the test database. The Object_List property and the test database shall both include all proprietary objects. Properties of proprietary objects that are not required by BACnet Clause 23.4.3 need not be included in the test database.
- (i) For each object included in the test database, all required properties for that object as defined in Clause 12 of BACnet shall be present. Standard properties which are not defined for the implemented Protocol_Revision shall not be present. In addition, if any of the properties supported for an object require the conditional presence of other properties, their presence shall be verified.
- (j) For each property that is required to be writable, or conditionally writable, that property shall be marked as writable or conditionally writable, in the EPICS.
- (k) The length of the Protocol_Services_Supported bitstring shall have the number of bits defined for BACnetProtocolServicesSupported for the IUT's declared protocol revision.
- (l) The length of the Protocol_Object_Types_Supported bitstring shall have the number of bits defined for BACnetObjectTypesSupported for the IUT's declared protocol revision.
- (m) For each object included in the test database, any properties that are deprecated or removed shall not appear after the Protocol_Revision in which the property was deprecated or removed.
- (n) If the Protocol_Revision property is present in the Device object and its value is greater than or equal to 14, the Property_List property of each object included in the test database shall have one entry for each property present, including non-standard properties with the exception of Object_Type, Object_Identifier, Object_Name, and Property_List.

- (o) If the Segmentation_Supported property in the Device object is SEGMENTED_BOTH or SEGMENTED_RECEIVE, then the value of the Max_Segments_Accepted property of the Device object shall be greater than 1.
- (p) For each property that is required to be read-only, that property shall not be marked as writable, or conditionality writable, in the EPICS.
- (q) For each property for which the standard limits the value range, the value for the property in the EPICS, if provided, shall be within the allowable range as defined in the property definition.
- (r) For each property which has a restricted value range defined in the EPICS, the restricted value range shall be within the allowable value range and contain the minimal value range as defined by the standard.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2024

6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

In order to shorten and clarify test descriptions a simple Testing and Conformance Scripting Language (TCSL) is used. Following the Backus-Naur Form (BNF) syntax for programming language grammars, the following symbols will be used:

<part>	language component names are enclosed in pointed brackets
::=	is defined as
<a> <c>	implicit concatenation
	pattern selection
()	required component
[]	optional component
()...	one required, may be repeated
[]...	optional, may be repeated
<d>...	component appears once, may be repeated
'?'	symbols are in single quotes
WOW	reserved words are upper case
--	indicates that the remaining characters in this line are a comment

Because TCSL is pseudo language that is not intended to be an implementation language for a TD, the rigorous forms of BNF are relaxed in some places and an English statement or phrase is used in their place.

In the tests defined using TCSL an exchange of messages exactly as prescribed constitutes a passing result unless some additional constraint is explicitly noted.

6.1 TCSL Components

6.1.1 Common Symbols and Characters

The following definitions are used to represent common symbols and characters.

<binary digit> ::=	'0' '1'
<decimal digit> ::=	'0' '1' '2' '3' '4' '5' '6' '7' '8' '9'
<hex digit> ::=	<decimal digit> 'A' 'B' 'C' 'D' 'E' 'F'
<single quote> ::=	(the single quote character)
<double quote> ::=	(the double quote character)
<conditional> ::=	'=' '<' '>' '<=' '>=' '~=' '<>'

The conditional '~=' is approximately equal to. The definition of approximate is dependant on the context of its use.

6.1.2 Integers

The following definitions are used to represent integers. These definitions match the syntax used in the BACnet standard.

<integer> ::=	<binary int> <decimal int> <hex int>
<binary int> ::=	B <single quote> <binary digit>... <single quote>
<decimal int> ::=	['-'] <unsigned> D <single quote> <decimal digit>... <single quote>
<hex int> ::=	X <single quote> <hex digit>... <single quote>
<unsigned> ::=	<decimal digit>...

6.1.3 Text Strings

Text strings representing the value of a property or service parameter may be upper, lower or mixed case and are enclosed in double quotes ("").

Examples: Object_Name = "CW_STEMP", Description = "AC1 Supply Temperature".

6.1.4 Enumerations

The value of a property or parameter that is an enumerated type is represented in all UPPER CASE letters without quotation.

Examples: TRUE, FALSE, RELIABLE, UNRELIABLE.

6.1.5 Property Identifiers

Property identifiers are represented by mixed case letters with each component word capitalized and joined with other component words, if present, by an underscore ().

Examples: Present Value, Reliability, Object Identifier, Object Type, and Vendor Name.

6.1.6 Service Parameters

The names of service parameters are enclosed in single quotes.

Examples: 'Return Read Access Specifications with Result', 'List of Read Access Results'.

6.1.7 Object Identifiers

The representation of object identifiers is defined as follows:

$$\langle \text{object identifier} \rangle ::= '(' \langle \text{object type} \rangle ',' \langle \text{instance number} \rangle ')'$$

Where:

<object type> ::= (one of the object types defined in BACnet Clause 12)

`<instance number> ::= <unsigned>`

Examples: (Analog Input, 1), (Device, 150)

6.2 TCSL Statements

Statements in TCSL fall into two categories, those that control the flow and order of tests and those that tell the TD to send data, receive data, or both.

$$\langle \text{statement} \rangle ::= \langle \text{simple statement} \rangle \mid \langle \text{compound statement} \rangle$$
$$\langle \text{compound statement} \rangle ::= \{ \langle \text{statement} \rangle \dots \}$$

A <compound statement> can be used anywhere a <simple statement> can be used.

```

<simple statement> ::= <if statement> | <repeat statement> | <error statement>
                    | <check statement> | <make statement>
                    | <transmit statement> | <receive statement>
                    | <write statement> | <verify statement> | <before statement>
                    | <read statement>

```

6.2.1 IF Statement

The IF statement is used to test for a condition and take an alternate flow of control based on the result of the test.

```
<if statement> ::= IF '(' <condition> ')' THEN <statement> [ ELSE <statement> ]
```

The <condition> is a simple English phrase describing a decision the TD must make. If the <condition> is true, the statement after the THEN keyword will execute. If the <condition> is false and the ELSE clause has been specified, the statement following the ELSE will execute. For example:

```
IF (it is raining outside) THEN
    VERIFY Present_Value = 1
ELSE
    VERIFY Present_Value = 2
```

6.2.2 REPEAT Statement

The REPEAT statement is used to iterate through a list of similar objects or values.

```
<repeat statement> ::= REPEAT <var> '=' '(' <list description> ')' DO <statement>
```

The <var> is some unique identifier that will take on each of the values in the <list description>. By convention it is usually the letter 'X', 'Y', or 'Z' and is specified as an uppercase character. The <list description> is a simple English phrase that describes the elements to iterate through. For example:

```
REPEAT X = (values specified by 6.3 appropriate to the object type) DO {
    WRITE Present_Value = X
    VERIFY Present_Value = X
}
```

6.2.3 ERROR Statement

Some TCSL steps may result in an error condition that is made visible by the ERROR statement.

```
<error statement> ::= ERROR [ <explanation string> ]
```

The optional <explanation string> is a text string provided to the operator of the TD for diagnostic purposes. For example:

```
ERROR "Retry count exceeded"
```

6.2.4 CHECK Statement

The CHECK statement is used when the operator must verify that some action by the TD has resulted in a change to the IUT that is not network visible.

```
<check statement> ::= CHECK '(' <condition> ')'
```

The operator is given an opportunity to notify the TD that an operation was or was not successful, the <condition> is a simple English phrase that describes what to check. If the operation was not successful, the test will fail. For example:

```
CHECK (Did the IUT reboot?)
```

6.2.5 MAKE Statement

The MAKE statement is used when the operator must perform some action to create a change in the IUT.

```
<make statement> ::= MAKE '(' <action> ')'
```

Where <action> is a text string describing the action that is to take place. For example:

```
MAKE (Out_Of_Service TRUE)
```

6.2.6 TRANSMIT Statement

The TRANSMIT statement is used to transmit a packet.

```
<transmit statement> ::= TRANSMIT <packet desc>
```

Where:

```
<packet desc> ::= [ <port> ',' ] [ <addressing> ',' ] ( <service specification> | <pdu specification> |
                                                         <string> )

<port> ::= PORT <port identifier>
<port identifier> ::= 'A' | 'B' ... 'Z'
<addressing> ::= ( <dst> | <src> | <dst> ',' <src> )
<src> ::= (SOURCE '=' <src parm value>) | <src parm list>
<src parm value> ::= TD | IUT | <device>
<src parm list> ::= (any src addr parameter name) '=' ( <src parm value> | <addr parm value>)
                                                         [ ',' <src parm list> ]
<dst> ::= (DESTINATION '=' <dst parm value>) | <dst parm list>
<dst parm value> ::= LOCAL BROADCAST | GLOBAL BROADCAST
                                                         | REMOTE BROADCAST <net>
                                                         | IUT | TD
                                                         | <device>
```


<dst parm list>::=	(any dst addr parameter name) '=' (<dst parm value> <addr parm value>) [',' <dst parm list>]
<addr parm value> ::=	(any media specific address description)
<device> ::=	'(device ' (device instance) ')' (a valid device description or variable)
<net> ::=	(a valid BACnet network number, description or variable)
<service specification> ::=	<BACnet service> [',' <pdu parm list>] [',' <service parm list>]
<BACnet service> ::=	(any BACnet service choice)
<service parm list> ::=	<service parameter> '=' <parameter value> [',' <service parm list>]
<service parameter> ::=	(parameter name specific to the BACnet service)
<pdu specification> ::=	<pdu type> [',' <pdu parm list>]
<pdu type> ::=	(any BACnet application, network, link, or MAC layer PDU type)
<pdu parm list> ::=	<pdu parameter> '=' <parameter value> [',' <pdu parm list>]
<pdu parameter> ::=	(any BACnet application, network, data link, or MAC layer PDU parameter)
<parameter value> ::=	(<atomic value> <parameter value list> <parameter cond value>)
<parameter value list> ::=	'(' <parameter value> [',' <parameter value>]... ')'
<parameter cond value> ::=	'(' IF <condition> THEN <parameter value> ELSE <parameter value> ')'
<string> ::=	<"> <ASCII Char> <">
<ASCII Char> ::=	(ANSI X3.4 character)

The SOURCE and DESTINATION parameters are used to briefly specify common combinations of NPDU, LPDU and MPDU parameter values. If no source addressing information is provided, then the source address shall be TD. If no destination addressing information is provided, then the destination address shall be IUT.

A list of <pdu parameter> = <parameter value> pairs indicate that the specified parameters shall convey the indicated values. The parameter values may be specified in any order.

A list of <service parameter> = <parameter value> pairs indicate that the specified parameters shall convey the indicated values. The parameter values may be specified in any order.

Example 1:

TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is

In this simple case, the Who-Is service does not have any mandatory parameters and the <pdu type> is known to be a BACnet-Unconfirmed-Request-PDU by definition. The DESTINATION implies parameter values in the NPDU, LPDU and MPDU layers. The following statement is identical, but more completely specified:

TRANSMIT
DA = LOCAL BROADCAST,
SA = TD,
DNET = GLOBAL BROADCAST,
BACnet-Unconfirmed-Request-PDU,
'Service Choice' = Who-Is

Example 2:

TRANSMIT ReadProperty-Request,
'Object Identifier' = (Analog Input,1),
'Property Identifier' = Present_Value

In this case a ReadProperty service request will be sent from the TD to the IUT with the specified service parameter values.

6.2.7 RECEIVE Statement

The RECEIVE procedure is used to define a message from the IUT.

<receive statement> ::= RECEIVE (<packet desc> | '(' <packet desc> ')' ['(' <packet desc> ')'] ...)

The <pdu specification> parameter is the same as used in the TRANSMIT statement. If unspecified the source addressing information defaults to IUT and destination addressing information defaults to TD. Note: When the destination addressing information refers to a device on a remote network, DA is allowed to be a local MAC broadcast.

Example: TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = any value selected by the TD,
'Monitored Object Identifier' = any object supporting COV notification,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = 300

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = the value from the previous subscription,
'Monitored Object' = the value from the previous subscription,
'Initiating Device Identifier' = IUT,
'Lifetime' = 300,
'List of Values' = values appropriate to the object type of the monitored object

6.2.8 WAIT Statement

The WAIT statement is used to pause the execution of the TD for some specified amount of time.

<wait statement> ::= WAIT <timer value>

Test Steps: The TD shall pause the amount of time specified by the <timer value> before proceeding to the next test step. The <timer value> shall be one of the timers specified in 6.3 of this standard, in ANSI/ASHRAE 135-2020, or as otherwise specified.

Example: WAIT **Internal Processing Fail Time**

6.2.9 WRITE Statement

The WRITE statement is used to modify the value of a specific property of an object.

<write statement> ::= WRITE [<object identifier> ','] <property identifier> '=' <property value>
[',' ARRAY INDEX '=' <index value>]
[',' PRIORITY '=' <write priority>]

This is a shortcut of the following statements:

1. TRANSMIT WriteProperty-Request,
'Object Identifier' = <object identifier>,
'Property Identifier' = <property identifier>,
'Property Value' = <property value>,
'Priority Array Index' = <array index>,
'Priority' = <write priority>
2. RECEIVE BACnet-SimpleACK-PDU

Note: In some tests <object identifier> is omitted from the description because it is clear from the context what object type should be written to and any instance of that object type would be acceptable. In cases where there may be some ambiguity this parameter will be explicitly specified.

If <index value> or <write priority> are omitted then the corresponding service parameter shall be omitted in the WriteProperty service request.

Example: WRITE (Analog Output, 1), Present_Value = 6.5, PRIORITY = 8

6.2.10 VERIFY Statement

<verify statement> ::= VERIFY [<object identifier> ','] <property identifier> <conditional> <property value>
[',' ARRAY INDEX '=' <array index>]

The verify procedure consists of the following steps:

1. WAIT **Internal Processing Fail Time**
2. TRANSMIT ReadProperty-Request,
 'Object Identifier' = <object identifier>,
 'Property Identifier' = <property identifier>,
 'Property Array Index' = <array index>
3. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = <object identifier>,
 'Property Identifier' = <property identifier>,
 'Property Array Index' = <array index>,
 'Property Value' = (any valid value x, where x <conditional> <property value> is TRUE subject to the resolution constraints of 4.4.2)

Example: WRITE (Analog Output, 1), Present_Value = 6.5, PRIORITY = 8
 VERIFY (Analog Output, 1), Present_Value = 6.5

6.2.11 BEFORE Statement

The BEFORE statement is used to test for the occurrence of an expected action before a timer expires.

<before statement> ::= BEFORE <timer> <statement>

The <timer value> shall be one of the timers specified in 6.3 of this standard, in ANSI/ASHRAE 135-2020, or as otherwise specified. If the action indicated by <statement> has not yet occurred when the timer expires the test fails. Otherwise this test step passes and the test continues. For example:

Example: BEFORE **Acknowledgment Fail Time** RECEIVE BACnet-SimpleACK-PDU

6.2.12 WHILE Statement

The WHILE statement is used to repeatedly perform a step or series of steps until some condition becomes FALSE.

<while statement> ::= WHILE '(' <condition> ')' DO <statement>

Example:

```
WHILE (IUT not initialized) DO {
    TRANSMIT Poll For Master
}
```

6.2.13 READ Statement

<read statement> ::= READ <variable> '=' [<object identifier> ','] <property identifier>
 [',' ARRAY INDEX '=' <array index>]

The read procedure consists of the following steps:

1. WAIT **Internal Processing Fail Time**
2. TRANSMIT ReadProperty-Request,
 'Object Identifier' = <object identifier>,
 'Property Identifier' = <property identifier>,
 'Property Array Index' = <array index>
3. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = <object identifier>,
 'Property Identifier' = <property identifier>,
 'Property Array Index' = <array index>,
 'Property Value' = (any valid value, <variable>)

Example: READ X = Priority_Array, ARRAY INDEX=10

6.2.14 Assignment Statement

The assignment statement is used to set the value of a TCSL variable

<assignment statement> := <variable> '=' '(' <value description> ')'

The <value description> is a simple English phrase describing the content that is to be placed into the variable. It may reference other variables already in use by the test. For example:

READ X = O1, Present_Value
 READ Y = O2, Present_Value
 MAX = (the larger of X and Y)

6.3 Time Dependencies

The BACnet standard does not define how long it should take for actions that result from service requests to become network visible. These time delays can reasonably be expected to vary from implementation to implementation. In a testing environment it is necessary to place a bound on these times in order to decide if a test has passed or failed. The timers defined in this clause are used for this purpose. The vendor shall provide the actual value of the timers for a particular implementation in the EPICS. See 4.5.9. The data type for these timers shall be Real.

6.3.1 Notification Fail Time

The **Notification Fail Time** is the elapsed time, in seconds, between when the conditions that constitute an event or change of value become externally observable and when a test is considered to have failed because the expected notification message has not been transmitted.

6.3.2 Internal Processing Fail Time

The **Internal Processing Fail Time** is the elapsed time, in seconds, between the receipt of a write to a BACnet property or some other event that changes the value of the property and when a test is considered to have failed because the property value has not been updated.

The **Internal Processing Fail Time** contained in the EPICS is the maximum value for all situations. In order to reduce test time, it is acceptable that a shorter **Internal Processing Fail Time** value be used on a per test basis where the shorter time will not negatively impact the test result.

6.3.3 Minimum ON/OFF Fail Time

The **Minimum ON/OFF Fail Time** is the maximum elapsed time, in seconds, between the expiration of a minimum on or minimum off timer and when the test is considered to have failed because the value at command priority level 6 is not as expected.

6.3.4 Schedule Evaluation Fail Time

The **Schedule Evaluation Fail Time** is the elapsed time, in seconds, between a change to a property defining a Calendar or a Schedule, or a change in the device's Local_Time, and when a test is considered to have failed because the Present_Value of the Calendar or Schedule does not reflect the correct state.

6.3.5 External Command Fail Time

The **External Command Fail Time** is the elapsed time, in seconds, between a change to the Present_Value of a Command object and when a test is considered to have failed because the first message associated with the newly commanded state has not been transmitted.

6.3.6 Program Object State Change Fail Time

The **Program Object State Change Fail Time** is the time, in seconds, between a write to the Program_Change property and when a test is considered to have failed because the expected program result was not observed.

6.3.7 Acknowledgment Fail Time

The **Acknowledgment Fail Time** is the elapsed time, in seconds, between when a BACnet confirmed service request is transmitted and the corresponding acknowledgment shall have been received.

6.3.8 Default Time Delay in Test Descriptions

For the test cases defined in this standard it is acceptable to have a time delay of up to **Internal Processing Fail Time** before a message specified by a RECEIVE statement is actually received unless a different timing constraint is explicitly stated.

6.3.9 Unconfirmed Response Fail Time

The **Unconfirmed Response Fail Time** is the elapsed time, in seconds, between when the IUT receives an unconfirmed request and when a test is considered to have failed because the expected unconfirmed request from the IUT has not been received. For example, this is the maximum amount of time that the IUT would take to initiate an I-Am request in response to a Who-Is request.

6.3.10 Channel Write Fail Time

The **Channel Write Fail Time** is the elapsed time, in seconds, between a change to the Present_Value of a Channel object and when a test is considered to have failed because the first write operation associated with the newly written value state has not been performed. If the Channel object has multiple target properties to write to, the time to write all of them would be less than or equal to the number of target properties times this value.

6.3.11 Auto-negotiation Fail Time

The Auto-negotiation Fail Time is the elapsed time, in seconds, between when auto-negotiation is requested and when a test is considered to have failed because the IUT has not completed auto-negotiation of link speed.

6.3.12 Activate Changes Fail Time

The **Activate Changes Fail Time** is the elapsed time, in seconds, that the IUT requires in order to complete activation of changes in a Network Port object, including any required reset of the device and when a test is considered to have failed because the IUT has not completed activation of the Network Port changes.

6.3.13 Foreign Device Registration Fail Time

The **Foreign Device Registration Fail Time** is the elapsed time, in seconds, between when the device sends a Register-Foreign-Device request and when the device considers the request to have failed due to having not received a BVLC-Result for the registration request.

6.3.14 Audit Notification Forwarder Fail Time

The Audit Notification Forwarder Fail Time is the elapsed time, in seconds, between when a forwarding audit log receives an audit notification and when a test is considered to have failed because the expected audit notification message has not been transmitted.

6.4 BACnet References

All references to BACnet clauses in this standard refer to ANSI/ASHRAE 135-2020, except as otherwise noted ("BACnet-2001" refers to ANSI/ASHRAE 135-2001).

6.5 TD Requirements

The tests defined in this test standard describe specific conversations between the TD and the IUT. While these test conversations take place, the IUT may initiate other conversations related to or unrelated to the current test. Unless otherwise noted in the test, for conversations initiated by the IUT that are not reflected in the test steps of the test, the TD shall generate a correct response consistent with the device capabilities and state that the TD is emulating.

6.6 Test Execution Considerations

There are some implicit test considerations which apply to all tests.

6.6.1 Value Comparisons

Value comparisons, such as those in conditionals and statements (6.1.1 and 6.2) should always take into account the resolution constraints of 4.4.2, such that if any value is written with a higher or finer granularity than the IUT supports, any and all subsequent comparisons to that written value shall not result in the test step failing once the constraints have been applied to the value from the testing device.

Any value from the IUT is compared to a value from the testing device, the comparison shall not fail as long as the values are identical once the resolution constraints are applied to the external value. Devices may either truncate or round values written/received at a finer resolution than they claim in 4.4.2. This should be consistent across all values of that datatype.

Floating point values may be stored in the IUT using a different precision than what the value is encoded with from the testing device. Comparisons to these types of values should always consider that the IUT can round or truncate them, and that the value may also lose precision when stored, although the IUT must always present the stored value using full precision on a subsequent read.

Example: An IUT has a property with type REAL which has a resolution of 0.5. If a TD writes 0.75, it is acceptable for the IUT to present, on a subsequent read, either 1.0 or 0.5 for the property value so long as it is consistent.

6.6.2 Functional Expectations

Because devices may store information in truncated or rounded format due to internal limitations, they may not behave exactly as all test cases prescribe. In these cases, it is acceptable for the IUT to present or act on information based on the granularity supported by the IUT. For example, an IUT may be configured with a schedule which begins at 12:01:01, but since the device only supports granularity of 1 minute, the scheduled behavior may begin at 12:01:00 or 12:02:00.

6.6.3 Complex Datatypes

Since all complex datatypes can be broken down into one or more primitive datatypes, value comparisons and functional expectations from the above sections shall apply to each of the components that comprise a complex datatype. When there are relationships present within the fields of a complex datatype, the IUT shall continue to meet the functional expectations between all fields of complex datatypes when applying any rounding or truncation.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2024

7. OBJECT SUPPORT TESTS

The IUT shall be tested to ensure that each property of each object contained in the test database is supported. This support shall be verified by reading each property and verifying the correctness of the value returned and, where appropriate, by writing to the property and verifying an appropriate response. The read support tests are defined in 7.1 and the write support tests are defined in 7.2. Some BACnet objects are also required to provide certain functionality based on the values of their properties. Tests for the purpose of verifying this functionality are defined in 7.3.

7.1 Read Support for Properties in the Test Database

7.1.1 Read Support Test Procedure

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

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

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

Test Steps:

```

1. REPEAT X = (all objects in the IUT's database) DO {
    REPEAT Y = (all properties in object X) DO {
        IF (Y = property indicated as not accessible via the ReadProperty Service) THEN {
            TRANSMIT ReadProperty-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y
            IF (Protocol_Revision >= 13) THEN
                RECEIVE BACnet-Error PDU,
                    'Error Class' = PROPERTY,
                    'Error Code' = READ_ACCESS_DENIED
            ELSE
                RECEIVE (BACnet-Error PDU,
                    'Error Class' = OBJECT | PROPERTY,
                    'Error Code' = (any error codes for an OBJECT or PROPERTY class))
                | (BACnet-Error-PDU,
                    'Error Class' = SERVICES,
                    'Error Code' = SERVICE_REQUEST_DENIED | OTHER)
            }
        ELSE IF (Y is an array and is too long to return given the IUT's APDU and segmentation limitations) THEN
        {
            TRANSMIT ReadProperty-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y
            RECEIVE BACnet-Abort-PDU,
                'Server' = TRUE,
                'Abort Reason' = SEGMENTATION_NOT_SUPPORTED
                | BUFFER_OVERFLOW
            TRANSMIT ReadProperty-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y,
                'Property Array Index' = 0
            RECEIVE ReadProperty-ACK,
                'Object Identifier' = X,
                'PropertyIdentifier' = Y,
                'Property Array Index' = 0,
                'Property Value' = (N: number of array elements in Y as indicated in EPICS)
            REPEAT Z = (1 .. N)) DO {

```



```

        VERIFY (X), Y = (the value for element Z as indicated in the EPICS),
    ARRAY INDEX = Z
    }
    ELSE
        VERIFY (X), Y = (the value for this property specified in the EPICS)
    }
}

```

7.1.2 Non-documented Property Test

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

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

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

Test Steps:

1. REPEAT X = (a tester selected set of objects) DO {
 REPEAT Y = (0 through 511 except 8 (all), 80 (optional) and 105 (required)) DO {
 IF (the property Y is not in the EPICS for object X) THEN
 TRANSMIT ReadProperty-Request,
 'Object Identifier' = X,
 'Property Identifier' = Y
 RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = UNKNOWN_PROPERTY
 }
 }
2. IF (Protocol_Revision >= 20) THEN
 REPEAT Y = (random selection of N property IDs between 4194304 and (232 - 1)) DO {
 IF (the property Y is not in the EPICS for object X) THEN
 TRANSMIT ReadProperty-Request,
 'Object Identifier' = X,
 'Property Identifier' = Y
 RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = UNKNOWN_PROPERTY
 }
 }

7.1.3 Verifying Property_List against the EPICS

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

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

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

Notes to Tester: Object_Name (77), Object_Type (79), Object_Identifier (75), and Property_List (371) will appear in the EPICS but shall not appear in the Property_List value. Any proprietary properties that are supported for the object-type shall be in the Property_List but are not required to appear in the EPICS. The order in which property identifiers appear in the EPICS is not required to match the order that they appear in the Property_List value. If the whole BACnetARRAY cannot be read because it exceeds the Maximum Transmissible APDU, then the tester shall read it element-by-element in order to obtain the complete value.

Test Steps:

1. READ OL = Object_List

2. REPEAT O1 = (each object in the content of OL) DO {
 - READ PL = O1, Property_List
 - CHECK (that the property identifiers in the EPICS for O1 and those in PL match, except as specified in Notes to Tester)

7.2 Write Support for Properties in the Test Database

7.2.1 Functional Range Requirements for Property Values

For each writable property, multiple values will be selected by the tester as defined in this clause to verify the range of supported values.

7.2.1.1 Enumerated and Boolean Values

For enumerated and Boolean values a set of the defined enumerations, including, at a minimum, the lowest, highest, and a mid-range value, shall be tested after taking into consideration any permitted restrictions as defined in 4.4.2.

7.2.1.2 Unsigned Integer, Signed Integer, Real, and Double Values

Properties with a continuous datatype shall be tested at the upper limit, lower limit, and two intermediate points selected by the tester. The vendor shall provide the actual value of the limits for a particular implementation in the EPICS. See 4.4.2.

7.2.1.3 Octetstrings and Characterstrings,

Properties with an octetstring or characterstring datatype shall be tested with a string of the minimum supported length, a string with the maximum supported length, and a string with some length between the two. The vendor shall provide the values of the minimum and maximum string lengths in the EPICS. For string properties that do not have a fixed maximum length, the vendor shall provide a maximum length that is acceptable under normal operating conditions for use in these tests. In such cases, no statement is made as to whether or not a longer string value would or would not be accepted by the property. See Clause 4.4.2.

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

7.2.1.4 Bitstring

Properties with a Bitstring datatype shall be tested with a single value that differs from the current value.

7.2.1.5 Date

Properties with a Date datatype shall be tested using a single date that differs from the current value.

7.2.1.6 Time

Properties with a Time datatype shall be tested by writing a time value with all fields explicitly specified. This time shall differ from the previous value by an amount greater than the resolution of the IUT's clock. When the property is read back, the time shall match the written value within the resolution of the IUT's clock.

One time value with an unspecified field shall also be tested. The unspecified field shall be a time value that is within the IUT's clock resolution. For all properties except the Local_Time property of the Device object, when this value is read back the time shall match the written value within the resolution of the IUT's clock and the unspecified field shall remain unspecified. For the special case of the Local_Time property, which is coupled to the system clock, the value returned for the unspecified portion of the time is a local matter.

7.2.1.7 Constructed Datatypes

For constructed datatypes the tester shall select one or more values to write that is consistent with the datatype.

7.2.2 Write Support Test Procedure

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

Test Concept: Each writable property is written multiple times verifying the writable range. After each write, the value is verified to have been updated in the property. The test is performed once using WriteProperty and once using

WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.

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

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

Test Steps:

1. REPEAT X = (all objects in the IUT's database, except Network Port objects) DO {
 - REPEAT Y = (all writable properties in object X) DO {
 - REPEAT Z = (all values meeting the functional range requirements of 7.2.1, and any additional restrictions placed on the allowable property values by the vendor) DO {
 - WRITE (X), Y = Z,
 - VERIFY (X), Y = Z

7.2.3 Read-only Property Test

Purpose: To verify that properties marked as read-only in the EPICS are in fact read-only.

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

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

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

Notes to Tester: When modifying a property, it is expected that an Error Class of PROPERTY with an error code of WRITE_ACCESS_DENIED will be returned, but the IUT may instead return an error_class of PROPERTY with an error_code of VALUE_OUT_OF_RANGE, or an error_class of RESOURCES with an error_code of NO_SPACE_TO_WRITE_PROPERTY.

Test Steps:

1. REPEAT X = (a tester selected set of objects) DO {
 - REPEAT Y = (all read-only properties in object X) DO {
 - IF (the property is not an array) THEN
 - READ Z = (X), property Y
 - TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = X,
 - 'Property Identifier' = Y,
 - 'Property Value' = Z
 - RECEIVE BACnet-Error-PDU,
 - 'Error Class' = PROPERTY,
 - 'Error Code' = WRITE_ACCESS_DENIED
 - TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = X,
 - 'Property Identifier' = Y,
 - 'Property Value' = (any value meeting the range requirements of 7.2.1 except Z)

```

RECEIVE BACnet-Error-PDU,
  'Error Class' =      PROPERTY,
  'Error Code' =      WRITE_ACCESS_DENIED

IF (the IUT supports AddListElement and the property is a list) THEN
  TRANSMIT AddListElement-Request,
    'Object Identifier' =  X,
    'Property Identifier' = Y,
    'List of Elements' =  (any elements value meeting the range requirements
                          of 7.2.1 excluding those in Z)
  RECEIVE BACnet-Error-PDU,
    'Error Class' =      PROPERTY,
    'Error Code' =      WRITE_ACCESS_DENIED
ELSE
  READ LEN = (X), Y, ARRAY INDEX =0
  IF (LEN > 0) THEN
    READ Z = (X), Y, ARRAY INDEX=1
    TRANSMIT WriteProperty-Request,
      'Object Identifier' =  X,
      'Property Identifier' = Y,
      'Property Value' =      Z,
      'Property Array Index' = 1
    RECEIVE BACnet-Error-PDU,
      'Error Class' =      PROPERTY,
      'Error Code' =      WRITE_ACCESS_DENIED

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

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

```

7.2.4 Date Pattern Properties Test

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

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. The value, written to the property, is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. The list of Specials comes from the Chapter 21 Application Types section on Date. The day-of-week field is redundant information and can be regenerated from the other fields. In order to not fail devices which always ensure this field is consistent with the other fields in the date value, the use of an unspecified day of week is always tested in conjunction with another pattern value.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a Date. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a Date, then this test shall be skipped.

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

Test Steps:

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

7.2.5 Time Pattern Properties Test

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

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

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

Test Steps:

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

7.2.6 DateTime Pattern Properties Test

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

Test Concept: The property being tested, P1, is written with each of the special date and time field values to ensure that the property accepts them. A date, D1, is selected which is within the date range that the IUT will accept for the property. A time, T1, is selected which is within the time range that the IUT will accept for the property. The value, written to the property, is the date D1 and time T1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype which contains the BACnetDateTime, the other fields in the value shall be set within the

range accepted by the IUT. The list of DateSpecials comes from the Chapter 21 Application Types section on Date, and the list of TimeSpecials comes from the Chapter 21 Application Types section on Time.

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

Test Steps:

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

7.2.7 Date Non-Pattern Properties Test

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

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

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

Test Steps:

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

7.2.8 Time Non-Pattern Properties Test

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

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

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

Test Steps:

1. REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified) {
 TRANSMIT WriteProperty-Request
 'Object Identifier' = O1,
 'Property Identifier' = P1,

```

    'Property Value' = (V1 updated with the special value SV)
RECEIVE BACnet-Error-PDU
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE
}

```

7.2.9 DateTime Non-Pattern Properties Test

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

Test Concept: The property being tested, P1, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT1 is selected which is within the range that the IUT will accept for the property. The value, V1, written to the property, is the datetime DT1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol_Revision 11 or higher.

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

Test Steps:

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {

1. TRANSMIT WriteProperty-Request


```

        'Object Identifier' = O1,
        'Property Identifier' = P1,
        'Property Value' = (DT1 updated with the special value SV)
      
```
2. RECEIVE BACnet-Error-PDU


```

        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE
      
```

}

7.2.10 BACnetDateRange Non-Pattern Properties Test

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

Test Concept: A BACnetDateRange property, or property that is a complex datatype containing a BACnetDateRange, P1 is written with each of the special date field values to ensure that the property does not accept them. Each half of the dateRange DR1 is selected so it is within the range that the IUT will accept for the property. The value, V1, written to the property, is the dateRange DR1 with one of its fields replaced with one of the date special values. If the property is a complex datatype such as a BACnetCalendarEntry, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher. The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a BACnetDateRange. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped.

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

Test Steps:

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {

1. TRANSMIT WriteProperty-Request


```

        'Object Identifier' = O1,
        'Property Identifier' = P1,
        'Property Value' = (DR1 with startDate updated with the special value SV)
      
```
2. RECEIVE BACnet-Error-PDU


```

        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE
      
```
3. TRANSMIT WriteProperty-Request


```

        'Object Identifier' = O1,
        'Property Identifier' = P1,
        'Property Value' = (DR1 with endDate updated with the special value SV)
4. Receive BACnet-Error-PDU
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE
}

```

7.2.11 BACnetDateRange Open-Ended Pattern Properties Test

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

Test Concept: A BACnetDateRange property, or property that has a complex datatype containing a BACnetDateRange, P1 is written with a fully unspecified date in either or both halves to ensure that the property accepts them. DR1 is selected which is within the date range that the IUT will accept for the property. The value, written to the property, is the dateRange DR1 replaced with a fully unspecified date in either or both startDate and endDate. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT.

Configuration Requirements: This test shall only be applied to devices claiming Protocol_Revision 11 or higher. The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a BACnetDateRange. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped.

Notes to Tester: If P1 is an array, then an array index shall be provided in the WRITES and VERIFYS.

Test Steps:

1. WRITE P1 = (DR1 updated with a fully unspecified date in startDate)
2. VERIFY P1 = (the value written)
3. WRITE P1 = (DR1 updated with a fully unspecified date in endDate)
4. VERIFY P1 = (the value written)
5. WRITE P1 = (DR1 updated with a fully unspecified date in both startDate and endDate)
6. VERIFY P1 = (the value written)

7.3 Object Functionality Tests

The tests defined in this clause are used to verify that the required functionality for various BACnet objects is supported. The tests are object type specific and in some cases also dependent of the value of particular properties. For each object type supported, all of the tests in this clause that apply shall be executed. It is sufficient to demonstrate the correct functionality for a single instance of each object type.

7.3.1 Property Tests

The tests in this clause apply to properties that appear in multiple object types. The functionality associated with the property shall be tested once for each object type that supports the property.

7.3.1.1 Out_Of_Service, Status_Flags, and Reliability Tests

7.3.1.1.1 Out_Of_Service, Status_Flags, and Reliability Test

Purpose: To verify that Present_Value is writable when Out_Of_Service is TRUE and. that the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties.

Test Concept: The value of the Out_Of_Service property is set to TRUE, and the Present_Value property is tested to be writable. The value of the Status_Flags property is validated and, if present, the value of the Reliability property is also validated. The value of the Status_Flags property, SF1, and, if present, the Reliability property, R1, are checked to ensure they return to their initial values when the value of the Out_Of_Service property is set to FALSE.

Configuration Requirements: If the selected object is commandable, the values of the entries in the Priority_Array above the selected priority, PTY1, shall be NULL.

Test Steps:

1. READ SF1 = Status_Flags

2. IF Reliability is present THEN
 READ R1 = Reliability
3. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = TRUE
ELSE
 MAKE (Out_Of_Service TRUE)
4. VERIFY Out_Of_Service = TRUE
5. VERIFY Status_Flags = (?, ?, ?, TRUE)
6. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
 WRITE Present_Value, PTY1 = X
 VERIFY Present_Value = X
}
7. IF (Reliability is present and writable) THEN
 REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
 NO_FAULT_DETECTED) DO {
 WRITE Reliability = X
 VERIFY Reliability = X
 VERIFY Status_Flags = (?, TRUE, ?, TRUE)
 WRITE Reliability = NO_FAULT_DETECTED
 VERIFY Reliability = NO_FAULT_DETECTED
 VERIFY Status_Flags = (?, FALSE, ?, TRUE)
 }
}
8. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
ELSE
 MAKE (Out_Of_Service FALSE)
9. VERIFY Out_Of_Service = FALSE
10. VERIFY Status_Flags = SF1
11. IF Reliability is present THEN
 . VERIFY Reliability = R1

7.3.1.1.2 Out_Of_Service for Commandable Value Objects Test

Purpose: To verify that Present_Value is no longer updated by software local to the IUT when Out_Of_Service is TRUE.

Test Concept: Select an object whose Present_Value is being modified by software local to the IUT at Priority PTY1. The value of the Out_Of_Service property is set to TRUE, the Present_Value property is written at PTY1, and the Present_value is checked to ensure the Present-Value is no longer being modified by software local to the IUT.

Configuration Requirements: The values of the entries in the Priority_Array above PTY1 shall be NULL.

Test Steps:

1. MAKE (Present_Value = PV1, any valid value, using software local to the IUT)
2. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = TRUE
ELSE
 MAKE (Out_Of_Service TRUE)
3. VERIFY Present_Value = PV1
4. WRITE Present_Value, PTY1 = PV2, any valid value other than PV1
5. MAKE (Present_Value = PV3, any valid value other than PV2, using software local to the IUT)
6. VERIFY Present_Value = PV2

7.3.1.1.3 Out_Of_Service, Status_Flags, and Reliability Test for Objects without Present_Value

Purpose: This test verifies the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties. If the PICS indicates that the Out_Of_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to objects that do not contain Present_Value.

Test Concept: Write to and verify the interrelationship between the Out_Of_Service, Status_Flags, and Reliability properties of an object which does not contain Present_Value.

Configuration Requirements: The selected object is configured such that its Reliability is NO_FAULT_DETECTED before execution of this test.

Notes to Tester: When applying this test to a Network Port object in a non-routing device, once the Network Port object is out of service, the only remaining part of the test that can be executed is the verification that no BACnet communications occur through that network port. The rest of the test shall be skipped.

Test Steps:

1. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = TRUE
ELSE
 MAKE (Out_Of_Service = TRUE)
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4. IF (Reliability is present and writable) THEN
 REPEAT X = (all values of the Reliability enumeration appropriate to the object type except NO_FAULT_DETECTED) DO {
 WRITE Reliability = X
 VERIFY Reliability = X
 VERIFY Status_Flags = (?, TRUE, ?, TRUE)
 WRITE Reliability = NO_FAULT_DETECTED
 VERIFY Reliability = NO_FAULT_DETECTED
 VERIFY Status_Flags = (?, FALSE, ?, TRUE)
 }
}
5. CHECK (functionality that should stop or be disabled is. For example, with a Network Port object, all communication of the protocol modeled by the object, through that port is disabled)
6. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
ELSE
 MAKE (Out_Of_Service = FALSE)
7. VERIFY Out_Of_Service = FALSE
8. VERIFY Status_Flags = (?, ?, ?, FALSE)

7.3.1.2 Relinquish Default Test

Purpose: To verify that the Present Value property takes on the value of Relinquish Default when all prioritized commands have been relinquished. This test applies to Analog Output, Analog Value, Binary Output, Binary Value, Multi-state Output, and Multi-state Value objects that are commandable.

Test Concept: A pre-requisite to this test is that an object has been provided for which all prioritized commands have been relinquished and any **Minimum ON/OFF Time** has been accounted for. The Present_Value is compared to the value of Relinquish Default to ensure that they are the same. If possible, the value of Relinquish_Default is changed to verify that Present_Value tracks the changes.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array have a value of NULL and no internal algorithms are issuing prioritized commands to this object.

Test Steps:

1. VERIFY Priority_Array = (NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL)
2. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value
3. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value
 'Property Value' = (any valid value, X)
4. VERIFY Relinquish_Default = X

5. IF (Relinquish_Default is writable) THEN
 - WRITE Relinquish_Default = (any valid value, Y, other than the one returned in step 3)
 - VERIFY Present_Value = Y

7.3.1.3 Command Prioritization Test

Purpose: To verify that the command prioritization algorithm is properly implemented. This test applies to all commandable objects.

Test Concept: The TD selects three different values V_{low} , V_{med} , and V_{high} chosen from the valid values specified in 4.4.2. For objects that are limited to two values, V_{low} and V_{high} shall be the same, and V_{med} shall be different. The TD also selects three priorities P_{low} , P_{med} , and P_{high} , all between 1 and 5, such that numerically $P_{low} > P_{med} > P_{high}$. The selected values are written one at a time to Present_Value at the corresponding priority. The Present_Value and Priority_Array are checked to verify correct operation. Priorities numerically smaller than 6 (higher priority) are used to eliminate **Minimum ON/OFF Time** considerations

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array with a priority higher than 6 have a value of NULL.

Test Steps:

1. WRITE Present_Value = V_{low} , PRIORITY = P_{low}
2. VERIFY Present_Value = V_{low}
3. VERIFY Priority_Array = V_{low} , ARRAY INDEX = P_{low}
4. REPEAT Z = (each index 1 through 5 not equal to P_{low}) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z
5. WRITE Present_Value = V_{high} , PRIORITY = P_{high}
6. VERIFY Present_Value = V_{high}
7. VERIFY Priority_Array = V_{high} , ARRAY INDEX = P_{high}
8. REPEAT Z = (each index 1 through 5 not equal to P_{low} or P_{high}) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z
9. WRITE Present_Value = V_{med} , PRIORITY = P_{med}
10. VERIFY Present_Value = V_{high}
11. VERIFY Priority_Array = V_{med} , ARRAY INDEX = P_{med}
12. REPEAT Z = (each index 1 through 5 not equal to P_{low} , P_{med} or P_{high}) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z
13. WRITE Present_Value = NULL, PRIORITY = P_{high}
14. VERIFY Present_Value = V_{med}
15. REPEAT Z = (each index 1 through 5 not equal to P_{low} or P_{med}) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z
16. WRITE Present_Value = NULL, PRIORITY = P_{med}
17. VERIFY Present_Value = V_{low}
18. REPEAT Z = (each index 1 through 5 not equal to P_{low}) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z
19. WRITE Present_Value = NULL, PRIORITY = P_{low}
20. REPEAT Z = (each index 1 through 5) DO {
 - VERIFY Priority_Array = NULL, ARRAY INDEX = Z

7.3.1.4 Minimum_Off_Time

Purpose: To verify that the minimum off time algorithm is properly implemented. If minimum off time is not supported this test shall be omitted. This test applies to Binary Output and Binary Value objects.

Test Concept: The initial Present_Value of the object tested is set to ACTIVE and it is controlled at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum on time to have expired.

The Present_Value is written to with a value of INACTIVE at priority 7. The value of slot 6 of the Priority_Array is monitored to verify that it contains the value INACTIVE for the duration of the minimum off time.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically less than 7 have a value of NULL, the Present_Value is ACTIVE, and no internal algorithms are issuing commands to this object at a priority numerically lower (higher priority) than the priority that is currently controlling Present_Value.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = 7
2. VERIFY Present_Value = INACTIVE
3. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
4. WAIT (approximately 90% of Minimum_Off_Time from step 1)
5. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
6. WAIT (**Minimum ON/OFF Fail Time** + Minimum_Off_Time from step 1)
7. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.5 Minimum_On_Time

Purpose: To verify that the minimum on time algorithm is properly implemented. If minimum on time is not supported this test shall be omitted. This test applies to Binary Output and Binary Value objects.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE and it is controlled at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum on time to have expired. The Present_Value is written to with a value of ACTIVE at priority 7. The value of slot 6 of the Priority_Array is monitored to verify that it contains the value ACTIVE for the duration of the minimum on time.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically less than 7 have a value of NULL, the Present_Value is INACTIVE, and no internal algorithms are issuing commands to this object at a priority numerically lower (higher priority) than the priority that is currently controlling Present_Value.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
4. WAIT (approximately 90% of Minimum_On_Time from step 1)
5. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
6. WAIT (**Minimum ON/OFF Fail Time** + Minimum_On_Time from step 1)
7. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.6 Minimum On/Off Time Tests

7.3.1.6.1 Override of Minimum Time

Purpose: To verify that higher priority commands override minimum on or off times. If neither minimum on time or minimum off time is supported this test shall be omitted. This test applies to Binary Output and commandable Binary Value objects.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE and it is controlled at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off and/or minimum on time to have expired. The Present_Value is written to with a value of ACTIVE at priority 7. The value of slot 6 of the Priority_Array is monitored to verify that it contains the value ACTIVE. Before the minimum on time expires the Present_Value is written to with a value of INACTIVE and a priority numerically lower (higher priority) than 6. This overrides the minimum on time and immediately initiates the minimum off time algorithm.

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array numerically less than 7 have a value of NULL and no internal algorithms are issuing commands to this object at a priority numerically lesser (higher priority) than the priority that is currently controlling Present_Value. Minimum_On_Time must be configured with a large enough value to allow execution of all test steps before it expires.

Notes to Tester: If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1, 2, 3, and 6 through 3 and ACTIVE in steps 4 through 76 and 5, and by using the Minimum_Off_Time in Step 4.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
4. BEFORE Minimum_On_Time
WRITE Present_Value = INACTIVE, PRIORITY = (any value numerically lower than 6 (higher priority))
5. VERIFY Present_Value = INACTIVE
6. VERIFY Priority_Array $\diamond$ ACTIVE, ARRAY INDEX = 6

7.3.1.6.2 Minimum Off Time - Writing at priorities numerically greater than 6

Purpose: To verify that commands written at a lower priority than 6 will not affect Present_Value while Minimum_Off_Time is in effect.

Test Concept: The initial Present_Value of the object tested is set to ACTIVE and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than P9 ($P9 > 7$). The object has been in this state long enough for any minimum on time to have expired. The Present_Value of the object is set to INACTIVE at a priority P9. Before Minimum_Off_Time expires, Present_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority_Array is monitored to verify that it contains the appropriate values, and Present_Value is monitored to verify that it does not change before Minimum_Off_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Active	Inactive	Inactive	Inactive	Inactive	Active
PA_Index = 6	Null	Inactive	Inactive	Inactive	Inactive	$\diamond$ Inactive
PA_Index = P7	Null	Null	Null	Active	Active	Active
PA_Index = P9	Null	Inactive	Null	Null	Active	Active
Relinquish_Default	Active	Active	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation the Minimum_Off_Time End of Test

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array from P9 and higher (numerically lesser) have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_Off_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = P9
2. VERIFY Present_Value = INACTIVE
3. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P9
5. VERIFY Present_Value = INACTIVE
6. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
- (Steps 4-6: Check that a NULL value at P9 does not affect ARRAY INDEX = 6 or PV)
7. WRITE Present_Value = ACTIVE, PRIORITY = P7 ($6 < P7 < P9$)
8. VERIFY Present_Value = INACTIVE
9. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = P7
10. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6

- (Steps 7-10: Check that an ACTIVE value at P7 does not affect ARRAY INDEX = 6 or PV)
11. WRITE Present_Value = ACTIVE, PRIORITY = P9
 12. VERIFY Present_Value = INACTIVE
 13. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = P9
 14. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
- (Steps 11-14: Check that an ACTIVE value at P9 does not affect ARRAY INDEX = 6 or PV)
15. WAIT (Minimum_Off_Time + **Internal Processing Fail Time**)
 16. VERIFY Present_Value = ACTIVE

7.3.1.6.3 Minimum On Time - Writing at priorities numerically greater than 6

Purpose: To verify that commands written at a lower priority than 6 will not affect Present_Value while Minimum_On_Time is in effect.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE, and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than P9 ($P9 > 7$). The object has been in this state long enough for any minimum off time to have expired. The Present_Value of the object tested is set to ACTIVE at a priority P9. Before Minimum_On_Time expires, Present_Value is written with values of ACTIVE and NULL at Priorities P9 and P7, where P7 is a priority between P9 and 6. The Priority_Array is monitored to verify that it contains the appropriate values, and Present_Value is monitored to verify that it does not change before Minimum_On_Time expires.

Test Step(s) →	Start of Test	1-3	4-6	7-10	11-15	16
Present_Value	Inactive	Active	Active	Active	Active	Inactive
PA_Index = 6	Null	Active	Active	Active	Active	◇Active
PA_Index = P7	Null	Null	Null	Inactive	Inactive	Inactive
PA_Index = P9	Null	Active	Null	Null	Inactive	Inactive
Relinquish_Default	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

End of Test

Minimum_On_Time

Configuration Requirements: The object to be tested shall be configured such that all slots from P9 and higher (numerically lesser) in the Priority_Array have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_On_Time must be configured with a large enough value to allow execution of test steps 1-14 before it expires.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = P9
 2. VERIFY Present_Value = ACTIVE
 3. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
 4. WRITE Present_Value = NULL, PRIORITY = P9
 5. VERIFY Present_Value = ACTIVE
 6. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Steps 4-6: Check that a NULL value at P9 does not affect ARRAY INDEX = 6 or PV)
7. WRITE Present_Value = INACTIVE, PRIORITY = P7 ($6 < P7 < P9$)
 8. VERIFY Present_Value = ACTIVE
 9. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = P7
 10. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Steps 7-10: Check that an INACTIVE value at P7 does not affect ARRAY INDEX = 6 or PV)
11. WRITE Present_Value = INACTIVE, PRIORITY = P9
 12. VERIFY Present_Value = ACTIVE
 13. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = P9
 14. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6

-- (Steps 11-14: Check that an INACTIVE value at P9 does not affect ARRAY INDEX = 6 or PV)

15. WAIT (Minimum_On_Time + **Internal Processing Fail Time**)

16. VERIFY Present_Value = INACTIVE

7.3.1.6.4 Minimum Off Time - Writing at priorities numerically lesser than 6

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum off time is in effect.

Test Concept: The initial Present_Value of the object tested is set to ACTIVE, and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum on time to have expired. The Present_Value of the object tested is set to INACTIVE at a priority P5 ($P5 < 6$). Before Minimum_Off_Time expires, Present_Value is written with values of NULL and ACTIVE, and the Present_Value and Priority_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Active	Inactive	Inactive	Active
PA Index = P5	Null	Inactive	Null	Active
PA Index = 6	Null	Inactive	Inactive	<>Inactive
Relinquish Default	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_Off_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present_Value = INACTIVE, PRIORITY = P5
 2. VERIFY Present_Value = INACTIVE
 3. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
 4. WRITE Present_Value = NULL, PRIORITY = P5
 5. VERIFY Present_Value = INACTIVE
 6. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
 7. VERIFY Priority_Array = NULL, ARRAY INDEX = P5
- (Steps 4-7: Check that a NULL value at P5 will NOT change ARRAY INDEX = 6 or PV)
8. WRITE Present_Value = ACTIVE, PRIORITY = P5
 9. VERIFY Present_Value = ACTIVE
 10. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = P5
 11. VERIFY Priority_Array <> INACTIVE, ARRAY INDEX = 6
- (Steps 8-11: Check that an ACTIVE value at P5 will change ARRAY INDEX = 6 and PV)

7.3.1.6.5 Minimum On Time - Writing at priorities numerically lesser than 6

Purpose: To verify that the value at priority 6 contains the appropriate value when commands are written at a higher priority and minimum on time is in effect.

Test Concept: The initial Present_Value of the object tested is set to INACTIVE, and it is controlled by the Relinquish_Default value or at a priority numerically greater (lower priority) than 6. The object has been in this state long enough for any minimum off time to have expired. The Present_Value of the object tested is set to ACTIVE at a priority P5 ($P5 < 6$). Before Minimum_On_Time expires, Present_Value is written with values of NULL and INACTIVE, and the Present_Value and Priority_Array properties are observed for correct behavior.

Test Steps →	Start of Test	1-3	4-7	8-11
Present_Value	Inactive	Active	Active	Inactive
PA_Index = P5	Null	Active	Null	Inactive
PA_Index = 6	Null	Active	Active	◇Active
Relinquish_Default	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

Configuration Requirements: The object to be tested shall be configured such that all slots in the Priority_Array higher (numerically lesser) than 6 have a value of NULL and no internal algorithms are issuing commands to this object. Minimum_On_Time must be configured with a large enough value to allow execution of the test steps before it expires.

Test Steps:

1. WRITE Present_Value = ACTIVE, PRIORITY = P5
2. VERIFY Present_Value = ACTIVE
3. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
4. WRITE Present_Value = NULL, PRIORITY = P5
5. VERIFY Present_Value = ACTIVE
6. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
7. VERIFY Priority_Array = NULL, ARRAY INDEX = P5
- (Steps 4-7: Check that a NULL value at P5 will NOT change ARRAY INDEX = 6 or PV)
8. WRITE Present_Value = INACTIVE, PRIORITY = P5
9. VERIFY Present_Value = INACTIVE
10. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = P5
11. VERIFY Priority_Array ◇ ACTIVE, ARRAY INDEX = 6
- (Steps 8-11: Check that an INACTIVE value at P5 will change ARRAY INDEX = 6 and PV)

7.3.1.6.6 Minimum_Off_Time - Clock is not affected by additional write operations

Purpose: To verify that the Minimum_Off_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE at priority P8, such that present-value and slot 6 in the priority-array change to INACTIVE. At time T1, which occurs before minimum off time expires, another write request, at priority P9, with a value of INACTIVE, is executed by the device. After minimum off time expires but before T1 + Minimum_Off_Time, slot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Active	Inactive	Inactive	Inactive
PA_Index = P6	Null	Inactive	Inactive	Null
PA_Index = PX8	Null	Inactive	Inactive	Inactive
PA_Index = PY9	Null	Null	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE and slot 6 in the Priority_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Notes to Tester: P8 and P9 may assume any value in the Priority_Array (except 6) and may be equal.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = INACTIVE, PRIORITY = P8
4. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
- (Execute step 5 at time T1)
5. WRITE Present_Value = INACTIVE, PRIORITY = PY9
- (Execute steps 6 and 7 before Minimum_Off_Time expires)
6. VERIFY Present_Value = INACTIVE
7. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
8. WAIT for Minimum_Off_Time to expire
- (Execute step 9 before T1 + Minimum_Off_Time)
9. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.6.7 Minimum_On_Time - Clock is not affected by additional write operations

Purpose: To verify that the Minimum_On_Time timer is not affected by subsequent write operations that do not cause present-value to change.

Test Concept: The initial Present_Value of the object being tested is set to INACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to ACTIVE, at priority P8, such that present-value and slot 6 in the priority-array change to ACTIVE. At time T1, which occurs before minimum on time expires, another write request, at priority P9, with a value of ACTIVE, is executed by the device. After minimum on time expires but before T1 + Minimum_On_Time, Sslot 6 in the priority-array is checked to verify that it returned to NULL and was not affected by the second request.

Test Step(s) →	1-2	3-4	5-8	9
Present_Value	Inactive	Active	Active	Active
PA_Index = P6	Null	Active	Active	Null
PA_Index = P8	Null	Active	Active	Active
PA_Index = P9	Null	Null	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum On Time

Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE, and slot 6 in the Priority_Array has a value of NULL. P8 and P9 are selected such that they are higher (lesser numerically) than any other commanding priority.

Notes to Tester: P8 and P9 may assume any value in the Priority_Array (except 6) and may be equal.

Test Steps:

1. VERIFY Present_Value = INACTIVE
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = ACTIVE, PRIORITY = P8
4. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Execute step 5 at time T1)
5. WRITE Present_Value = ACTIVE, PRIORITY = P9
- (Execute steps 6 and 7 before Minimum_On_Time expires)
6. VERIFY Present_Value = ACTIVE
7. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
8. WAIT for Minimum_On_Time to expire
- (Execute step 9 before T1 + Minimum_On_Time)
9. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.6.8 Ensuring Minimum_Off_Time starts at transition to INACTIVE

Purpose: To verify that Minimum_Off_Time does not start immediately after a write operation while Minimum_On_Time is in effect and present-value is ACTIVE.

Test Concept: The initial Present_Value of the object being tested is set to INACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to ACTIVE at P9, where P9 is a priority between 7 and 16, such that present-value and slot 6 in the priority-array change to ACTIVE. Before Minimum_On_Time expires, Present_Value is written to INACTIVE at P7, where P7 is a priority between 7 and P9, such that Present_Value would change if Minimum_On_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:

T1 = the time when the INACTIVE request is executed by the device + Minimum_Off_Time

T2 = the time when the ACTIVE request is executed by the device + Minimum_On_Time + Minimum_Off_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Inactive	Active	Active	Inactive	Inactive	Inactive
PA_Index = 6	Null	Active	Active	Inactive	Inactive	Null
PA_Index = P7	Null	Null	Inactive	Inactive	Inactive	Inactive
PA_Index = P9		Active	Active	Active	Active	Active

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

T1

T2

Configuration Requirements: The object to be tested shall be configured such that the present value is set to INACTIVE, and slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_On_Time and Minimum_Off_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum_On_Time and Minimum_Off_Time properties, this test shall be skipped.

Notes to Tester: P9 and P7 may be equal.

Test Steps:

1. VERIFY Present_Value = INACTIVE
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = ACTIVE, PRIORITY = P9
4. VERIFY Present_Value = ACTIVE
5. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Execute steps 6 through 7 before Minimum_On_Time expires)
6. WRITE Present_Value = INACTIVE, PRIORITY = P7
7. VERIFY Present_Value = ACTIVE
8. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6

9. WAIT for Minimum_On_Time to expire
-- (Execute steps 10 and 11 before T1)
10. VERIFY Present_Value = INACTIVE
11. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
-- (Execute step 12 between T1 and T2)
12. VERIFY Present_Value = INACTIVE
13. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
-- (Execute step 14 and 15 after T2)
14. VERIFY Present_Value = INACTIVE
15. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.6.9 Ensuring Minimum_On_Time starts at transition to ACTIVE

Purpose: To verify that Minimum_On_Time does not start immediately after a write operation while Minimum_Off_Time is in effect and present-value is INACTIVE.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE at P9, where P9 is a priority between 7 and 16, such that present-value and slot 6 in the priority-array change to INACTIVE. Before Minimum_Off_Time expires, Present_Value is written to ACTIVE at P7, where P7 is a priority between 7 and P9, such that Present_Value would change if Minimum_Off_Time were not in effect. Slot 6 in the priority-array is monitored at specific time intervals to ensure that it contains the appropriate value. Time references T1 and T2 are defined for this test as follows:

T1 = the time when the ACTIVE request is executed by the device + Minimum_On_Time

T2 = the time when the INACTIVE request is executed by the device + Minimum_Off_Time + Minimum_On_Time

Test Steps →	1-2	3-5	6-9	10-11	12-13	14-15
Present_Value	Active	Inactive	Inactive	Active	Active	Active
PA_Index = 6	Null	Inactive	Inactive	Active	Active	Null
PA_Index = P7	Null	Null	Active	Active	Active	Active
PA_Index = P9		Inactive	Inactive	Inactive	Inactive	Inactive

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

T1

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE, and slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_On_Time and Minimum_Off_Time values sufficiently large enough to allow execution of this test. If no object exists with both Minimum_On_Time and Minimum_Off_Time properties, this test shall be skipped.

Notes to Tester: P9 and P7 may be equal.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = INACTIVE, PRIORITY = P9
4. VERIFY Present_Value = INACTIVE
5. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
-- (Execute steps 6 through 7 before Minimum_Off_Time expires)
6. WRITE Present_Value = ACTIVE, PRIORITY = P7
7. VERIFY Present_Value = INACTIVE
8. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
9. WAIT for Minimum_Off_Time to expire
-- (Execute steps 10 and 11 before T1)

10. VERIFY Present_Value = ACTIVE
11. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Execute step 12 between T1 and T2)
12. VERIFY Present_Value = ACTIVE
13. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
- (Execute step 14 and 15 after T2)
14. VERIFY Present_Value = ACTIVE
15. VERIFY Priority_Array = NULL, ARRAY INDEX = 6

7.3.1.6.10 Ensuring Minimum Times Are Not Affected By Time Changes

Purpose: To verify that minimum times are not affected by changing the time in a device via TimeSynchronization or UTCTimeSynchronization requests.

Test Concept: The initial Present_Value of the object being tested is set to ACTIVE, and the value at slot 6 in the priority-array has a value of NULL. Present_Value of the object is written to INACTIVE such that present-value and slot 6 in the priority-array change to INACTIVE. Before Minimum_Off_Time expires, the time is changed to a value T1 which is more than Minimum_Off_Time in the future and Present_Value, and Slot 6 in the priority-array are read to verify that they were not affected by the time change. After Minimum_Off_Time expires, slot 6 in the priority-array is read again to verify that it is no longer INACTIVE.

Configuration Requirements: The object to be tested shall be configured such that the present value is set to ACTIVE, and slot 6 in the Priority_Array has a value of NULL. If the IUT does not support TimeSynchronization or UTC-TimeSynchronization, then this test shall be omitted.

Notes to Tester: The test above is written for Minimum_Off_Time. To execute this test for Minimum_On_Time, use INACTIVE where ACTIVE is specified, ACTIVE where INACTIVE is specified, and Minimum_On_Time where Minimum_Off_Time is specified.

Test Steps:

1. VERIFY Present_Value = ACTIVE
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = INACTIVE
4. VERIFY Present_Value = INACTIVE
5. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
6. TRANSMIT
 - DA = GLOBAL BROADCAST,
 - SA = TD
 - BACnet-Unconfirmed-Request-PDU,
 - 'Service Choice' = TimeSynchronization-Request,
 - Date = T1,
 - Time = T1
7. TRANSMIT
 - DA = GLOBAL BROADCAST,
 - SA = TD
 - BACnet-Unconfirmed-Request-PDU,
 - 'Service Choice' = UTC-TimeSynchronization-Request,
 - Date = T1,
 - Time = T1
8. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
9. WAIT (the remainder of Minimum_Off_Time)
10. VERIFY Priority_Array <> INACTIVE, ARRAY INDEX = 6

7.3.1.6.11 Minimum_Off_Time - Value Source Mechanism

Purpose: To verify that the value source used for priority 6 is the commanded object while Minimum_Off_Time is in effect.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. When Minimum_Off_Time takes effect, the Present_Value is written. The Value_Source and Value_Source_Array properties are monitored to verify that the source for priority 6 is the commanded object.

Configuration Requirements: The object, O1, to be tested shall be configured such that slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_Off_Time values sufficiently large enough to allow execution of this test. If no object exists with Minimum_Off_Time property, this test shall be skipped.

Notes to Tester: Comparisons with O1 are with or without the optional device-identifier of IUT.

Test Steps:

1. VERIFY Value_Source = (any valid value)
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = INACTIVE, PRIORITY > 6
4. VERIFY Present_Value = INACTIVE
5. VERIFY Priority_Array = INACTIVE, ARRAY INDEX = 6
6. VERIFY Value_Source = O1
7. VERIFY Value_Source_Array = Value_Source, ARRAY INDEX = 6
9. WAIT (Minimum ON/OFF Fail Time + Minimum_Off_Time)
10. VERIFY Value_Source <> O1

7.3.1.6.12 Minimum_On_Time - Value Source Mechanism

Purpose: To verify that the value source used for priority 6 is the commanded object while Minimum_On_Time is in effect.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. When Minimum_On_Time takes effect, the Present_Value is written. The Value_Source and Value_Source_Array properties are monitored to verify that the source for priority 6 is the commanded object.

Configuration Requirements: The object, O1, to be tested shall be configured such that slot 6 in the Priority_Array has a value of NULL. The object being tested must also be configured with Minimum_On_Time values sufficiently large enough to allow execution of this test. If no object exists with Minimum_On_Time property, this test shall be skipped.

Notes to Tester: Comparisons with O1 are with or without the optional device-identifier of IUT.

Test Steps:

1. VERIFY Value_Source = (any valid value)
2. VERIFY Priority_Array = NULL, ARRAY INDEX = 6
3. WRITE Present_Value = ACTIVE, PRIORITY > 6
4. VERIFY Present_Value = ACTIVE
5. VERIFY Priority_Array = ACTIVE, ARRAY INDEX = 6
6. VERIFY Value_Source = O1
7. VERIFY Value_Source_Array = Value_Source, ARRAY INDEX = 6
8. WAIT (Minimum ON/OFF Fail Time + Minimum_On_Time)
9. VERIFY Value_Source <> O1

7.3.1.7 COV Tests

Tests to demonstrate COV functionality are covered in 8.2 and 9.6.

7.3.1.7.1 COV_Resubscription_Interval Test

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

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

Configuration Requirements: O1 is configured to acquire data from TD by COV notification. Non-zero values shall be chosen for COV_Resubscription_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

1. IF (the IUT uses SubscribeCOV) THEN

```

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

```

'Monitored Property Identifier' = MPI1,
'COV Increment' = CI1

11. TRANSMIT BACnet-SimpleACK-PDU

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

7.3.1.8 Change of State Tests

Purpose: To verify that the properties of objects that collectively track state changes (changes in Present_Value) function as required.

Test Concept: The Present_Value of the object under test is changed. The Change_Of_State_Count property is checked to verify that it has been incremented, and the Change_Of_State_Time property is checked to verify that it has been updated. The Change_Of_State_Count is reset and Time_Of_State_Count_Reset is checked to verify that it has been updated appropriately.

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

Test Steps:

1. READ PV = Present_Value
2. READ N = Change_of_State_Count
3. IF (PV = ACTIVE) THEN
 - IF (Present_Value is writable) THEN
 - WRITE Present_Value = INACTIVE
 - VERIFY Present_Value = INACTIVE
 - ELSE
 - MAKE (Present_Value = INACTIVE)
- ELSE
 - IF (Present_Value is writable) THEN
 - WRITE Present_Value = ACTIVE
 - VERIFY Present_Value = ACTIVE
 - ELSE
 - MAKE (Present_Value = ACTIVE)
4. VERIFY (Change_of_State_Count = N+1)
5. VERIFY (Change_Of_State_Time ~= the current local date and time)
6. IF (Change_of_State_Count is writable) THEN
 - WRITE Change_of_State_Count = 0
- ELSE
 - MAKE (Change_of_State_Count = 0)
7. VERIFY Time_Of_State_Count_Reset ~= (the current local date and time)

7.3.1.9 Elapsed Active Time Tests

Purpose: To verify that the properties of objects that collectively track active time function properly.

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

Configuration Requirements: The object being tested shall be configured such that the Present_Value or Feedback_Value if that is used for the calculation, and Elapsed_Active_Time properties are writable or another means of changing these properties shall be provided. Whether Present_Value or Feedback_Value is used as the indicator for the calculation of the Elapsed_Active_Time is a local matter.

Notes To Tester: It was intentional to specify that the alternative use of Feedback_Value tracking specified in 135-2010ad-3 is allowed regardless of the Protocol_Revision claimed by the implementation.

Test Steps:

1. IF (Present_Value is writable) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
ELSE
 MAKE (Present_Value = INACTIVE)
2. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
 WAIT (long enough for Feedback_Value to reflect the Present_Value)
 VERIFY Feedback_Value = INACTIVE
3. READ Elapsed_Active_Time = initialElapsedTime
- verify that Elapsed_Active_Time does not change when the object is INACTIVE
4. WAIT (more than Internal_Processing Fail Time + at least 1 second)
5. VERIFY Elapsed_Active_Time = initialElapsedTime
- verify that Elapsed_Active_Time correctly reflects the time the object is ACTIVE
6. IF (Present_Value is writable) THEN
 WRITE Present_Value = ACTIVE
 VERIFY Present_Value = ACTIVE
ELSE
 MAKE (Present_Value = ACTIVE)
7. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
 WAIT (long enough for Feedback_Value to reflect the Present_Value)
 VERIFY Feedback_Value = ACTIVE
8. READ initialTime = (the IUT's Device object) Local_Time
9. WAIT (more than **Internal Processing Fail Time** + 30 seconds)
10. IF (Present_Value is writable) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
ELSE
 MAKE (Present_Value = INACTIVE)
11. IF (Feedback_Value is used for Elapsed_Active_Time tracking) THEN
 WAIT (long enough for Feedback_Value to reflect the Present_Value)
 VERIFY Feedback_Value = INACTIVE
12. READ currentTime = (the IUT's Device object) Local_Time
13. READ totalElapsedTime = Elapsed_Active_Time
14. CHECK (totalElapsedTime == (currentTime - initialTime) - initialElapsedTime)
- verify ability to reset Elapsed_Active_Time, if it is writable
15. IF (Elapsed_Active_Time is writable) THEN
 WRITE Elapsed_Active_Time = 0
 READ currentDate = (the IUT's Device object) Local_Date
 READ currentTime = (the IUT's Device object) Local_Time
 VERIFY Time_Of_Active_Time_Reset == { currentDate, currentTime }

7.3.1.10 Event_Enable Tests

7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL, TO_NORMAL, and TO_FAULT

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

Test Concept: The IUT is configured with an event-generating object, O1, such that the Event_Enable property is tested in all supported states. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

Configuration Requirements: If the Event_Enable property is configurable, repeat the test with Event_Enable = (T, F, F), (F, T, F), (F, F, T). If the Event_Enable property is not configurable, then follow the test steps as written and verify correct behavior for the value of the Event_Enable property. All other properties in O1, and any supporting objects, shall be configured to allow these events to be generated. The event-generating object shall be in a NORMAL state at the start of the test. D1 is either the pTimeDelay parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).

D2 is either the pTimeDelayNormal parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. WAIT (pTimeDelay + **Notification Fail Time**)
3. IF (O1 contains pFeedbackValue) THEN
MAKE (pFeedbackValue differ from pMonitoredValue)
ELSE IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value that is OFFNORMAL)
ELSE
MAKE (pMonitoredValue have a value that is OFFNORMAL)
4. WAIT (D1)
5. BEFORE **Notification Fail Time**
IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN {
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = OFFNORMAL,
'Event Values' = (values appropriate to the event type)
TRANSMIT BACnet-SimpleACK-PDU
}
ELSE
CHECK (verify that the IUT did not transmit an event notification message)
6. VERIFY pCurrentState = OFFNORMAL
7. IF (O1 contains pFeedbackValue) THEN
MAKE (pFeedbackValue equal to pMonitoredValue)
ELSE IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value that is NORMAL)
ELSE
MAKE (pMonitoredValue have a value that is NORMAL)
8. WAIT (D2)
9. BEFORE **Notification Fail Time**
IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN {
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),

```

'Priority' = (the value configured to correspond to a TO-NORMAL transition),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = OFFNORMAL,
'To State' = NORMAL,
'Event Values' = (values appropriate to the event type)
TRANSMIT BACnet-SimpleACK-PDU
}
ELSE
    CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY pCurrentState = NORMAL
11. IF (O1 can be placed into a fault condition) THEN {
    MAKE (a condition exist that will cause O1 to generated a TO-FAULT transition)
    BEFORE Notification Fail Time
    IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN {
        RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = O1(the event-generating object configured for this test),
        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (the class corresponding to the object being tested),
        'Priority' = (the value configured to correspond to a TO-FAULT transition),
        'Event Type' = (any valid event type),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = NORMAL,
        'To State' = FAULT,
        'Event Values' = (values appropriate to the event type)
        TRANSMIT BACnet-SimpleACK-PDU
    }
    ELSE
        CHECK (verify that the IUT did not transmit an event notification message)
        VERIFY Event_State = FAULT
    }
}

```

7.3.1.10.2 Event_Enable Tests for TO_NORMAL only Algorithms

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

Test Concept: The IUT is configured such that the Event_Enable property indicates that some event transitions are to trigger an event notification. Each event transition is triggered, and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

Configuration Requirements: In the Notification Class object providing recipient information, the value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE). The event-generating object shall be in a NORMAL state at the start of the test. D1 is either the pTimeDelayNormal parameter or, in case of the EXTENDED and proprietary algorithms, a vendor specific delay (may be zero).

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (the TO-NORMAL bit of the Event_Enable property equal to TRUE)
3. MAKE (a condition exist that will cause the object to generate a TO_NORMAL transition)
4. WAIT D1
5. BEFORE **Notification Fail Time**
 - RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),

- 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)
6. TRANSMITBACnet-SimpleACK-PDU
7. VERIFY pCurrentState = NORMAL
8. IF (Event_Enable can be changed such that the TO-NORMAL transition is FALSE) THEN {
 MAKE (the TO-NORMAL bit of the Event_Enable property equal to FALSE)
 MAKE (a condition exist that would cause the object to generate a TO_NORMAL transition)
 WAIT (D1 + **Notification Fail Time**)
 CHECK (verify that the IUT did not transmit an event notification message)
 MAKE (the TO_NORMAL bit of the Event_Enable property equal to TRUE)
 }
9. IF (the event-generating object can be placed into a fault condition) THEN {
 IF (Event_Enable can be modified) THEN
 MAKE (Event_Enable TO-FAULT transition equal TRUE)
 IF (Event_Enable TO-FAULT transition = TRUE) THEN {
 MAKE (the event-triggering object change to a fault condition)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-FAULT transition),
 'Event Type' = (IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = FAULT,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 VERIFY pCurrentState = FAULT
 MAKE (the event-generating object change to a normal condition)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY),
 'Message Text' = (optional, any valid message text),

```

        'Notify Type' =          EVENT | ALARM,
        'AckRequired' =         TRUE | FALSE,
        'From State' =          FAULT,
        'To State' =             NORMAL,
        'Event Values' =         (values appropriate to the event type)
    TRANSMIT BACnet-SimpleACK-PDU
    VERIFY pCurrentState = NORMAL
}
}
10. IF (Event_Enable can be modified) THEN
    MAKE (Event_Enable TO-FAULT transition equal FALSE)
11. IF (Event_Enable TO-FAULT transition = FALSE) THEN {
    MAKE (the event-triggering object change to a fault condition)
    VERIFY pCurrentState = FAULT
    CHECK (verify that the IUT did not transmit an event notification message)
    MAKE (the event-triggering object change to a normal condition)
}
}

```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

7.3.1.11 Acked_Transitions Tests

7.3.1.11.1 Acked_Transitions Test

Purpose: To verify that the Acked_Transitions property tracks whether or not an acknowledgment has been received for a previously issued event notification. It also verifies the interrelationship between Status_Flags and Event_State.

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

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

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip sending the BACnet-SimpleACK-PDU messages after receiving the notifications.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
3. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
4. IF (pMonitoredValue is writable) THEN
WRITE pMonitoredValue = (a value that is OFFNORMAL)
ELSE
MAKE (pMonitoredValue have a value that is OFFNORMAL)
5. WAIT (pTimeDelay)
6. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (PI1: any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the event-generating object configured for this test),
'Time Stamp' = (Toffnormal: any valid time stamp),
'Notification Class' = (the class corresponding to the object being tested),
'Priority' = (Poffnormal: the value configured to correspond to a TO-OFFNORMAL)

- transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = (the notify type configured for this event),
 'AckRequired' = TRUE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = (values appropriate to the event type)
7. TRANSMIT BACnet-SimpleACK-PDU
 8. VERIFY pCurrentState = OFFNORMAL
 9. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
 10. IF (Protocol_revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 11. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that is NORMAL)
 ELSE
 MAKE (pMonitoredValue have a value that is NORMAL)
 12. WAIT (pTimeDelayNormal)
 13. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (PI2: any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (Tnormal: any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (Pnormal: the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = (the notify type configured for this event),
 'AckRequired' = TRUE,
 'From State' = OFNORMAL,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)
 14. TRANSMIT BACnet-SimpleACK-PDU
 15. VERIFY pCurrentState = NORMAL
 16. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
 17. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
 18. IF (the event-triggering object can be placed into a fault condition) THEN {
 MAKE (a condition exist that will cause the object to generate a fault condition)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (PI3: any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (Tfault: any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (Pfault: the value configured to correspond to a TO-FAULT transition),
 'Event Type' = IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = (the notify type configured for this event),
 'AckRequired' = TRUE,
 'From State' = NORMAL,
 'To State' = FAULT,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 VERIFY pCurrentState = FAULT

VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)

TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (PI3),

'Event Object Identifier' = (the event-generating object configured for this test),

'Event State Acknowledged' = FAULT,

'Acknowledgement Source' = (a character string),

'Time Stamp' = (Tfault),

'Time of Acknowledgment' = (the TD's current time)

RECEIVE BACnet-SimpleACK-PDU

IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI3),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Tfault the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (Pfault),

'Event Type' = IF (Protocol_Revision < 13)
(any valid event type),

ELSE

CHANGE_OF_RELIABILITY,

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK_NOTIFICATION,

'To State' = FAULT

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI3),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Tfault the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (Pfault),

'Event Type' = (any valid event type),

'Notify Type' = ACK_NOTIFICATION

TRANSMIT BACnet-SimpleACK-PDU

VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)

}

19. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (PI2),

'Event Object Identifier' = (the event-generating object configured for this test),

'Event State Acknowledged' = NORMAL,

'Time Stamp' = (Tnormal),

'Acknowledgement Source' = (a character string),

'Time of Acknowledgment' = (the TD's current time)

20. RECEIVE BACnet-SimpleACK-PDU

21. IF (Protocol_Revision is present and Protocol_Revision >= 1) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI2),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Tnormal the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (Pnormal),

'Event Type' = (any valid event type),

'Notify Type' = ACK_NOTIFICATION,

'To State' = NORMAL

ELSE

BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI2),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Tnormal the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object bind tested),

'Priority' = (Pnormal),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK_NOTIFICATION

22. TRANSMIT BACnet-SimpleACK-PDU

23. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)

24. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (PI1),

'Event Object Identifier' = (the event-generating object configured for this test),

'Event State Acknowledged' = OFFNORMAL,

'Time Stamp' = (Toffnormal),

'Acknowledgement Source' = (a character string),

'Time of Acknowledgment' = (the TD's current time)

25. RECEIVE BACnet-SimpleACK-PDU

26. IF (Protocol_Revision is present and Protocol_Revision >= 1) THEN

BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI1),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Toffnormal the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (Poffnormal),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK_NOTIFICATION,

'To State' = OFFNORMAL

ELSE

BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (PI1),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the event-generating object configured for this test),

'Time Stamp' = (Toffnormal the IUT's current time or sequence number),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (Poffnormal),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK_NOTIFICATION

27. TRANSMIT BACnet-SimpleACK-PDU

28. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

7.3.1.11.2 Acked_Transitions Test for Latching Objects

Purpose: To verify that the Acked_Transitions property tracks the acknowledgment state for a transition type.

Test Concept: This test is a single transition test for latching life safety objects which are not able to perform the regular Acked_Transitions test for all transitions. An object, O1, in the IUT is made to generate a transition which requires an acknowledgement. The Acked_Transitions property is verified that the corresponding flag is cleared (set to FALSE). The transition is acknowledged, and the flag is verified to have been set back to TRUE.

Configuration Requirements: O1 is configured to generate events and to require acknowledgements for the transition being tested. O1 should have no event transitions which have outstanding acknowledgements.

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip sending the BACnet-SimpleACK-PDU messages after receiving the notifications.

Test Steps:

1. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
2. MAKE (O1 transition)
3. WAIT (pTimeDelay)
4. **BEFORE Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = (PI1: any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = (T1: any valid time stamp),
'Notification Class' = (NC1: the class corresponding to the object being tested),
'Priority' = (PRIO1: the value configured to correspond to the transition type),
'Event Type' = (E1: any valid event type),
'Message Text' = (optional, any valid message text),
'Notify Type' = (the notify type configured for this event),
'AckRequired' = TRUE,
'From State' = S1,
'To State' = S2,
'Event Values' = (values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentState = S2
7. IF S2 is NORMAL THEN
VERIFY Acked_Transitions = (TRUE, TRUE, FALSE)
VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
ELSE IF S2 is FAULT THEN
VERIFY Acked_Transitions = (TRUE, FALSE, TRUE)
VERIFY Status_Flags = (TRUE, TRUE, ?, ?)
ELSE
VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
8. TRANSMIT AcknowledgeAlarm-Request,
'Acknowledging Process Identifier' = PI1,
'Event Object Identifier' = O1,
'Event State Acknowledged' = S2,
'Time Stamp' = T1,
'Acknowledgement Source' = (a character string),
'Time of Acknowledgment' = (the TD's current time)
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Protocol_Revision is present and Protocol_Revision >= 1) THEN
BEFORE Notification Fail Time
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = PI1,
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = O1,
'Time Stamp' = the IUT's current time or sequence number,
'Notification Class' = NC1,
'Priority' = PRIO1,
'Event Type' = E1,
'Message Text' = (optional, any valid message text),
'Notify Type' = ACK_NOTIFICATION,
'To State' = S2
ELSE
BEFORE Notification Fail Time
RECEIVE ConfirmedEventNotification-Request,
'Process Identifier' = PI1,
'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,
 'Time Stamp' = the IUT's current time or sequence number,
 'Notification Class' = NC1,
 'Priority' = PRIO1,
 'Event Type' = E1,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ACK_NOTIFICATION

11. TRANSMIT BACnet-SimpleACK-PDU

12. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

7.3.1.12 Notify_Type Test

Purpose: To verify that the value of the Notify_Type property determines whether an event notification is transmitted as an alarm or as an event.

Configuration Requirements: The IUT shall be configured with two event-generation objects, E1 and E2. Object E1 shall be configured with a Notify_Type of ALARM and E2 shall be configured with a Notify_Type of EVENT. The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE)

In the test description below X1 and X2 are used to designate the pMonitoredValue algorithm parameter linked to E1 and E2 respectively. X1 and X2 shall be set to a value that results in a NORMAL condition of E1 and E2, respectively.

Notes to Tester: If Notify_Type is writable this test may be performed with one event generating object by changing Notify_Type from ALARM to EVENT in order to cover both cases.

Test Steps:

1. VERIFY (E1), pCurrentState = NORMAL
2. VERIFY (E2), pCurrentState = NORMAL
3. WAIT (pTimeDelay + **Notification Fail Time**)
4. IF (X1 is writable) THEN
 WRITE X1 = (a value that will cause a transition in E1)
 ELSE
 MAKE (X1 a value that will cause a transition in E1)
5. IF (the transition is not a FAULT transition) THEN
 IF (the transition is a TO-OFFNORMAL transition) THEN
 WAIT (pTimeDelay)
 ELSE
 WAIT (pTimeDelayNormal)
6. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (E1),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (any valid value),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = (any valid value),
 'Event Values' = (values appropriate to the event type)
7. TRANSMIT BACnet-SimpleACK-PDU
8. IF (X2 is writable) THEN
 WRITE X2 = (a value that will cause a transition in E2)
 ELSE
 MAKE (X2 a value that will cause a transition in E2)
9. IF (the transition is not a FAULT transition) THEN
 IF (the transition is a TO-OFFNORMAL transition) THEN

WAIT (pTimeDelay)

ELSE

WAIT (pTimeDelayNormal)

10. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (E2),

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (any valid value),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT,

'AckRequired' = TRUE | FALSE,

'From State' = NORMAL,

'To State' = (any valid value),

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

11. TRANSMIT BACnet-SimpleACK-PDU

7.3.1.13 Limit_Enable Tests

7.3.1.13.1 Limit_Enable Test, LowLimitEnable

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

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

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

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

Test Steps:

1. MAKE pLimitEnable = (TRUE, ?)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value less than pLowLimit)
4. WAIT (pTimeDelay)

5. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the object configured for this test),

'Time Stamp' = (the current local time),

'Notification Class' = (the class corresponding to the object being tested),

'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),

'Event Type' = (the algorithm appropriate to the object type i.e., OUT_OF_RANGE, SIGNED_OUT_OF_RANGE, UNSIGNED_OUT_OF_RANGE, or DOUBLE_OUT_OF_RANGE),

'Message Text' = (optional, any valid message text),

- 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = LOW_LIMIT,
 'Event Values' = (values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
 7. VERIFY pCurrentState = LOW_LIMIT
 8. MAKE (pMonitoredValue a value that is between pLowLimit + pDeadband and pHighLimit)
 9. WAIT (pTimeDelayNormal)
 10. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the object configured for this test),
 'Time Stamp' = (the current local time),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (the algorithm appropriate to the object type i.e. OUT_OF_RANGE,
 SIGNED_OUT_OF_RANGE, UNSIGNED_OUT_OF_RANGE, or
 DOUBLE_OUT_OF_RANGE,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = LOW_LIMIT,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)
 11. TRANSMIT BACnet-SimpleACK-PDU
 12. MAKE pLimitEnable = (FALSE, ?)
 13. VERIFY pCurrentState = NORMAL
 14. MAKE (pMonitoredValue a value less than pLowLimit)
 15. WAIT (pTimeDelay + **Notification Fail Time**)
 16. CHECK (verify that no notification message was transmitted)
 17. VERIFY pCurrentState = NORMAL

7.3.1.13.2 Limit_Enable Test, HighLimitEnable

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

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

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

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

Test Steps:

1. MAKE pLimitEnable = (?, TRUE)
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue a value that exceeds pHighLimit)
4. WAIT (pTimeDelay)

5. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the object configured for this test),
 'Time Stamp' = (the current local time),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = (the algorithm appropriate to the object type i.e., OUT_OF_RANGE,
 SIGNED_OUT_OF_RANGE, UNSIGNED_OUT_OF_RANGE, or
 DOUBLE_OUT_OF_RANGE,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = HIGH_LIMIT,
 'Event Values' = (values appropriate to the event type)

6. TRANSMIT BACnet-SimpleACK-PDU

7. VERIFY pCurrentState = HIGH_LIMIT

8. MAKE (pMonitoredValue a value that is between pLowLimit and pHighLimit - pDeadband)

9. WAIT (pTimeDelayNormal)

10. BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the object configured for this test),
 'Time Stamp' = (the current local time),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (the algorithm appropriate to the object type i.e., OUT_OF_RANGE,
 SIGNED_OUT_OF_RANGE, UNSIGNED_OUT_OF_RANGE, or
 DOUBLE_OUT_OF_RANGE,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = HIGH_LIMIT,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)

11. TRANSMIT BACnet-SimpleACK-PDU

12. MAKE pLimitEnable = (2, FALSE)

13. VERIFY pCurrentState = NORMAL

14. MAKE (pMonitoredValue a value that exceeds pHighLimit)

15. WAIT (pTimeDelay + Notification Fail Time)

16. CHECK (verify that no notification message was transmitted)

17. VERIFY pCurrentState = NORMAL

7.3.1.14 Process_Identifier Tests

7.3.1.14.1 Process_Identifier Property Test

Purpose: To verify that the Process_Identifier property correctly supports the required value range. This test shall be performed only if the Protocol_Revision property is present in the Device object and has a value in the range of 1 through 3.

Test Concept: Verify that the Process_Identifier property can be set to any value in the range $0 \dots 2^{32}-1$ by setting it to the minimum, the maximum, and a randomly chosen value.

Test Steps:

1. IF the Process_Identifier property is not writable THEN
MAKE (Process_Identifier = $2^{32}-1$)
ELSE
WRITE Process_Identifier = $2^{32}-1$
2. VERIFY Process_Identifier = $2^{32}-1$
3. IF the Process_Identifier property is not writable THEN
MAKE (Process_Identifier = 0)
ELSE
WRITE Process_Identifier = 0
4. VERIFY Process_Identifier = 0
5. IF the Process_Identifier property is not writable THEN
MAKE (Process_Identifier = (any value in the range 0.. $2^{32}-1$))
ELSE
WRITE Process_Identifier = (any value in the range 0.. $2^{32}-1$)
6. VERIFY Process_Identifier = (the value used in step 5)

7.3.1.14.2 Recipient_List Test

Purpose: To verify that the Process_Identifier contained in a Recipient_List property correctly supports the required value range.

Test Concept: Verify that the Process_Identifier contained in the Recipient_List property can be set to any value in the range 0 .. $2^{32}-1$ by setting it to the minimum, the maximum and a randomly chosen value. This test shall be performed only if the Protocol_Revision property is present in the Device object and has a value greater than or equal to 1.

Test Steps:

1. MAKE (an entry in the Recipient_List = (any valid Valid_Days, any valid time range, any valid Recipient, $2^{32}-1$, TRUE | FALSE depending on the IUT support, any valid transitions bitstring))
2. VERIFY Process_Identifier = $2^{32}-1$
3. MAKE (an entry in the Recipient_List = (any valid Valid_Days, any valid time range, any valid Recipient, 0, TRUE | FALSE depending on the IUT support, any valid transitions bitstring))
4. VERIFY Process_Identifier = 0
5. MAKE (an entry in the Recipient_List = (any valid Valid_Days, any valid time range, any valid Recipient, any value in the range 0.. $2^{32}-1$, TRUE | FALSE depending on the IUT support, any valid transitions Bitstring))
6. VERIFY Process_Identifier = (the value used in step 5)

7.3.1.15 Number_Of_States Range Test

Purpose: This test case verifies that the Present_Value property of a Multi-state Input, Multi-state Output or Multi-state Value object is limited to the range 1 through the value of its Number_Of_States property.

Test Concept: The Number_Of_States property is first verified to be a non-zero value. An attempt is then made to modify the Present_Value property of the referenced object to contain a value greater than the value of the Number_Of_States property. An attempt is then made to set the Present_Value property to 0. The test shall verify a correct error response for each case.

Configuration Requirements: The IUT shall be configured to contain a Multi-state Input, Multi-state Output, or Multi-state Value object, Object1, which contains a writable Present_Value property. Object1 shall be configured such that the Present_Value property is writable before executing the test. If the Present_Value property cannot be made writable, then this test shall be skipped.

Test Steps:

1. READ COUNT = Number_Of_States
2. VERIFY Number_Of_States = (a non-zero value)
3. TRANSMIT WriteProperty-Request,
'Object Identifier' = Object1,
'Property Identifier' = Present_Value,

- 'Property Value' = (any value larger than COUNT)
4. RECEIVE BACnet-Error-PDU,
Error Class = PROPERTY,
Error Code = VALUE_OUT_OF_RANGE
 5. VERIFY (Object1), P1 = (a value between 1 and COUNT, inclusive)
 6. TRANSMIT WriteProperty-Request,
'Object Identifier' = Object1,
'Property Identifier' = Present_Value,
'Property Value' = 0
 7. RECEIVE BACnet-Error-PDU,
Error Class = PROPERTY,
Error Code = VALUE_OUT_OF_RANGE
 8. VERIFY (Object1), P1 = (a value between 1 and COUNT, inclusive)

Note To Tester: When testing an object that is commandable, any priority may be selected.

7.3.1.16 Array Resizing Test

The test in this clause shall be applied to resizable arrays in devices claiming Protocol_Revision 4 or higher. It may be applied to resizable arrays in devices claiming Protocol_Revision 3 or lower, but only where conformance to the rules on resizing arrays of Protocol_Revision 4 is claimed.

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

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

Test Steps:

1. WRITE (the array property being tested) = (array of non-zero size N_1)
2. VERIFY (array is as written in step 1)
3. WRITE (the array property being tested) = (array of non-zero size N_2 , where $N_2 \leq N_1$)
4. VERIFY (array is as written in step 3)
5. WRITE (the array property being tested) = (array of non-zero size N_3 , where $N_3 \geq N_1$)
6. VERIFY (array is as written in step 5)
7. WRITE (the array property being tested) = (a non-zero unsigned value N_4 , where $N_4 \leq N_1$), ARRAY INDEX = 0
8. VERIFY (array contains first N_4 elements of the array written in step 5)
9. WRITE (the array property being tested) = (N_5 , where $N_5 \geq N_4$), ARRAY INDEX = 0
10. VERIFY (array contains first N_4 elements of the array written in step 5, plus $N_5 - N_4$ additional elements, initialized to particular values if specified for the array property being tested)
11. TRANSMIT WriteProperty-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = (the array property being tested),
'Property Array Index' = (N_6 , where $N_6 \geq N_5$),
'Property Value' = (one array element)
12. RECEIVE BACnet-Error-PDU
Error Class = PROPERTY,
Error Code = INVALID_ARRAY_INDEX
13. VERIFY (array is unchanged from step 10)
14. IF (the array can be resized to have zero elements) THEN
WRITE (the array property being tested) = (empty array)
VERIFY (array is empty)

7.3.1.17 Event_Message_Texts Tests

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

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

Configuration Requirements: The IUT shall be configured with an event-generation object, O1 which shall be in a NORMAL Event_State at the beginning of the test, and if Event_Enable is configurable, it shall have all bits set to TRUE for which the object supports transitions.

Test Steps:

1. READ EMT = Event_Message_Texts
2. IF (Event_Enable is (TRUE, ?, ?) and O1 can generate TO_OFFNORMAL transitions) THEN {
 - IF (pMonitoredValue is writable) THEN
 - WRITE pMonitoredValue = (a value that is offnormal)
 - ELSE
 - MAKE (pMonitoredValue a value that is offnormal)
 - WAIT (pTimeDelay)
 - BEFORE **Notification Fail Time**
 - RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid timestamp),
 - 'Notification Class' = (the class corresponding to the object being tested),
 - 'Priority' = (the configured TO_OFFNORMAL priority),
 - 'Event Type' = (any valid event type),
 - 'Message Text' = (M: any valid value placed into EMT[1]),
 - 'Notify Type' = Notify_Type,
 - 'AckRequired' = (the configured value for the TO_OFFNORMAL transition),
 - 'From State' = NORMAL,
 - 'To State' = (any valid offnormal state),
 - 'Event Values' = (values appropriate to the event type)
 - VERIFY Event_Message_Texts = EMT
3. IF (Event_Enable is (?, ?, TRUE) and O1 can generate TO_NORMAL transitions) THEN {
 - IF (pMonitoredValue is writable) THEN
 - WRITE pMonitoredValue = (a value that will result in a TO_NORMAL transition)
 - ELSE
 - MAKE (pMonitoredValue a value that will result in a TO_NORMAL transition)
 - WAIT (pTimeDelayNormal)
 - BEFORE **Notification Fail Time**
 - RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the class corresponding to the object being tested),
 - 'Priority' = (the configured TO_NORMAL priority),
 - 'Event Type' = (any valid event type),
 - 'Message Text' = (M: any valid value placed into EMT[3]),
 - 'Notify Type' = Notify_Type,
 - 'AckRequired' = (the configured value for the TO_NORMAL transition),
 - 'From State' = (any valid value),
 - 'To State' = NORMAL,
 - 'Event Values' = (values appropriate to the event type)
 - VERIFY Event_Message_Texts = EMT
4. IF (Event_Enable is (?, TRUE, ?) and O1 can generate TO_FAULT transitions) THEN {
 - MAKE (a condition exist that will cause O1 to generate a TO_FAULT transition)
 - BEFORE **Notification Fail Time**

```

RECEIVE ConfirmedEventNotification-Request,
  'Process Identifier' = (any valid process ID),
  'Initiating Device Identifier' = IUT,
  'Event Object Identifier' = O1,
  'Time Stamp' = (any valid time stamp),
  'Notification Class' = (the class corresponding to the object being tested),
  'Priority' = (the configured TO_FAULT priority),
  'Event Type' = (IF (Protocol_Revision < 13 THEN
    (any valid event type),
    ELSE
    (CHANGE_OF_RELIABILITY,
  'Message Text' = (M: any valid value placed into EMT[2]),
  'Notify Type' = Notify_Type,
  'AckRequired' = (the configured value for the TO_FAULT transition),
  'From State' = (any valid value),
  'To State' = FAULT,
  'Event Values' = (values appropriate to the event type)
VERIFY Event_Message_Texts = EMT
}

```

7.3.1.18 Event_Message_Texts_Config Test

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

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

Configuration Requirements: If possible, configure each entry in the Event_Message_Texts_Config property of Object O1 to be distinct. ES1 shall be the state to which O1 transitions. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. ESINDEX shall be the array index associated with ES1 (1 for offnormal states, 2 for FAULT, and 3 for NORMAL). The notification class for O1 is configured for UnconfirmedEventNotification.

Test Steps:

1. MAKE (a condition exist which will cause O1 to transition to ES1)
2. WAIT D1
3. BEFORE **Notification Fail Time**

```

RECEIVE UnconfirmedEventNotification-Request
  'Process Identifier' = (any valid process identifier),
  'Initiating Device Identifier' = IUT,
  'Event Object Identifier' = O1,
  'Time Stamp' = (any valid time stamp),
  'Notification Class' = (the notification class configured for O1),
  'Priority' = (any valid priority),
  'Event Type' = (any standard event type),
  'Message Text' = T1,
  'Notify Type' = ALARM | EVENT,
  'AckRequired' = TRUE | FALSE,
  'From State' = (any valid event state),
  'To State' = ES1,
  'Event Values' = (any values appropriate to the event type)

```

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

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

7.3.1.19 Event_Algorithm_Inhibit Tests

7.3.1.19.1 Event_Algorithm_Inhibit Test

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

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

Configuration Requirements: O1 is configured to detect and report unconfirmed events, is in the NORMAL state, and, if supported, is configured to detect fault conditions. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Event_Algorithm_Inhibit = FALSE
3. MAKE (a condition exist which will result in a transition of O1. If possible, 'To State' shall be an offnormal event state)
4. WAIT D1
5. BEFORE **Notification Fail Time**
 - RECEIVE UnconfirmedEventNotification-Request
 - 'Process Identifier' = (PID1: any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the notification class configured for O1),
 - 'Priority' = (the value configured for the transition),
 - 'Event Type' = (ET1: any valid event type),
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = (value from the Notify_Type property configured for O1),
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = (ES1: any event state appropriate to the event type),
 - 'Event Values' = (any values appropriate to the event type)
6. WRITE Event_Algorithm_Inhibit = TRUE
7. IF (ES1 <> NORMAL) THEN
 - BEFORE **Notification Fail Time**
 - RECEIVE UnconfirmedEventNotification-Request
 - 'Process Identifier' = PID1,
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the notification class configured for O1),
 - 'Priority' = (the value configured for the transition),
 - 'Event Type' = ET1,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = (value from the Notify_Type property configured for O1),
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = ES1,
 - 'To State' = NORMAL,
 - 'Event Values' = (any values appropriate to the event type)
8. VERIFY pCurrentState = NORMAL
9. MAKE (a condition exist which would result in a transition of O1 other than TO-FAULT, if Event_Algorithm_Inhibit were FALSE)

10. WAIT D1

11. WAIT **Notification Fail Time**

12. CHECK (that the IUT did not send any event notifications for O1)

13. VERIFY pCurrentState = NORMAL

14. IF (O1 supports fault detection) THEN
 MAKE (a fault condition exist for O1)
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedEventNotification-Request
 'Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the notification class configured for O1),
 'Priority' = (the value configured for the transition),
 'Event Type' = CHANGE_OF_RELIABILITY,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = (value from the Notify_Type property configured for O1),
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = FAULT,
 'Event Values' = (any values appropriate for CHANGE_OF_RELIABILITY)
 MAKE (remove the fault condition)
 WAIT (pTimeDelayNormal)
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedEventNotification-Request
 'Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the notification class configured for O1),
 'Priority' = (the value configured for the transition),
 'Event Type' = CHANGE_OF_RELIABILITY,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = (value from the Notify_Type property configured for O1),
 'AckRequired' = TRUE | FALSE,
 'From State' = FAULT,
 'To State' = NORMAL,
 'Event Values' = (any values appropriate for CHANGE_OF_RELIABILITY)

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

7.3.1.19.2 Event_Algorithm_Inhibit Summarization Test

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

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

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

Test Steps:

1. AT = READ Acked_Transitions

2. CHECK (AT $\triangleleft$ (T, T, T))
2. VERIFY Acked_Transitions = (?, T, ?)
3. VERIFY Event_Algorithm_Inhibit = TRUE
4. TRANSMIT GetEventInformation
5. RECEIVE GetEventInformation-Ack,
 'List of Event Summaries' = (list of object identifiers which includes O1)
 'More Events' = FALSE

7.3.1.19.3 Event_Algorithm_Inhibit Acknowledgement Test

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

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

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

Test Steps:

1. AT = READ Acked_Transitions
2. CHECK (AT $\triangleleft$ (T, T, T))
3. VERIFY Event_Algorithm_Inhibit = TRUE
4. TRANSMIT AcknowledgeAlarm
 'Acknowledging Process Identifier' = (any valid value),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = ES_TO_ACK,
 'Time Stamp' = TS_TO_ACK,
 'Time of Acknowledgment' = (the current timestamp)
5. RECEIVE BACnet-SimpleACK-PDU
6. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (the configured process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class configured for O1),
 'Priority' = (the value configured for the transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ACK_NOTIFICATION,
 'To State' = ES_TO_ACK
7. TRANSMIT BACnet-SimpleACK-PDU
8. AT2 = READ Acked_Transitions
9. CHECK (AT2 is equal to AT, except the bit associated with ES_TO_ACK is TRUE)

7.3.1.20 Event_Algorithm_Inhibit_Ref Tests

7.3.1.20.1 Event_Algorithm_Inhibit_Ref Test

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

Test Concept: Execute test 7.3.1.19.1 against an object, O2, which supports both Event_Algorithm_Inhibit_Ref and Event_Algorithm_Inhibit, and instead of writing Event_Algorithm_Inhibit, write the property referenced by Event_Algorithm_Inhibit_Ref to change the value in the Event_Algorithm_Inhibit property.

Configuration Requirements: If the IUT has no object which has an Event_Algorithm_Inhibit_Ref property, this test shall be skipped.

7.3.1.20.2 Event_Algorithm_Inhibit Writable Test

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

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

Test Steps:

1. WRITE Event_Algorithm_Inhibit = TRUE
2. WRITE Event_Algorithm_Inhibit = FALSE

7.3.1.21 Reliability_Evaluation_Inhibit Tests

7.3.1.21.1 Reliability_Evaluation_Inhibit Test

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

Test Concept: Select an event generating object, O1, which supports the Reliability_Evaluation_Inhibit property. With Reliability_Evaluation_Inhibit FALSE, make a fault condition exist. Verify that Reliability changes and, if event reporting is supported, that a notification is generated. Set Reliability_Evaluation_Inhibit to TRUE. Verify that the Reliability changes to NO_FAULT_DETECTED and, if event reporting is supported, that a TO_NORMAL notification is generated. Remove the fault condition and ensure that no notification is generated. Make a fault condition exist and verify that Reliability remains NO_FAULT_DETECTED, and that no notification is generated.

Configuration Requirements: O1 is configured to detect and, if event reporting is supported, report unconfirmed events, is in the NORMAL state, has Out_Of_Service set to FALSE and Reliability_Evaluation_Inhibit equals FALSE, so that reliability evaluation for that object is configured to detect fault conditions. If no object exists in the IUT for which fault conditions can be generated and which meets these configuration requirements then this test shall be skipped.

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

Test Steps:

1. VERIFY Out_Of_Service = FALSE
2. VERIFY pCurrentState = NORMAL
3. VERIFY Reliability = NO_FAULT_DETECTED
4. MAKE(a fault condition exist for O1)
5. IF the IUT supports event reporting THEN
 - BEFORE Notification Fail Time
 - RECEIVE UnconfirmedEventNotification-Request
 - 'Process Identifier' = (the value configured for the transition),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid timestamp),
 - 'Priority' = (any valid priority),
 - 'Event Type' = CHANGE_OF_RELIABILITY,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = ALARM | EVENT,
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = FAULT,

- 'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)
6. VERIFY Reliability ∇ NO_FAULT_DETECTED
 7. IF Reliability_Evaluation_Inhibit is writable THEN
 WRITE Reliability_Evaluation_Inhibit = TRUE
ELSE
 MAKE(Reliability_Evaluation_Inhibit TRUE)
 8. IF the IUT supports event reporting THEN
 BEFORE **Internal Processing Fail Time + Notification Fail Time**
 RECEIVE UnconfirmedEventNotification-Request
 'Process Identifier' = (the value configured for the transition),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid timestamp),
 'Priority' = (any valid priority),
 'Event Type' = CHANGE_OF_RELIABILITY,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = FAULT,
 'To State' = NORMAL,
 'Event Values' = (any values appropriate to CHANGE_OF_RELIABILITY)
 9. VERIFY Reliability = NO_FAULT_DETECTED
 10. VERIFY pCurrentState = NORMAL
 11. MAKE (remove the fault condition)
 12. WAIT (pTimeDelayNormal)
 13. WAIT **Notification Fail Time**
 14. CHECK (that the IUT did not send any event notifications for O1)
 15. VERIFY Reliability = NO_FAULT_DETECTED
 16. MAKE (a fault condition exist for O1)
 17. WAIT **Notification Fail Time**
 18. VERIFY Reliability = NO_FAULT_DETECTED
 19. VERIFY pCurrentState = NORMAL
 20. CHECK (that the IUT did not send any event notifications for O1)

7.3.1.21.2 Reliability_Evaluation_Inhibit Summarization Test

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

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

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

Test Steps:

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

7.3.1.22 Event_Detection_Enable Tests

7.3.1.22.1 Event_Detection_Enable Inhibits Event Generation

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

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

Configuration Requirements: O1 is configured to detect and report unconfirmed events and requires acknowledgments for all transitions. Event_Detection_Enable is equal to TRUE. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. For this test, NO_TS equals a BACnetDateTime with all unspecified values, a BACnet Time with all unspecified values, or a sequence number of 0.

Test Steps:

1. VERIFY Event_Detection_Enable = TRUE
2. MAKE (a condition exist which will cause O1 to transition, to an offnormal state if possible)
3. WAIT D1
4. BEFORE **Notification Fail Time**
 - RECEIVE UnconfirmedEventNotification-Request
 - 'Process Identifier' = (any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the notification class configured for O1),
 - 'Priority' = (the value configured for the transition),
 - 'Event Type' = (any valid event type),
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = (value from the Notify_Type property configured for O1),
 - 'AckRequired' = TRUE,
 - 'From State' = (any valid event state),
 - 'To State' = (any event state appropriate to the event type),
 - 'Event Values' = (any values appropriate to the event type)
5. IF Event_Detection_Enable is writable THEN
 - WRITE Event_Detection_Enable = FALSE
 - ELSE
 - MAKE (Event_Detection_Enable to FALSE. This property is expected to be set during system configuration and is not expected to change dynamically.)
6. WAIT (D1 + **Notification Fail Time** + **Internal Processing Fail Time**)
7. CHECK (that the IUT did not send any further event notifications for O1)
8. VERIFY pCurrentState = NORMAL
9. VERIFY Acked_Transitions = (T,T,T)
10. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 - VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
11. IF (Event_Message_Texts property exists) THEN
 - VERIFY Event_Message_Texts = ["", "", ""]
12. MAKE (a condition exist which would cause O1 to transition, if Event_Detection_Enable were TRUE)
13. WAIT (D1 + **Notification Fail Time**)
14. CHECK (that the IUT did not send any event notifications for O1)
15. VERIFY pCurrentState = NORMAL
16. VERIFY Acked_Transitions = (T,T,T)
17. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 - VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
18. IF (Event_Message_Texts property exists) THEN
 - VERIFY Event_Message_Texts = ["", "", ""]

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

7.3.1.22.2 Event_Detection_Enable Inhibits FAULT

Purpose: To verify that Event_Detection_Enable disables fault reporting.

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

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

Test Steps:

1. VERIFY Event_Detection_Enable = FALSE
2. IF Reliability is writable THEN
3. WRITE Reliability = (any value other than NO_FAULT_DETECTED)
4. ELSE
5. MAKE (a condition exist which would cause O1 to transition to FAULT, if Event_Detection_Enable were TRUE)
6. WAIT **Notification Fail Time**
7. CHECK (that the IUT did not send any event notifications due to this condition)
8. VERIFY pCurrentState = NORMAL
9. VERIFY Acked_Transitions = (T,T,T)
10. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
11. VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
12. IF (Event_Message_Texts property exists) THEN
13. VERIFY Event_Message_Texts = ["", "", ""]

7.3.1.23 Array Resizing Test using WritePropertyMultiple Service

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

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

Test Steps:

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

--Resize the array to make it smaller in size

4. TRANSMIT WritePropertyMultiple-Request,
 'Object Identifier' = O1,
 'Property Identifier' = P1
 'Property Value' = (array A2 of non-zero size N2, where N2 <= N1)

5. RECEIVE BACnet-SimpleACK-PDU
6. VERIFY P1 = (array A2), ARRAY INDEX = 0, (array size N2)

--Resize the array to make it larger in size

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

--Modify the existing content of element

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

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

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

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

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

--Resize the array to size zero

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

7.3.1.24 Non-zero Writable State Count Test

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

Test Concept: The `Change_of_State_Count` property is set with a non-zero value. The `Time_Of_State_Count_Reset` property is checked to verify that it has not been updated. The `Time_Of_State_Count_Reset` property is then checked to be writable.

Configuration Requirements: The object being tested shall contain `Change_of_State_Count` and `Time_Of_State_Count_Reset` properties, and its `Change_of_State_Count` property must accept writes of non-zero values.

Test Steps:

1. READ TSCR = `Time_Of_State_Count_Reset`
2. WRITE `Change_of_State_Count` = (a value > 0)
3. VERIFY `Time_Of_State_Count_Reset` = TSCR
4. WRITE `Time_Of_State_Count_Reset` = (T1: any valid value)
5. VERIFY `Time_Of_State_Count_Reset` = T1

7.3.1.25 Non-zero Writable Elapsed Active Time Test

Purpose: To verify that `Time_Of_Active_Time_Reset` is writable when `Elapsed_Active_Time` accepts writes of non-zero values and does not automatically change when `Elapsed_Active_Time` is written to a non-zero value.

Test Concept: The `Present_Value` or `Feedback_Value` is made INACTIVE and the `Elapsed_Active_Time` is set with a non-zero value. The `Time_Of_Active_Time_Reset` property is checked to verify that it has not been updated. The `Time_Of_Active_Time_Reset` property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which `Elapsed_Active_Time` and `Time_Of_Active_Time_Reset` properties are present, and in which the `Elapsed_Active_Time` property accepts writes of non-zero values.

Test Steps:

1. IF (`Present_Value` is writable) THEN
 WRITE `Present_Value` = INACTIVE
 VERIFY `Present_Value` = INACTIVE
 ELSE
 MAKE (`Present_Value` = INACTIVE)
2. READ TATR = `Time_Of_Active_Time_Reset`
3. WRITE `Elapsed_Active_Time` = a supported non-zero value
4. VERIFY `Time_Of_Active_Time_Reset` = TATR
5. WRITE `Time_Of_Active_Time_Reset` = (T1: any valid value)
6. VERIFY `Time_Of_Active_Time_Reset` = T1

7.3.1.26 Strike Count Tests**7.3.1.26.1 Non-zero Writable Strike Count Test**

Purpose: To verify that `Time_Of_Count_Reset` is writable when `Strike_Count` accepts writes of non-zero values and does not automatically change when `Strike_Count` is written to a non-zero value.

Test Concept: The `Strike_Count` property is set with a non-zero value. The `Time_Of_Strike_Count_Reset` property is checked to verify that it has not been updated. The `Time_Of_Strike_Count_Reset` property is then checked to be writable.

Configuration Requirements: The object being tested shall be chosen in which `Strike_Count` and `Time_Of_Strike_Count_Reset` properties are present, and in which the `Strike_Count` property accepts writes of non-zero values.

Test Steps:

1. READ TSCR = `Time_Of_Strike_Count_Reset`

2. WRITE Strike_Count = (a value > 0)
3. VERIFY Time_Of_Strike_Count_Reset = TSCR
4. WRITE Time_Of_Strike_Count_Reset = (T1: any valid value)
5. VERIFY Time_Of_Strike_Count_Reset = T1

7.3.1.26.2 Strike Count Test

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

Test Concept: The Present_Value or Feedback_Value of O1 can be used as the source S1 to increment Strike_Count. S1 is transitioned from OFF to ON. The Strike_Count property is checked to verify that it has been incremented. The Strike_Count is reset and Time_Of_Strike_Count_Reset is checked to verify that it has been updated appropriately. Strike_Count is set to a non-zero value and the Time_Of_Strike_Count_Reset is unchanged.

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

Test Steps:

1. C1 = Strike_Count
2. MAKE (S1 transition OFF to ON)
3. VERIFY (Strike_Count = C1 + 1)
4. IF (Strike_Count is writable) THEN
MAKE (Strike_Count = 0)
VERIFY (Time_Of_Strike_Count_Reset = current local time)
5. IF (Strike_Count is writable to a non-zero value) THEN
MAKE (Strike_Count > 0)
VERIFY (Time_Of_Strike_Count_Reset is unchanged)

7.3.1.27 Blink Warn Tests

7.3.1.27.1 Blink-Warn WARN Command Test

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

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Active is FALSE.

	Binary object	Lighting Output	Lighting Output object
PROP_REF	Present_Value		Present_Value or Lighting_Command
C1	WARN		-1.0 if PROP_REF = Present_Value, otherwise WARN
V1	ON		>1.0

Test Steps:

1. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Active = FALSE
4. WRITE PROP_REF = C1, PRIORITY = PTY1
5. BEFORE **Internal Processing Fail Time**
CHECK (blink-warn occurred)
6. VERIFY Egress_Active = FALSE
7. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1

7.3.1.27.2 Blink-Warn WARN_OFF Command Test

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

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN_OFF command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority_Array value at priority PTY1 after Egress_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Active is FALSE, and Egress_Time is a non-zero value.

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

Test Steps:

1. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
8. WHILE (Egress_Active = TRUE)
WAIT 1s
9. T2 = current local time
10. CHECK (blink warn occurred)
11. VERIFY Egress_Time ≈ (T2 - T1)
12. VERIFY Priority_Array = V2, ARRAY INDEX = PTY1

7.3.1.27.3 Blink-Warn WARN_RELINQUISH Command Test

Purpose: To verify the correct operation of the blink-warn WARN_RELINQUISH commands.

Test Concept: Select an object O1 that supports blink-warn commands. Ensure O1 is not in egress mode and the specific properties have been configured to support blink-warn. Execute blink-warn WARN_RELINQUISH command by writing C1 to PROP_REF at a priority PTY1 of O1 and validate the specified blink-warn command functions correctly. Validate the Priority_Array value at priority PTY1 after Egress_Time seconds has elapsed.

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, and Egress_Active is FALSE.

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

Test Steps:

1. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1

2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
8. WHILE (Egress_Active = TRUE)
 - WAIT 1s
9. T2 = current local time
10. CHECK (blink warn occurred)
11. VERIFY Egress_Time $\approx$ (T2 – T1)
12. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

7.3.1.27.4 Blink-Warn STOP Command Test

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

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. The Priority_Array at PTY1 has a value V1, Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, and Egress_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present Value	Lighting Command
C1	WARN_RELINQUISH or WARN_OFF	WARN_RELINQUISH or WARN_OFF
V0	NULL or OFF	NULL or 0.0
V1	ON	>1.0
V2	OFF	0.0

Test Steps:

1. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. T1 = current local time
7. BEFORE **Internal Processing Fail Time**
 - CHECK (blink-warn occurred)
8. VERIFY Egress_Active = TRUE
9. WAIT less than Egress_Time
 - WRITE PROP_REF = STOP, PRIORITY = PTY1
10. T2 = current local time
11. WAIT **Internal Processing Fail Time**
12. VERIFY Egress_Active = FALSE
13. WAIT Egress_Time – (T2 – T1) + **Internal Processing Fail Time**
14. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1

7.3.1.27.5 Blink-Warn WARN Command Failure Test

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

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

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

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

Test Steps:

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

1. VERIFY Blink_Warn_Enable = TRUE
2. WRITE Present_Value = V1, PRIORITY = PTY1
3. VERIFY Egress_Active = FALSE
4. WRITE Present_Value = V2, PRIORITY = PTY2
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
7. VERIFY Egress_Active = FALSE
8. VERIFY Priority_Array = V1, ARRAY INDEX = PTY1
9. WRITE Present_Value = NULL, PRIORITY = PTY2

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

10. WRITE Present_Value = V3, PRIORITY = PTY1
11. WRITE PROP_REF = C1, PRIORITY = PTY1
12. WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
13. VERIFY Egress_Active = FALSE
14. VERIFY Priority_Array = V3, ARRAY INDEX = PTY1
15. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for Blink_Warn_Enable is FALSE

16. IF (Blink_Warn_Enable is writable) THEN
WRITE Blink_Warn_Enable = FALSE
WRITE PROP_REF = C1, PRIORITY = PTY1
WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
VERIFY Egress_Active = FALSE
VERIFY Priority_Array = V1, ARRAY INDEX = PTY1

7.3.1.27.6 Blink-Warn WARN_OFF Command Failure Test

Purpose: To verify blink-warn WARN_OFF command does not occur when the specified priority is not the highest active priority, the Present_Value is either 0.0 or OFF, or Blink_Warn_Enable is FALSE.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value and Egress_Active is FALSE.

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

Test Steps:

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

1. VERIFY Blink_Warn_Enable = TRUE
2. VERIFY Egress_Time > 0
3. WRITE Present_Value = V1, PRIORITY = PTY1
4. VERIFY Egress_Active = FALSE
5. WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP_REF = C1, PRIORITY = PTY1
7. WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_Array = V3, ARRAY INDEX = PTY1
10. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for the Present_Value is OFF or 0.0

11. WRITE Present_Value = V3, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
12. WRITE PROP_REF = C1, PRIORITY = PTY1
13. WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
14. VERIFY Egress_Active = FALSE
15. VERIFY Priority_Array = V3, ARRAY INDEX = PTY1
16. WRITE Present_Value = NULL, PRIORITY = PTY2
17. WRITE Present_Value = V1, PRIORITY = PTY1

-- Test for Blink_Warn_Enable is FALSE

18. IF (Blink_Warn_Enable is writable) THEN
WRITE Blink_Warn_Enable = FALSE
WRITE PROP_REF = C1, PRIORITY = PTY1
WAIT **Internal Processing Fail Time**
CHECK (blink-warn did not occur)
VERIFY Egress_Active = FALSE
VERIFY Priority_Array = V3, ARRAY INDEX = PTY1

7.3.1.27.7 Blink-Warn WARN_RELINQUISH Command Failure Test

Purpose: To verify blink-warn WARN_RELINQUISH command does not occur when the specified priority is not the highest active priority, the value at the specified priority is V0, the value of the next highest non-NULL priority, including Relinquish_Default, is V1, or Blink_Warn_Enable is FALSE.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 shall have a value of NULL, at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V2, and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, Egress_Active is FALSE.

	Binary Lighting Output object	Lighting Output object
PROP_REF	Present_Value	Present_Value or Lighting_Command
C1	WARN_RELINQUISH	-2.0 if PROP_REF = Present_Value, otherwise WARN_RELINQUISH
V0	OFF or NULL	0.0 or NULL

V1	ON	>1.0
V2	OFF	0.0

Test Steps:

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

1. VERIFY Blink_Warn_Enable = TRUE
2. VERIFY Egress_Time > 0
3. WRITE Present_Value = V1, PRIORITY = PTY1
4. VERIFY Egress_Active = FALSE
5. WRITE Present_Value = V1, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
6. WRITE PROP_REF = C1, PRIORITY = PTY1
7. **WAIT Internal Processing Fail Time**
CHECK (blink-warn did not occur)
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

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

11. WRITE Present_Value = V2 PRIORITY = PTY1
12. WRITE PROP_REF = C1, PRIORITY = PTY1
13. **WAIT Internal Processing Fail Time**
CHECK (blink-warn did not occur)
14. VERIFY Egress_Active = FALSE
15. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

-- Test for the value at the specified priority is NULL

16. WRITE Present_Value = NULL, PRIORITY = PTY1
17. WRITE PROP_REF = C1, PRIORITY = PTY1
18. **WAIT Internal Processing Fail Time**
CHECK (blink-warn did not occur)
19. VERIFY Egress_Active = FALSE
20. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

-- Test for the value of the next highest non-NULL priority is ON or > 1.0

21. WRITE Present_Value = V1 PRIORITY = PTY1
22. WRITE Present_Value = V1, PRIORITY = PTY3, a value numerically greater than PTY1
23. WRITE PROP_REF = C1, PRIORITY = PTY1
24. **WAIT Internal Processing Fail Time**
CHECK (blink-warn did not occur)
25. VERIFY Egress_Active = FALSE
26. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
27. WRITE Present_Value = NULL, PRIORITY = PTY3

-- Test for the value of Relinquish_Default is ON or > 1.0

28. IF (Relinquish_Default is writable) THEN
WRITE Present_Value = V1, PRIORITY = PTY1
WRITE Relinquish_Default = V1
WRITE PROP_REF = C1, PRIORITY = PTY1
WAIT Internal Processing Fail Time
CHECK (blink-warn did not occur)
VERIFY Egress_Active = FALSE
VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
WRITE Relinquish_Default = V2

-- Test for Blink_Warn_Enable is FALSE

29. IF (Blink_Warn_Enable is writable) THEN
WRITE Present_Value = V1, PRIORITY = PTY1
WRITE Blink_Warn_Enable = FALSE
WRITE PROP_REF = C1, PRIORITY = PTY1

WAIT Internal Processing Fail Time

CHECK (blink-warn did not occur)

VERIFY Egress_Active = FALSE

VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1

7.3.1.27.8 Blink-Warn WARN_OFF Command Halted Test

Purpose: To verify blink-warn WARN_OFF execution is halted when a higher priority entry is written or the Present_Value at the specified priority is changed.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and no internal algorithms are issuing commands to O1 at a priority numerically less than or equal to PTY1. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value and Egress_Active is FALSE.

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

Test Steps:

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

1. WRITE Present_Value = V1, PRIORITY = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**
CHECK (blink-warn occurred)
7. BEFORE Egress_Active = FALSE
WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_Array = V4, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the Present_Value at the specified property is changed

11. WRITE Present_Value = V1, PRIORITY = PTY1
12. VERIFY Blink_Warn_Enable = TRUE
13. VERIFY Egress_Time > 0
14. VERIFY Egress_Active = FALSE
15. WRITE PROP_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**
CHECK (blink-warn occurred)
17. BEFORE Egress_Active = FALSE
WRITE Present_Value = V3, PRIORITY = PTY1
18. VERIFY Egress_Active = FALSE
19. VERIFY Priority_Array = V3, ARRAY INDEX = PTY1

7.3.1.27.9 Blink-Warn WARN_RELINQUISH Command Halted Test

Purpose: To verify blink-warn WARN_RELINQUISH execution is halted when a higher priority entry is written or the Present_Value at the specified priority is changed.

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

Configuration Requirements: O1 shall be configured such that all slots in the Priority_Array numerically less than PTY1 have a value of NULL and at least one slot numerically greater than PTY1 or Relinquish_Default shall have a value of V1 and all other slots numerically greater than PTY1 shall have a value of V0. No internal algorithms are issuing commands to O1 at any priority. Blink_Warn_Enable is TRUE, Egress_Time is a non-zero value, Egress_Active is FALSE.

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

Test Steps:

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

1. WRITE Present_Value = V2, PRIORITY = PTY1
2. VERIFY Blink_Warn_Enable = TRUE
3. VERIFY Egress_Time > 0
4. VERIFY Egress_Active = FALSE
5. WRITE PROP_REF = C1, PRIORITY = PTY1
6. BEFORE **Internal Processing Fail Time**
CHECK (blink-warn occurred)
7. BEFORE Egress_Active = FALSE
WRITE Present_Value = V2, PRIORITY = PTY2, a value not equal to 6 and less than PTY1
8. VERIFY Egress_Active = FALSE
9. VERIFY Priority_Array = NULL, ARRAY INDEX = PTY1
10. WRITE Present_Value = NULL, PRIORITY = PTY2

-- Test for the Present_Value at the specified property is changed

11. WRITE Present_Value = V2, PRIORITY = PTY1
12. VERIFY Blink_Warn_Enable = TRUE
13. VERIFY Egress_Time > 0
14. VERIFY Egress_Active = FALSE
15. WRITE PROP_REF = C1, PRIORITY = PTY1
16. BEFORE **Internal Processing Fail Time**
CHECK (blink-warn occurred)
17. BEFORE Egress_Active = FALSE
WRITE Present_Value = V3, PRIORITY = PTY1
18. VERIFY Egress_Active = FALSE
19. VERIFY Priority_Array = V3, ARRAY INDEX = PTY1

7.3.1.28 Value Source Mechanism Tests

7.3.1.28.1 Writing to the Value_Source Property by a Device Other than the Device that Commanded the Object

Purpose: To verify the IUT correctly refuses an attempt to write a Value_Source property by a device other than the device that most recently commanded the object.

Test Concept: Command an object, O1, that supports the value source mechanism, from device D1, and verify the updated Value_Source. Attempt to write to the Value_Source property from device D2. Verify that an error is returned and Value_Source does not change.

Test Steps:

1. TRANSMIT WriteProperty-Request,
SOURCE = D1,
'Object Identifier' = O1,
'Property Identifier' = (P1: the property monitored by the Value_Source mechanism for this object type),
'Priority' = (PRIO: absent or any value other than 6)
'Property Value' = (X2: any valid value)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (O1 is commandable) THEN
VERIFICATION Priority_Array = X2, ARRAY_INDEX = PRIO
ELSE
VERIFICATION (P1) = X2
4. VERIFICATION Value_Source = (D1's device identifier or network address)
5. TRANSMIT WriteProperty-Request,
SOURCE = D2,
'Object Identifier' = O1,
'Property Identifier' = Value_Source,
'Priority' = PRIO,
'Property Value' = (any valid value)
6. RECEIVE BACnet-Error PDU,
'Error Class' = PROPERTY,
'Error Code' = WRITE_ACCESS_DENIED
7. VERIFICATION Value_Source = (D1's device identifier or network address)

7.3.1.28.2 Non-commandable Value_Source Property Test

Purpose: To verify that the Value_Source property indicates the source of the current Present_Value in a non-commandable object.

Test Concept: Select a non-commandable object with a writable Present_Value which supports the Value Source mechanism. Present_Value is written, and it is verified that Value_Source is updated appropriately. Value_Source is then written to verify that the last writer can update it.

Test Steps:

1. WRITE Present_Value = V1
2. VERIFICATION Present_Value = V1
3. VERIFICATION Value_Source = (TD's device identifier or network address)
4. WRITE Value_Source = (any valid value, V2)
5. VERIFICATION Value_Source = V2

7.3.1.28.3 Value_Source Property None Test

Purpose: To verify that the Value_Source property shall have the value 'None' when there is no active value source.

Test Concept: If there is no active value source, i.e., the Present_Value has taken on the value of Relinquish_Default, then the Value_Source property shall have the value 'None'.

Configuration Requirements: The object (O1) to be tested shall have 1 non-NULL entry in its Priority_Array and the Current_Command_Priority has a value other than NULL or 6.

Test Steps:

1. READ PRIO = Current_Command_Priority
2. CHECK(PRIO <> 6 and PRIO <> NULL)
3. VERIFICATION Value_Source = (is not 'None')
4. WRITE Present_Value = NULL, PRIORITY = PRIO
5. VERIFICATION Last_Command_Time ~= (the current local time)
6. IF (O1 has Minimum_On_Time or Minimum_Off_Time properties) THEN
WAIT the larger of Minimum_Off_Time and Minimum_On_Time
7. VERIFICATION Current_Command_Priority = NULL

8. VERIFY Value_Source = 'None' -- the value is the choice 'none'

7.3.1.28.4 Commandable Value Source Test

Purpose: To verify that the Value_Source, Value_Source_Array, and Last_Command_Time update correctly when Present_Value is written in a commandable object.

Test Concept: A commandable object which supports the value source mechanism is selected for the test. The Present_Value is written. Last_Command_Time, Value_Source, and Value_Source_Array properties are checked to verify that they have been updated appropriately. Value_Source is then written, and it is verified that Last_Command_Time property has not changed.

Configuration Requirements: The object being tested shall be commandable and support the Value Source mechanism. No other internal processes shall be controlling the object.

Test Steps:

-- Verify that the value source properties are updated when Present_Value is commanded.

1. WRITE Present_Value = (V1: any valid value), PRIORITY = (PRIO: any value other than 6)
2. VERIFY Value_Source_Array = (SRC1: TD's device identifier or network address), ARRAY_INDEX = PRIO
3. VERIFY Value_Source = SRC1
4. VERIFY Last_Command_Time ~= (the current local time)

-- Verify that Value_Source can be written and that Last_Command_Time does not update.

5. READ T1 = (O1), Last_Command_Time
6. WRITE Value_Source = (SRC2: any valid value), PRIORITY = PRIO
7. VERIFY Value_Source_Array = SRC2, ARRAY_INDEX = PRIO
8. IF (Current_Command_Priority == PRIO) THEN
 VERIFY Value_Source = SRC2
9. VERIFY Last_Command_Time = T1

7.3.1.28.5 Life Safety Value_Source Property Test

Purpose: To verify that the Value_Source property indicates the source of the current Mode property in a life safety object.

Test Concept: Select a life safety object which supports the Value Source mechanism. Mode is written, and it is verified that Value_Source is updated appropriately. Value_Source is then written to verify that the last writer can update it.

Test Steps:

1. WRITE Mode = V1
2. VERIFY Mode = V1
3. VERIFY Value_Source = (TD's device identifier or network address)
4. WRITE Value_Source = (any valid value, V2)
5. VERIFY Value_Source = V2

7.3.1.29 Audit_Level Property Tests

7.3.1.29.1 Object Specific Configurable Audit_Level NONE Test

Purpose: Verify that the Audit_Level property, in an auditable object, controls the audit level for the object.

Test Concept: An object, O1, is selected which supports a configurable Audit_Level property. With O1 configured to report notifications, Audit_Level is changed to NONE. An auditable action is performed on O1 and it is verified that no notification is generated.

Audit_Level is then changed to AUDIT_CONFIG, and an auditable config action is performed on the object. It is verified that a notification is sent. An auditable non-config action is performed on the object, and it is verified that no notification is sent.

Audit_Level is then changed to AUDIT_ALL, and an auditable action is performed on the object. It is verified that a notification is sent.

Configuration Requirements: The IUT is configured to generate audit notifications. The selected object, O1, shall have auditable configuration operations that can be applied to it. AR is the Audit Reporter object configured to report for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

1. WRITE O1, Audit_Level = NONE
2. WRITE AR, Audit_Level = (AUDIT_CONFIG or AUDIT_ALL)
3. MAKE(perform an operation on O1 that would be reported by the AR if O1.Audit_Level were AUDIT_ALL)
4. WAIT(AR.Maximum_Send_Delay + **Notification Fail Time**)
5. CHECK(that the IUT did not report an audit notification for the operation)
6. IF (O1 has auditable configuration operations, such as writable configuration properties) THEN {
 - WRITE O1, Audit_Level = AUDIT_CONFIG
 - WRITE AR, Audit_Level = NONE
 - MAKE(perform a config operation on O1)
 - IF the IUT is configured to generate unconfirmed audit notifications THEN {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = (a notification of the operation performed)
 - } ELSE {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE ConfirmedAuditNotification-Request,
 - 'Notifications' = (a notification of the operation performed)
 - TRANSMIT BACnet-SimpleACK-PDU
- MAKE(perform an auditable non-config operation on O1)
- WAIT AR.Maximum_Send_Delay + **Notification Fail Time**
- CHECK(that the IUT did not report an audit notification for the operation)
- }
7. WRITE O1, Audit_Level = AUDIT_ALL
8. WRITE AR, Audit_Level = NONE
9. MAKE(perform an auditable operation on O1)
10. IF the IUT is configured to generate unconfirmed audit notifications THEN {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = (a notification of the operation performed)
- } ELSE {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = (a notification of the operation performed)
 - TRANSMIT BACnet-SimpleACK-PDU
- }

7.3.1.29.2 Audit Reporter Audit_Level Test

Purpose: Verify that the Audit_Reporter's Audit_Level property is used in objects without an Audit_Level, or when an object's Audit_Level is DEFAULT.

Test Concept: An object, O1, is selected which supports a configurable Audit_Level property, or which does not have an Audit_Level property. The Audit_Reporter for O1 is referred to as AR1. If O1 has an Audit_Level property, it is set to DEFAULT.

With O1 configured to report notifications, AR1's Audit_Level is changed to NONE. An auditable action is performed on O1 and it is verified that no notification is generated.

AR1's Audit_Level is then changed to AUDIT_CONFIG, and an auditable config action is performed on O1. It is verified that a notification is sent. An auditable non-config action is performed on O1, and it is verified that no notification is sent.

Audit_Level is then changed to AUDIT_ALL, and an auditable action is performed on O1. It is verified that a notification is sent.

Configuration Requirements: The IUT is configured to generate audit notifications. The selected object, O1, should have auditable configuration operations and auditable non-configuration operations that can be applied to it. AR is the Audit Reporter object configured to report for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

1. IF O1 contains an Audit_Level property THEN
 WRITE O1, Audit_Level = DEFAULT
2. WRITE AR.Audit_Level = NONE
3. MAKE(perform an operation on O1 that would be reported by the AR if O1.Audit_Level were AUDIT_ALL)
4. WAIT AR.Maximum_Send_Delay + **Notification Fail Time**
5. CHECK(that the IUT did not report an audit notification for the operation)
6. WRITE AR.Audit_Level = AUDIT_CONFIG
7. IF O1 supports auditable config operations THEN {
 MAKE(perform a config operation on O1)
 IF the IUT is configured to generate unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = (a notification of the operation performed)
 } ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = (a notification of the operation performed)
 TRANSMIT BACnet-SimpleACK-PDU
 }
}
8. IF O1 supports auditable config operations THEN {
 MAKE(perform a non-config operation on O1)
 WAIT AR.Maximum_Send_Delay + **Notification Fail Time**
 CHECK(that the IUT did not report an audit notification for the operation)
}
9. WRITE AR.Audit_Level = AUDIT_ALL
10. IF O1 supports auditable config operations THEN {
 MAKE(perform a config operation on O1)
 IF the IUT is configured to generate unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = (a notification of the operation performed)
 } ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = (a notification of the operation performed)
 TRANSMIT BACnet-SimpleACK-PDU
 }
}
11. IF O1 supports auditable config operations THEN {
 MAKE(perform a non-config operation on O1)
 IF the IUT is configured to generate unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

```

    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = (a notification of the operation performed)
  } ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = (a notification of the operation performed)
    TRANSMIT BACnet-SimpleACK-PDU
  }
}

```

7.3.1.29.3 Audit_Level Change Notification Test

Purpose: Verify that changes to Audit_Level property results in audit notifications.

Test Concept: An object, O1, is selected which supports a writable and/or configurable Audit_Level property. The Audit_Level property is written to each valid audit level, and a notification of the change is checked for. The Audit_Level property is then configured (not via BACnet writes) to each valid audit level, and a notification of the change is checked for.

Configuration Requirements: The IUT is configured to generate audit notifications, and O1's Audit_Level property is set to NONE.

Test Steps:

- IF Audit_Level is writable THEN {
 REPEAT AL = (AUDIT_CONFIG, AUDIT_ALL, DEFAULT, NONE) DO {
 IF O1 is an Audit Reporter and AL is DEFAULT THEN {
 -- don't test DEFAULT on Audit Report object
 } ELSE {
 WRITE O1, Audit = AL
 IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = (a notification indicating change of
 Audit_Level)
 } ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = (a notification indicating change of
 Audit_Level)
 TRANSMIT BACnet-SimpleACK-PDU
 }
 }
 }
 }
- IF Audit_Level is changeable without writing it via BACnet THEN {
 REPEAT AL = (AUDIT_CONFIG, AUDIT_ALL, DEFAULT, NONE) DO {
 IF O1 is an Audit Reporter and AL is DEFAULT THEN {
 -- don't test DEFAULT on Audit Report object
 } ELSE {
 MAKE(O1, Audit = AL without using BACnet services to write the property)
 IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = (a notification indicating change of
 Audit_Level)
 } ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = (a notification indicating change of

```

        Audit_Level)
    TRANSMIT BACnet-SimpleACK-PDU
  }
}
}

```

7.3.1.30 Audit_Notification_Recipient Property Tests

7.3.1.30.1 Audit_Notification_Recipient Test

Purpose: Verify that an Audit_Notification_Recipient accepts and correctly uses all forms of recipient addresses.

Test Concept: With an object configured to report notifications, the Device object's Audit_Notification_Recipient property is set to a local broadcast. An action is performed on the IUT which should result in an audit notification, and it is verified that the notification is locally broadcast. This is repeated for global broadcasts, and unicast-recipients. Only the unicast form is tested for devices which do not generate UnconfirmedAuditNotifications.

Configuration Requirements: The IUT is configured to report audit notifications. If the IUT supports sending unconfirmed audit notifications, it shall be configured to do so. Audit_Notification_Recipient is configured to be the TD, and audit reporting is enabled in the IUT.

Notes to Tester: Where the IUT is expected to send multiple audit notifications for an operation, the notifications can be generated in any order.

Test Steps:

-- Local Broadcast Recipient

1. IF the IUT supports sending unconfirmed audit notifications THEN {
 - WRITE Audit_Notification_Recipient = (local broadcast recipient)
 - BEFORE Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = (a notification for the change to Audit_Notification_Recipient)
 - |
 - (UnconfirmedAuditNotification-Request
 - DESTINATION = GLOBAL BROADCAST,
 - 'Notifications' = (a notification for the change to Audit_Notification_Recipient))
 - IF the first notification was not globally broadcast) THEN {
 - RECEIVE UnconfirmedAuditNotification-Request,
 - DESTINATION = LOCAL BROADCAST,
 - 'Notifications' = (a notification for the change to Audit_Notification_Recipient)}
 - MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the Audit_Notification_Recipient property)
 - BEFORE Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - DESTINATION = LOCAL BROADCAST,
 - 'Notifications' = (a notification correctly indicating the operation performed)

-- Global Broadcast Recipient

2. IF the IUT supports sending unconfirmed audit notifications THEN {
 - WRITE Audit_Notification_Recipient = (global broadcast recipient)
 - BEFORE Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - DESTINATION = LOCAL BROADCAST,
 - 'Notifications' = (a notification for the change to Audit_Notification_Recipient)
 - |
 - (UnconfirmedAuditNotification-Request
 - DESTINATION = GLOBAL BROADCAST,
 - 'Notifications' = (a notification for the change to Audit_Notification_Recipient))
 - IF (the first notification was not globally broadcast) THEN {

```

    RECEIVE UnconfirmedAuditNotification-Request,
    DESTINATION = GLOBAL BROADCAST,
    'Notifications' = (a notification for the change to Audit_Notification_Recipient)
  }
  MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the
    Audit_Notification_Recipient property)
  BEFORE Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    DESTINATION = GLOBAL BROADCAST,
    'Notifications' = (a notification correctly indicating the operation performed)
  }

-- Unicast Recipient
3.  WRITE Audit_Notification_Recipient = (D1: a device other than the TD)
4.  IF the IUT is configured to send unconfirmed notifications THEN {
    BEFORE (Maximum_Send_Delay plus maximum time to resolve TD)
      RECEIVE UnconfirmedAuditNotification-Request,
      DESTINATION = TD,
      'Notifications' =(a notification for the change to Audit_Notification_Recipient)
    | (UnconfirmedAuditNotification-Request
      DESTINATION = GLOBAL BROADCAST,
      'Notifications' = (a notification for the change to Audit_Notification_Recipient))
  } ELSE {
    BEFORE (Maximum_Send_Delay plus maximum time to resolve TD)
      RECEIVE ConfirmedAuditNotification-Request,
      DESTINATION = D1,
      'Notifications' = (a notification for the change to Audit_Notification_Recipient)
    TRANSMIT BACnet-SimpleACK-PDU,
    SOURCE = D1
  }
5.  IF (the first notification was not globally broadcast) THEN
    RECEIVE UnconfirmedAuditNotification-Request,
    DESTINATION = TD,
    'Notifications' = (a notification for the change to Audit_Notification_Recipient)

6.  MAKE(perform an operation on the IUT which will result in an audit notification, other than changing the
    Audit_Notification_Recipient property)
7.  BEFORE Maximum_Send_Delay
    RECEIVE UnconfirmedAuditNotification-Request,
    DESTINATION = TD,
    'Notifications' = (a notification correctly indicating the operation performed)

```

7.3.1.31 Audit_Priority_Filter Property Tests

7.3.1.31.1 Audit_Priority_Filter Target Audit Reporting Test

Purpose: Verify that Audit_Priority_Filter correctly filters the generation of audit notifications based on priority.

Test Concept: An auditable commandable object for which Audit_Priority_Filter filtering can be applied is configured to report all write operations. The Audit_Priority_Filter is configured to restrict audit notifications to all but a single priority X. The object is commanded at a priority other than X and it is verified that an audit notification is sent. The object is then commanded at priority X and it is verified that no audit notification is sent.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. If the IUT does not support the Priority_Array property in any object for which audit reporting can be configured, or if the IUT does not support a configurable Audit_Priority_Filter property, this test shall be skipped. AR shall be the Audit Reporter object for O1.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

-- Test the object's Audit_Priority_Filter

1. IF O1 supports a configurable Audit_Priority_Filter property THEN {
 - WRITE O1, Audit_Priority_Filter = { all ones except priority X (bit X-1) }
 - TRANSMIT WriteProperty-Request,
 - 'Invoke Id' = I,
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Present_Value,
 - 'Property Value' = (V: any valid value),
 - 'Priority' = (PRIO: any valid priority, but not X)
 - BEFORE **Internal Processing Fail Time**
 - RECEIVE BACnet-SimpleACK-PDU
 - (BACnet-Error-PDU,
 - 'Error Type' = (E: any error))
 - IF the IUT is configured to send unconfirmed audit notifications THEN {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = ({ -- there may be a second notification included for the write to Audit_Priority_Filter. If there is, the order of the notifications in the list is not relevant
 - source-timestamp absent
 - target-timestamp = (IUT's local time),
 - source-device = TD,
 - source-object absent
 - operation = WRITE,
 - source-comment absent
 - target-comment = (any valid value, or absent),
 - invoke-id = (the invoke ID of the write to Present_Value),
 - source-user-id = (the value from the operation if provided, otherwise absent),
 - source-user-role = (the value from the operation if provided, otherwise absent),
 - target-device = IUT,
 - target-object = O1,
 - target-property = Present_Value,
 - target-priority = PRIO,
 - target-value = V,
 - current-value = (the value before the write. may be absent if the value size is larger than 32 encoded octets),
 - result = (E, if the op failed, otherwise absent)
 - })
 - } ELSE {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE ConfirmedAuditNotification-Request,
 - 'Notifications' = ({ -- there may be a second notification included for the write to Audit_Priority_Filter. If there is, the order of the notifications in the list is not relevant
 - source-timestamp absent
 - target-timestamp = (IUT's local time),
 - source-device = TD,
 - source-object absent
 - operation = WRITE,
 - source-comment absent
 - target-comment = (any valid value, or absent),
 - invoke-id = (the invoke ID of the write to Present_Value),
 - source-user-id = (the value from the operation if provided, otherwise absent),
 - source-user-role = (the value from the operation if provided, otherwise absent),
 - target-device = IUT,
 - target-object = O1,
 - target-property = Present_Value,
 - target-priority = PRIO,
 - target-value = V,

current-value = (the value before the write. may be absent if the value
size is larger than 32 encoded octets),
result = (E, if the op failed, otherwise absent)

})

TRANSMIT BACnet-SimpleACK-PDU

}

TRANSMIT WriteProperty-Request,

'Invoke Id' = I,

'Object Identifier' = O1,

'Property Identifier' = Present_Value,

'Property Value' = (V: any valid value),

'Priority' = (PRIO: X)

BEFORE Internal Processing Fail Time

RECEIVE BACnet-SimpleACK-PDU

|

(BACnet-Error-PDU,

'Error Type' = (E: any error))

WAIT AR.Maximum_Send_Delay + Notification Fail Time

CHECK(no audit notification sent)

}

-- Test the Audit Reporter object's Audit_Priority_Filter

2. IF AR supports a configurable Audit_Priority_Filter property and O1 has no Audit_Priority_Filter or it can be configured to None THEN {

IF O1 has an Audit_Priority_Filter THEN

WRITE O1, Audit_Priority_Filter = NONE

WRITE AR, Audit_Priority_Filter = { all ones except priority X (bit X-1) }

TRANSMIT WriteProperty-Request,

'Invoke Id' = I,

'Object Identifier' = O1,

'Property Identifier' = Present_Value,

'Property Value' = (V: any valid value),

'Priority' = (PRIO: any valid priority, but not X)

BEFORE Internal Processing Fail Time

RECEIVE BACnet-SimpleACK-PDU

|

(BACnet-Error-PDU,

'Error Type' = (E: any error))

IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + Notification Fail Time

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = ({ -- there may be a second notification included for the write
to Audit_Priority_Filter. If there is, the order of the notifications in the list is not relevant

-- source-timestamp absent

target-timestamp = (IUT's local time),

source-device = TD,

-- source-object absent

operation = WRITE,

-- source-comment absent

target-comment = (any valid value, or absent),

invoke-id = (the invoke ID of the write to Present_Value),

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = IUT,

target-object = O1,

target-property = Present_Value,

target-priority = PRIO,

target-value = V,

current-value = (the value before the write. may be absent if the value
size is larger than 32 encoded octets),

result = (E, if the op failed, otherwise absent)


```

    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( { -- there may be a second notification included for the write
to Audit_Priority_Filter. If there is, the order of the notifications in the list is not relevant
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = WRITE,
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = (the invoke ID of the write to Present_Value),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise absent),
        target-device = IUT,
        target-object = O1,
        target-property = Present_Value,
        target-priority = PRIO,
        target-value = V,
        current-value = (the value before the write. May be absent if the value
size is larger than 32 encoded octets),
        result = (E, if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}
TRANSMIT WriteProperty-Request,
'Invoke Id' = I,
'Object Identifier' = O1,
'Property Identifier' = Present_Value,
'Property Value' = (V: any valid value),
'Priority' = (PRIO: X)
BEFORE Internal Processing Fail Time
RECEIVE BACnet-SimpleACK-PDU
|
(BACnet-Error-PDU
'Error Type' = (E: any error))
WAIT AR.Maximum_Send_Delay + Notification Fail Time
CHECK(no audit notification sent)
}

```

7.3.1.32 Auditable_Operations Property Tests

7.3.1.32.1 Non-configurable Auditable_Operations Property Test

Purpose: Verify that non-configurable Auditable_Operations properties are set to the required value.

Test Concept: Select an object, O1, which has a non-configurable Auditable_Operations property. Verify that the property is set to the required value.

Test Steps:

1. READ AO = Audit_Operations
2. CHECK(Audit_Operations = (FALSE,TRUE,TRUE,TRUE,?,TRUE,?,?,?,?,FALSE,FALSE,?,?,?,...))
 - flags READ, NOTIFICATION and SUBSCRIPTION are FALSE,
 - flags WRITE, CREATE, DELETE, ACKNOWLEDGE-ALARM are TRUE, and
 - all other flags can be any value

7.3.1.32.2 Auditable_Operations Target Audit Reporting Test

Purpose: Verify that Auditable_Operations controls which operations are auditable.

Test Concept: The IUT is configured to report some operations and not others through the Auditable_Operations property. Each of the standard auditable operations are performed against the IUT, and the filtering provided by Auditable_Operations is verified to be correct.

Configuration Requirements: The IUT is configured to report all audit notifications with the exception that at least some auditable operations are disabled via Auditable_Operations.

Test Steps:

1. REPEAT OP = (each of the standard auditable operations for which the IUT is configured to report except any operation which is unreasonably difficult to perform, such as AUDITING_FAILURE) DO {
MAKE(perform OP on the IUT)
IF the IUT is configured to send unconfirmed audit notifications THEN {
BEFORE (Audit Reporter for OP).Maximum_Send_Delay + **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = OP,
-- source-comment absent
target-comment = (any valid value, or absent unless OP is GENERAL),
invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property. shall be 16 or absent if no priority supplied and the target is a property),
target-value = (the target value or absent if no target value for the operation. May be absent if the value size is larger than 32 encoded octets),
current-value = (the value before the op or absent if no target value. may be absent if the value size is larger than 32 encoded octets),
result = (the reason for failure if OP failed, otherwise absent)
})
} ELSE {
BEFORE (Audit Reporter for OP).Maximum_Send_Delay + **Notification Fail Time**
RECEIVE ConfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = OP,
-- source-comment absent
target-comment = (any valid value, or absent unless OP is GENERAL),
invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
source-user-id = (the value from the operation if provided, otherwise

```

        absent),
        source-user-role = (the value from the operation if provided, otherwise
        absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
        property),
        target-priority = (the priority supplied, or absent if the target is not a
        property. shall be 16 or absent if no priority supplied and the
        target is a property),
        target-value = (the target value or absent if no target value for the
        operation. may be absent if the value size is larger than 32
        encoded octets),
        current-value = (the value before the op or absent if no target value.
        may be absent if the value size is larger than 32 encoded octets),
        result = (the reason for failure if OP failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}
}
2. REPEAT OP = (each of the standard auditable operations for which the IUT is configured to NOT report
except any operation which is unreasonably difficult to perform, such as AUDITING_FAILURE) DO {
    MAKE(perform OP on the the IUT)
    WAIT (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    CHECK(no audit notification was received)
}

```

7.3.1.32.3 Auditable_Operations Source Audit Reporting Test

Purpose: Verify that Auditable_Operations controls which operations performed by the IUT are auditable.

Test Concept: The IUT is configured to report some source operations and not others through the Auditable_Operations property. Each of the standard auditable operations which the IUT can perform are performed by the IUT, and the filtering provided by Auditable_Operations is verified to be correct.

Configuration Requirements: The IUT is configured to report all source audit notifications with the exception that at least some auditable operations are disabled via Auditable_Operations.

Test Steps:

1. REPEAT OP = (each of the standard auditable operations the IUT is able to perform on another device and is configured to report except any operation which is unreasonably difficult to perform) DO {
 MAKE(the IUT perform OP on the TD)
 IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE (Audit Reporter for OP).Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({
 source-timestamp = (IUT's local time),
 -- target-timestamp absent
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not
 initiated by an object),
 operation = OP,
 source-comment = (any valid value, or absent unless OP is
 GENERAL),
 -- target-comment absent
 invoke-id = (the invoke id from the operation, or absent if it was
 unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise
 absent),
 source-user-role = (the value from the operation if provided, otherwise

```

        absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE (Audit Reporter for OP).Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not
            initiated by an object),
        operation = OP,
        source-comment = (any valid value, or absent unless OP is
            GENERAL),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}
}

```

2. REPEAT OP = (each of the standard auditable operations the IUT is able to perform on another device and is configured to NOT report except any operation which is unreasonably difficult to perform) DO {

MAKE(the IUT perform OP on the TD)

WAIT AR.Maximum_Send_Delay + **Notification Fail Time**

CHECK(no audit notifications were sent)

}

7.3.1.33 Maximum_Send_Delay Property Tests

7.3.1.33.1 Maximum_Send_Delay Test

Purpose: Verify that Audit Reporter objects abide the Maximum_Send_Delay value.

Test Concept: An Audit Reporter object is selected which contains a Maximum_Send_Delay property. A sequence of N auditable operations is performed on the IUT, and if the IUT is a client, the IUT is made to perform a sequence of M auditable operations. The IUT is then monitored to ensure that the audit notifications are sent within the selected Maximum_Send_Delay.

N, the number of operations performed on the IUT shall be 2 or greater. M, the number of operations that the IUT will perform shall be 0 if the IUT does not perform auditable operations, and 2 or greater otherwise.

The value that Maximum_Send_Delay is configured for shall be large enough to allow for all of the operations to be performed.

Test Steps:

1. WRITE AR1, Maximum_Send_Delay = (MSD: a value large enough to accomplish and report N+M auditable operations)
2. WRITE AR2, Maximum_Send_Delay = MSD
3. MAKE(perform N auditable operations on the IUT which would be reported through AR1)
4. MAKE(the IUT perform M auditable operations which would be reported through AR2)
5. WAIT Maximum_Send_Delay + (Notification Fail Time * M+N)
6. CHECK(that all expected operations are reported)

7.3.1.34 Monitored_Objects property Tests

7.3.1.34.1 Monitored_Objects Test

Purpose: Verify that the correct Audit Reporter is used for an auditable object.

Test Concept: Each Audit Reporter, which contains a Monitored_Objects property, in the device is tested individually. The selected Audit Reporter is enabled, and all others are disabled. An object that the enabled Audit Reporter reports for is selected. An auditable operation is performed on the object, and it is verified that an audit notification is generated. An object that the enabled Audit Reporter does not report for is selected. An auditable operation is performed on the object, and it is verified that no audit notification is generated.

Configuration Requirements: The IUT is configured to send unconfirmed audit notifications. If the Monitored_Objects property is not supported by any Audit Reporter objects in the IUT, this test shall be skipped. If the IUT only supports a single Audit Reporter object for target object reporting, and its Monitored_Objects property is always set to all objects, this test shall be skipped. Configure all Audit Reporters to report all operations and set Audit_Level to NONE for all Audit Reporter objects.

Notes to Tester: In this test, the audit reporting configuration properties are assumed to be writable. If one is only configurable, then the WRITE operation should be replaced with MAKE.

Test Steps:

1. IF the Monitored_Objects property is writable in one or more of the AR objects THEN
MAKE(reconfigure which AR is used by which objects)
2. REPEAT AR = (each Audit Reporter object) DO {
O1 = (an object O1 which reports thru AR as determined by Monitored_Objects and AR precedence)
O2 = (an object O2 which reports thru a different Audit Reporter, AR2, as determined by
Monitored_Objects and AR precedence)
WRITE AR.Audit_Level = AUDIT_ALL
MAKE(perform an auditable operation on O1)
IF the IUT is configured to send unconfirmed auto notifications THEN {
BEFORE AR.Maximum_Send_Delay + Notification Fail Time
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = (a notification indicating the operation)

```

    } ELSE {
        BEFORE AR.Maximum_Send_Delay + Notification Fail Time
        RECEIVE ConfirmedAuditNotification-Request,
        'Notifications' = (a notification indicating the operation)
        TRANSMIT BACnet-SimpleACK-PDU
    }
    MAKE(perform an auditable operation on O2)
    WAIT AR2.Maximum_Send_Delay + Notification Fail Time
    CHECK(that the IUT did not report an audit notification for the operation)
    WRITE AR.Audit_Level = NONE
}

```

7.3.1.35 Send_Now Property Tests

7.3.1.35.1 Send_Now Test

Purpose: Verify that writing True to Send_Now results in the sending of delayed audit notifications.

Test Concept: The IUT is configured to delay sending audit notifications. An Audit Reporter object, AR, is selected which contains a Send_Now property. An auditable operation is performed on the IUT. Send_Now is then written and it is verified that the audit notifications are sent.

Configuration Requirements: If the IUT cannot be made to delay sending notifications without heroic efforts, this test shall be skipped.

Test Steps:

1. WRITE AR, Maximum_Send_Delay = (a value large enough to accomplish the test)
2. MAKE(perform whatever actions are required to make the IUT delay sending notifications)
3. WRITE AR, Send_Now = TRUE
4. IF the IUT is configured to send unconfirmed notifications THEN {
 - BEFORE **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = (one or more notifications for operation applied to the IUT)
- ELSE {
 - BEFORE **Notification Fail Time**
 - RECEIVE ConfirmedAuditNotification-Request,
 - 'Notifications' = (one or more notifications for operation applied to the IUT)
 - TRANSMIT BACnet-SimpleACK-PDU

7.3.2 Object Specific Tests

The tests in this clause apply only to the specified object type. Only a single instance of each supported object type must be tested.

7.3.2.1 Analog Input Object Tests

7.3.2.1.1 Input Tracking Test

Purpose: To verify the ability to track and represent the value of an analog input.

Configuration Requirements: The IUT shall be connected to an analog input that can be externally controlled during the test. Any scaling information that may be needed to verify that the value is reasonable shall also be provided. The Analog Input object associated with this physical input shall be configured with Out_Of_Service = FALSE.

Test Steps:

1. MAKE (the real analog input take on a known value near the middle of the supported range)
2. VERIFY Present_Value = (a value that corresponds to the known input signal)
3. MAKE (the real analog input take on a higher known value within the supported range)
4. VERIFY Present_Value = (a value that corresponds to the known input signal)
5. MAKE (the real analog input take on a value lower than the one used in step 1)

6. VERIFY Present_Value = (a value that corresponds to the known input signal)

7.3.2.1.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.1.3 Change of Value Tests

Tests to verify change of value reporting capabilities for Analog Input objects are covered in 8.2.1, 8.2.2, 8.3.1 and 8.3.2.

7.3.2.1.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Analog Input objects are covered in 8.4.6.

7.3.2.2 Analog Output Object Tests

7.3.2.2.1 Output Tracking Test

Purpose: To verify the ability to represent and implement a physical analog output.

Configuration Requirements: The IUT shall be configured with an analog output that can be observed with a multimeter during the test. Any scaling information that may be needed to verify that the value is reasonable shall also be provided. The Analog Output object associated with this physical output shall be configured with Out_Of_Service = FALSE.

Test Steps:

1. WRITE Present_Value = (a value near the middle of the supported range),
PRIORITY = (a priority higher than any internal algorithms writing to this property)
2. VERIFY Present_Value = (the value written in step 1)
3. CHECK (with a multimeter to verify that the physical output value corresponds to the value of the Present_Value property)
4. WRITE Present_Value = (a value near the high limit of the supported range),
PRIORITY = (a priority higher than any internal algorithms writing to this property)
5. VERIFY Present_Value = (the value written in step 4)
6. CHECK (with a multimeter to verify that the physical output value corresponds to the value of the Present_Value property)
7. WRITE Present_Value = (a value near the low limit of the supported range),
PRIORITY = (a priority higher than any internal algorithms writing to this property)
8. VERIFY Present_Value = (the value written in step 7)
9. CHECK (with a multimeter to verify that the physical output value corresponds to the value of the Present_Value property)

7.3.2.2.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.2.3 Prioritized Commands Tests

Tests to verify prioritized commands of Analog Output objects are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.2.4 Change of Value Tests

Tests to verify change of value reporting capabilities for Analog Output objects are covered in 8.2.1, 8.2.2, 8.3.1 and 8.3.2.

7.3.2.2.5 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Analog Output objects are covered in 8.4.6.

7.3.2.3 Analog Value Object Tests

7.3.2.3.1 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.3.2 Prioritized Commands Tests

Tests to verify prioritized commands of Analog Value objects are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.3.3 Change of Value Tests

Tests to verify change of value reporting capabilities for Analog Value objects are covered in 8.2.1, 8.2.2, 8.3.1 and 8.3.2.

7.3.2.3.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Analog Value objects are covered in 8.4.6.

7.3.2.4 Averaging Object Tests

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

7.3.2.4.1 Reinitializing the Samples

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

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

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

Test Steps:

1. WRITE Attempted_Samples = 0,
2. WAIT (at least two sample times),
3. VERIFY Minimum_Value = (a value x : $-\text{INF} < x < \text{INF}$),
4. VERIFY Average_Value = (a value $\neq \text{NaN}$),
5. VERIFY Maximum_Value = (a value x : $\text{Minimum\_Value} \leq x < \text{INF}$),
6. VERIFY Attempted_Samples = (a value $x \geq 2$),
7. VERIFY Valid_Samples = (a value $x \geq 2$),
8. WRITE Window_Interval = (any new value that will result in an appropriate sample time),
9. WAIT (at least two sample times),
10. VERIFY Minimum_Value = (a value x : $-\text{INF} < x < \text{INF}$),
11. VERIFY Average_Value = (a value $\neq \text{NaN}$),
12. VERIFY Maximum_Value = (a value x : $\text{Minimum\_Value} \leq x < \text{INF}$),
13. VERIFY Attempted_Samples = (a value $x \geq 2$),
14. VERIFY Valid_Samples = (a value $x \geq 2$),
15. WRITE Window_Samples = (any new value that will result in an appropriate sample time),
16. IF (Object_Property_Reference is writable) THEN {
 - WAIT (at least two sample times),
 - VERIFY Minimum_Value = (a value x : $-\text{INF} < x < \text{INF}$),
 - VERIFY Average_Value = (a value $\neq \text{NaN}$),
 - VERIFY Maximum_Value = (a value x : $\text{Minimum\_Value} \leq x < \text{INF}$),
 - VERIFY Attempted_Samples = (a value $x \geq 2$),
 - VERIFY Valid_Samples = (a value $x \geq 2$),
 - WRITE Object_Property_Reference = (any new value),
 - IF (Samples_are_taken_immediately) THEN
 - VERIFY Attempted_Samples = 1,
 - VERIFY Minimum_Value = Average_Value,,
 - VERIFY Maximum_Value = Average_Value,
 - VERIFY Valid_Samples = 1
 - ELSE
 - VERIFY Attempted_Samples = 0,
 - VERIFY Minimum_Value = INF ,
 - VERIFY Maximum_Value = $-\text{INF}$,
 - VERIFY Average_Value = NaN ,

```

    VERIFY Valid_Samples = 0
  }

```

7.3.2.4.2 Managing the Sample Window

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

Test Concept: An Averaging object is configured to monitor a property that can be controlled manually by the testing agent or by the TD. The TD initializes the sample and then monitors the Minimum_Value, Average_Value, Maximum_Value, Attempted_Samples, and Valid_Samples properties after each sampling interval to verify that their values are properly tracking the monitored value. This requires the ability to manipulate the values of the monitored property value and a slow enough sampling interval to permit the analysis. This continues until after the sample window is full.

Configuration Requirements: The IUT shall be configured with an Averaging object used to monitor a property that can be controlled by the testing agent or by the TD. The sampling interval shall be configured to allow time to change the monitored property value and to determine if each of the properties Minimum_Value, Average_Value, Maximum_Value, Attempted_Samples, and Valid_Samples correctly changes after each sample interval.

Test Steps:

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

7.3.2.5 Binary Input Object Tests

7.3.2.5.1 Input Tracking Test

Purpose: To verify the ability to track and represent the value of a binary input.

Configuration Requirements: The IUT shall be connected to a binary input that can be externally controlled during the test.

Test Steps:

1. MAKE (the real binary input ACTIVE)
2. VERIFY Present_Value = ACTIVE
3. MAKE (the real binary input INACTIVE)
4. VERIFY Present_Value = INACTIVE

7.3.2.5.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.5.3 Polarity Property Tests

Purpose: To verify that the Polarity property interacts properly with the associated physical input. If the Polarity property is not writable this test shall be omitted.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Binary Input object being tested),
 'Property Identifier' = Polarity
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Binary Input object being tested),
 'Property Identifier' = Polarity
 'Property Value' = NORMAL | REVERSE
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Binary Input object being tested),
 'Property Identifier' = Present_Value
4. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Binary Input object being tested),
 'Property Identifier' = Present_Value
 'Property Value' = ACTIVE | INACTIVE
5. IF (the Polarity value in step 2 was NORMAL) THEN
 WRITE Polarity = REVERSE
 VERIFY Polarity = REVERSE
 ELSE
 WRITE Polarity = NORMAL
 VERIFY Polarity = NORMAL
6. IF (the Present_Value in step 4 was ACTIVE) THEN
 VERIFY Present_Value = INACTIVE
 ELSE
 VERIFY Present_Value = ACTIVE

7.3.2.5.4 Change of State Properties Tests

Test to verify the ability to monitor changes of state for Binary Input objects are covered in 7.3.1.8.

7.3.2.5.5 Active Time Properties Tests

Tests to verify the ability to monitor active time for Binary Input objects are covered in 7.3.1.9.

7.3.2.5.6 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Binary Input objects are covered in 8.4.2.

7.3.2.6 Binary Output Object Tests

7.3.2.6.1 Output Tracking Test

Purpose: To verify the ability to represent and implement a physical binary output.

Configuration Requirements: The IUT shall be configured with a binary output that can be observed during the test.

Test Steps:

1. WRITE Present_Value = ACTIVE
2. VERIFY Present_Value = ACTIVE
3. CHECK (verify that the output is active)
4. WRITE Present_Value = INACTIVE
5. VERIFY Present_Value = INACTIVE
6. CHECK (verify that the output is inactive)

7.3.2.6.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.6.3 Polarity Property Tests

Purpose: To verify that the Polarity property interacts properly with the associated physical output. If the Polarity property is not writable this test shall be omitted.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Binary Output object being tested),
 'Property Identifier' = Polarity
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Binary Output object being tested),
 'Property Identifier' = Polarity,
 'Property Value' = NORMAL | REVERSE
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Binary Output object being tested),
 'Property Identifier' = Present_Value
4. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Binary Output object being tested),
 'Property Identifier' = Present_Value,
 'Property Value' = ACTIVE | INACTIVE
5. CHECK (the status of the physical output)
6. IF (the Polarity value in step 2 was NORMAL) THEN
 WRITE Polarity = REVERSE
 VERIFY Polarity = REVERSE
 ELSE
 WRITE Polarity = NORMAL
 VERIFY Polarity = NORMAL
7. IF (the Present_Value in step 4 was ACTIVE) THEN
 VERIFY Present_Value = ACTIVE
 ELSE
 VERIFY Present_Value = INACTIVE
8. CHECK (the status of the physical output and verify that it is the complement of the status found in step 5)

7.3.2.6.4 Change of State Tests

Test to verify the ability to monitor changes of state for Binary Output objects are covered in 7.3.1.8.

7.3.2.6.5 Elapsed_Active_Time Properties Tests

Tests to verify the ability to monitor active time for Binary Output objects are covered in 7.3.1.9.

7.3.2.6.6 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Binary Output objects are covered in 8.4.4.

7.3.2.6.7 Minimum On and Minimum Off Time Tests

Tests to verify the operation of minimum on and minimum off time algorithms are covered in 7.3.1.4 and 7.3.1.5.

7.3.2.6.8 Prioritized Commands Tests

Tests to verify the operation of the command prioritization algorithm are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.7 Binary Value Object Tests

7.3.2.7.1 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.7.2 Change of State Tests

Test to verify the ability to monitor changes of state for Binary Value objects are covered in 7.3.1.8.

7.3.2.7.3 Elapsed_Active_Time Properties Tests

Tests to verify the ability to monitor active time for Binary Value objects are covered in 7.3.1.9.

7.3.2.7.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Binary Value objects are covered in 8.4.2.

7.3.2.7.5 Minimum On and Minimum Off Time Tests

Tests to verify the operation of minimum on and minimum off time algorithms are covered in 7.3.1.4 and 7.3.1.5.

7.3.2.7.6 Prioritized Commands Tests

Tests to verify the operation of the command prioritization algorithm are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.8 Calendar Object Tests

These tests verify that the Present_Value property of the Calendar object bears the relationship to Date_List specified by BACnet Clause 12.9.6.

7.3.2.8.1 Single Date Rollover Test

Purpose: To verify the ability to represent the Calendar status when the Date_List is in the form of an individual date. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date_List containing a single date. The IUT's clock is set to the date that immediately precedes the one specified in Date_List and a time near the end of the day. The test verifies that the Present_Value of the Calendar object is initially FALSE and that as the time rolls over to the next day the Present_Value changes to TRUE.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a Date. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a Date, then this test shall be skipped.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
 'Time' = (the day preceding the one specified in Date_List,
 24:00:00 – **Schedule Evaluation Fail Time** – 1 minute)) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (the day preceding the one specified in Date_List,
 24:00:00 – **Schedule Evaluation Fail Time** – 1 minute converted to UTC)) |
 MAKE (the local time = 24:00:00 – **Schedule Evaluation Fail Time** – 1 minute)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present_Value = FALSE
4. WAIT (**Schedule Evaluation Fail Time** + 2 minutes)
5. VERIFY Present_Value = TRUE

7.3.2.8.2 Date Range Test

Purpose: To verify the ability to represent the Calendar status when the Date_List is in the form of a BACnetDateRange. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date_List containing a single BACnetDateRange. The IUT's clock is set to a time and date that is outside of the date range. The Present_Value is read and verified to be FALSE. The clock is reset to a value within the date range and the Present_Value is read again to verify that it has the value TRUE. If the IUT can be configured with wildcard fields in the date range then it shall be tested with and without wildcards.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a BACnetDateRange. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetDateRange, then this test shall be skipped

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
 'Time' = (any day and time outside of the specified date range selected by the tester)) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (any day and time outside of the specified date range selected by the tester)) |
 MAKE (the local time = any day and time outside of the specified date range selected by the tester)
2. WAIT **Schedule Evaluation Fail Time**

3. VERIFY Present_Value = FALSE
4. (TRANSMIT TimeSynchronization-Request,
 'Time' = (any day and time inside the specified date range selected by the tester)) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (any day and time inside the specified date range selected by the tester)) |
 MAKE (the local time = any day and time inside the specified date range selected by the tester)
5. WAIT **Schedule Evaluation Fail Time**
6. VERIFY Present_Value = TRUE

7.3.2.8.3 WeekNDay Test

Purpose: To verify the ability to represent the Calendar status when the Date_List is in the form of a BACnetWeekNDay. Either execution of the TimeSynchronization or the UTCTimeSynchronization service must be supported or another means must be supplied to reset the IUT's clock during the test.

Test Concept: The Calendar object is configured with a Date_List containing a single BACnetWeekNDay. The IUT's clock is set to a time and date, T1, that matches the BACnetWeekNDay mask. The Present_Value is read and verified to be TRUE. The clock is reset to a value, T2, that matches the BACnetWeekNDay mask except for the month. The Present_Value is read and verified to be FALSE. The clock is reset again to a value, T3, that matches the BACnetWeekNDay mask except for the week of the month. The Present_Value is read and verified to be FALSE. The clock is reset again to a value, T4, that matches the BACnetWeekNDay mask except for the day of the week. The Present_Value is read and verified to be FALSE. In between each change, the clock is reset to T1 to force the Present_Value back to TRUE.

Configuration Requirements: The IUT shall be configured with a Calendar object that contains a Date_List with a single BACnetCalendarEntry in the form of a BACnetWeekNDay. The BACnetWeekNDay shall be the 11th month, last seven days, and Saturday. If Date_List property cannot be configured with a BACnetCalendarEntry in the form of a BACnetWeekNDay, then this test shall be skipped.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T1) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T1))
 MAKE (the local time = T1)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present_Value = TRUE
4. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T2) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T2)) |
 MAKE (the local time = T2)
5. WAIT **Schedule Evaluation Fail Time**
6. VERIFY Present_Value = FALSE
7. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T1) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T1)) |
 MAKE (the local time = T1)
8. WAIT **Schedule Evaluation Fail Time**
9. VERIFY Present_Value = TRUE
10. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T3) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T3)) |
 MAKE (the local time = T3)
11. WAIT **Schedule Evaluation Fail Time**
12. VERIFY Present_Value = FALSE
13. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T1) |

- (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T1)) |
 MAKE (the local time = T1)
14. WAIT **Schedule Evaluation Fail Time**
15. VERIFY Present_Value = TRUE
16. (TRANSMIT TimeSynchronization-Request,
 'Time' = (T4)) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (T4)) |
 MAKE (the local time = T4)
17. WAIT **Schedule Evaluation Fail Time**
18. VERIFY Present_Value = FALSE

7.3.2.9 Command Object Tests

7.3.2.9.1 All Writes Successful with Post Delay Test

Purpose: To verify that a Command object can successfully execute an action list that includes post delays.

Test Concept: The IUT is configured with an action list that includes manipulating a sequence of externally visible outputs with a time delay between each output. The TD triggers this action list and the tester observes the external changes. If the IUT does not support Post Delay, then this test shall be omitted. If the IUT does not support action list configuration for this Test Concept, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object having an action list, X, that includes writing to a sequence of externally visible outputs. There shall be a post delay between writes to the externally visible outputs that is long enough for the tester to observe the delay.

Test Steps:

1. WRITE Present_Value = X
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY In_Process = TRUE
4. CHECK (for the externally visible actions and verify that there is a post delay)
5. VERIFY In_Process = FALSE
6. VERIFY All_Writes_Successful = TRUE

7.3.2.9.2 Quit on Failure Test

Purpose: To verify that a Command object can successfully execute Quit_On_Failure procedures.

Test Concept: The IUT is configured with two action lists that include a sequence of externally visible outputs with a write somewhere in the sequence that will fail. The action lists are identical except that one has Quit_On_Failure set to TRUE and the other set to FALSE. The TD triggers both action lists. The external outputs are observed to verify that the failure procedures are properly implemented. If the IUT does not support action list configuration for this Test Concept, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object having at least two action lists, X and Y, that includes writing to a sequence of externally visible outputs. Somewhere in the sequence there shall be a write command that will fail that is followed by write commands that will succeed. Both action lists shall be identical except that list X shall have Quit_On_Failure set to TRUE and Y shall have Quit_On_Failure set to FALSE. Delays shall be configured into the Action, if necessary, sufficient to let the tester observe that In_Process equals TRUE while the action is in process. If no Command object can be so configured, then verification of In_Process being TRUE shall be skipped

Test Steps:

1. WRITE Present_Value = X
2. VERIFY In_Process = TRUE
3. CHECK (for the externally visible actions and verify that they stop when the failure occurs)
4. VERIFY In_Process = FALSE,
5. VERIFY All_Writes_Successful = FALSE
6. WRITE Present_Value = Y

7. VERIFY In_Process = TRUE
8. CHECK (for the externally visible actions and verify that they continue to the end after the failure occurs)
9. VERIFY In_Process = FALSE,
10. VERIFY All_Writes_Successful = FALSE

7.3.2.9.3 External Writes Test

Purpose: To verify that a Command object can write to external objects

Test Concept: The IUT is configured with a Command object having an action list that includes writing to an object in the TD. The TD invokes this action list by writing the appropriate value to the Command object. The TD verifies that the IUT transmits the appropriate WriteProperty-Request.

Configuration Requirements: The IUT shall be configured with a Command object that has an Action property that contains an action list, X, that includes a command to write to the property P1 of object O1 with the value V1 in the TD.

Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

Test Steps:

1. WRITE Present_Value = X
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY In_Process = TRUE
4. BEFORE External Command Fail Time
RECEIVE WriteProperty-Request,
 'Object Identifier' = O1,
 'Property Identifier' = P1,
 'Property Value' = V1
5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (Post_Delay is present)
 WAIT (Post_Delay)
7. VERIFY In_Process = FALSE

7.3.2.9.4 Empty Action List Test

Purpose: To verify that a Command object takes no action when Present_Value is written to with a non-zero value that corresponds to an empty action list.

Test Concept: The IUT is configured with at least one empty action list. The TD triggers the action list. The external outputs are observed to verify that no changes occurred. If the IUT does not support action list configuration for this Test Concept, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object that has an Action property with at least one empty action list.

Test Steps:

1. WRITE Present_Value = (an index corresponding to an empty action list)
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY In_Process = FALSE
4. VERIFY All_Writes_Successful = TRUE
5. CHECK (if any of the actions of the Command object have externally visible results verify that no changes occurred)

7.3.2.9.5 Action 0 Test

Purpose: To verify that a Command object takes no action when Present_Value is written to with a value of 0.

Configuration Requirements: The IUT shall be configured with a Command object that has an Action property with at least one non-empty action list.

Test Steps:

1. WRITE Present_Value = 0
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY In_Process = FALSE
4. VERIFY All_Writes_Successful = TRUE
5. CHECK (if any of the actions of the Command object have externally visible results verify that no changes occurred)

7.3.2.9.6 Action_Text Test

Purpose: To verify that the size of the Action array corresponds to the size of the Action_Text array.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Array Index' = 0
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Value' = (any integer greater than 0)
3. VERIFY Action_Text = (the size of the Action array from step 2), ARRAY INDEX = 0

7.3.2.9.7 Write While In_Process is TRUE Test

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

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

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

Test Steps:

1. WRITE Present_Value = X
2. WRITE Present_Value = Y
3. IF (Protocol_Revision is present and Protocol_Revision > 10) THEN
 RECEIVE BACnet-Error-PDU,
 Error Class = OBJECT,
 Error Code = BUSY
 ELSE
 (RECEIVE BACnet-Error-PDU,
 Error Class = OBJECT,
 Error Code = BUSY) |
 (RECEIVE BACnet-Error-PDU,
 Error Class = SERVICES,
 Error Code = SERVICE_REQUEST_DENIED | OTHER)
4. CHECK (that the externally visible actions of X take place)
5. CHECK (that the externally visible actions of Y do not take place)
6. VERIFY In_Process = FALSE,
7. VERIFY All_Writes_Successful = TRUE

7.3.2.9.8 Action Size Changes Action_Text Size Test

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

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

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

Test Steps:

1. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Array Index' = 0,
 'Property Value' = 2
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY Action = 2, ARRAY INDEX = 0
4. VERIFY Action_Text = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Array Index' = 0,
 'Property Value' = (some value greater than 2)
6. RECEIVE BACnet-SimpleACK-PDU
7. VERIFY Action = (the value written in step 5), ARRAY INDEX = 0
8. VERIFY Action_Text = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Value' = (Action array of length 2)
10. RECEIVE BACnet-SimpleACK-PDU
11. VERIFY Action = 2, ARRAY INDEX = 0
12. VERIFY Action_Text = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Value' = (Action array of length greater than 2)
14. RECEIVE BACnet-SimpleACK-PDU
15. VERIFY Action = (the length of the array written in step 13), ARRAY INDEX = 0
16. VERIFY Action_Text = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action,
 'Property Array Index' = 0,
 'Property Value' = 2
18. RECEIVE BACnet-SimpleACK-PDU
19. VERIFY Action = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Action_Text = 2, ARRAY INDEX = 0

7.3.2.9.9 Action_Text Size Changes Action Size Test

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

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

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

Test Steps:

1. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action_Text,
 'Property Array Index' = 0,
 'Property Value' = 2
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY Action_Text = 2, ARRAY INDEX = 0
4. VERIFY Action = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action_Text,
 'Property Array Index' = 0,
 'Property Value' = (some value greater than 2)
6. RECEIVE BACnet-SimpleACK-PDU
7. VERIFY Action_Text = (the value written in step 5), ARRAY INDEX = 0
8. VERIFY Action = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action_Text,
 'Property Value' = (Action_Text array of length 2)
10. RECEIVE BACnet-SimpleACK-PDU
11. VERIFY Action_Text = 2, ARRAY INDEX = 0
12. VERIFY Action = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action_Text,
 'Property Value' = (Action_Text array of length greater than 2)
14. RECEIVE BACnet-SimpleACK-PDU
15. VERIFY Action_Text = (the length of the array written in step 13), ARRAY INDEX = 0
16. VERIFY Action = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Command object being tested),
 'Property Identifier' = Action_Text,
 'Property Array Index' = 0,
 'Property Value' = 2
18. RECEIVE BACnet-SimpleACK-PDU
19. VERIFY Action_Text = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Action = 2, ARRAY INDEX = 0

7.3.2.10 Device Object Tests

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

7.3.2.10.1 Active_COV_Subscriptions SubscribeCOV Test

Purpose: This test case verifies that the IUT correctly updates the Active_COV_Subscriptions property when COV subscriptions are created, cancelled and timed-out using SubscribeCOV.

Test Concept: INC1, INC2, and INC3 are each not present if the property is not numeric; present if a valid Increment was provided in the subscription; and optionally present otherwise.

Configuration Requirements: In this test, the tester shall choose three standard objects, O₁, O₂, and O₃, for which the device supports SubscribeCOV. O₁, O₂, and O₃ are not required to refer to different objects. The tester shall also choose three non-zero unique process identifiers, P₁, P₂, and P₃, and three non-zero lifetimes L₁, L₂, and L₃. Lifetime L₁ shall be

long enough to allow the initial part of the test to run through to step 14. Lifetimes L_2 and L_3 shall be long enough for the whole test to be completed without expiring.

The IUT shall start the test with no entries in its Active_COV_Subscriptions property.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P_1 ,
 'Monitored Object Identifier' = O_1 ,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L_1
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = P_1 ,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O_1 ,
 'Time Remaining' = (a value approximately equal to L_1),
 'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF P_1 is numeric THEN
 VERIFY Active_COV_Subscriptions = { { {TD, P_1 }, { O_1 , Present_Value}, TRUE, (a value less than L_1),
 (INC₁ : not present or a valid Increment) } }
- ELSE
 VERIFY Active_COV_Subscriptions = { { {TD, P_1 , { O_1 , Present_Value }, TRUE, (a value less than L_1),
 (INC₁: not present)} }
6. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P_2 ,
 'Monitored Object Identifier' = O_2 ,
 'Issue Confirmed Notifications' = FALSE,
 'Lifetime' = L_2
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = P_2 ,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O_2 ,
 'Time Remaining' = (a value approximately equal to L_2),
 'List of Values' = (values appropriate to the object type of the monitored object)
9. VERIFY Active_COV_Subscriptions = { { {TD, P_1 }, { O_1 , Present_Value}, TRUE, (a value less than L_1),
 INC₁},
 { {TD, P_2 }, { O_2 , Present_Value}, FALSE, (a value less than L_2),
 (INC₂: not present if the property is not numeric; present if a valid
 Increment was provided in the subscription; optionally
 present otherwise) } } }
10. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P_3 ,
 'Monitored Object Identifier' = O_3 ,
 'Issue Confirmed Notifications' = FALSE,
 'Lifetime' = L_3
11. RECEIVE BACnet-SimpleACK-PDU
12. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = P_3 ,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O_3 ,
 'Time Remaining' = (a value approximately equal to L_3),
 'List of Values' = (values appropriate to the object type of the monitored object)

13. IF P_3 is numeric THEN
 VERIFY Active_COV_Subscriptions = { { {TD, P_1 }, { O_1 , Present_Value}, TRUE, (a value less than L_1),
 INC₁},
 { {TD, P_2 }, { O_2 , Present_Value}, FALSE, (a value less than L_2),
 INC₂},
 { {TD, P_3 }, { O_3 , Present_Value}, FALSE, (a value less than L_3),
 INC₃: : not present or a valid Increment} }
 ELSE
 VERIFY Active_COV_Subscriptions = { { {TD, P_1 }, { O_1 , Present_Value}, TRUE, (a value less than L_1),
 INC₁ },
 { {TD, P_2 }, { O_2 , Present_Value}, FALSE, (a value less than L_2),
 INC₂ },
 { {TD, P_3 }, { O_3 , Present_Value}, FALSE, (a value less than L_3),
 (INC₃: not present)} }
 14. WAIT L_1 + the IUT's timer granularity
 15. VERIFY Active_COV_Subscriptions = { { {TD, P_2 }, { O_2 , Present_Value}, FALSE, (a value less than L_2),
 INC₂ (a valid Increment if the property is REAL)},
 { {TD, P_3 }, { O_3 , Present_Value}, FALSE, (a value less than L_3),
 INC₃ (a valid Increment if the property is REAL)} }
 16. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P_3 ,
 'Monitored Object Identifier' = O_3
 17. RECEIVE BACnet-SimpleACK-PDU
 18. VERIFY Active_COV_Subscriptions = { { {TD, P_2 }, { O_2 , Present_Value}, FALSE, (a value less than L_2),
 INC₂ (a valid Increment if the property is REAL)} }
 19. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P_2 ,
 'Monitored Object Identifier' = O_2
 20. RECEIVE BACnet-SimpleACK-PDU
 21. VERIFY Active_COV_Subscriptions = { }

7.3.2.10.2 Active_COV_Subscriptions SubscribeCOVProperty Test

Purpose: This test case verifies that the IUT correctly updates the Active_COV_Subscriptions property when COV subscriptions are created, cancelled and timed-out using SubscribeCOVProperty.

Configuration Requirements: In this test, the tester shall choose three objects and properties, O_1 , O_2 , and O_3 , for which the device supports SubscribeCOVProperty. O_1 , O_2 , and O_3 are not required to refer to different objects. The tester shall also choose three non-zero unique process identifiers, P_1 , P_2 , and P_3 , and three non-zero lifetimes, L_1 , L_2 and L_3 . Lifetime L_1 shall be long enough to allow the initial part of the test to run through to step 14. Lifetimes L_2 and L_3 shall be long enough for the whole test to be completed without expiring.

Test Steps: The test steps for this test case are identical to the test steps in Clause 7.3.2.10.1 except that SubscribeCOVProperty is used instead of SubscribeCOV and the Active_COV_Subscriptions entries shall reflect the property actually subscribed to (and not Present_Value if the subscribed property is not Present_Value).

7.3.2.10.3 Successful increment of the Database_Revision property after creating an object

Purpose: To verify that the Database_Revision property of the Device object increments when an object is created. If an object cannot be created, this test shall be omitted.

Test Concept: The Database_Revision property of the Device object is read. An object is created. The Database_Revision property of the Device object is read again to verify that it incremented.

Configuration Requirements: none.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Device object),
 'Property Identifier' = Database_Revision
2. RECEIVE ReadProperty-ACK,

- 'Object Identifier' = (the Device object),
- 'Property Identifier' = Database_Revision,
- 'Property Value' = (any value = initial value)
- 3. MAKE (create an object)
- 4. TRANSMIT ReadProperty-Request,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision
- 5. RECEIVE ReadProperty-ACK,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision,
 - 'Property Value' = (greater than initial value taking wrapping into account)

7.3.2.10.4 Successful increment of the Database_Revision property after deleting an object

Purpose: To verify that the Database_Revision property of the Device object increments when an object is deleted. If an object cannot be deleted, this test shall be omitted.

Test Concept: The Database_Revision property of the Device object is read. An object is deleted. The Database_Revision property of the Device object is read again to verify that it incremented.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision
2. RECEIVE ReadProperty-ACK,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision,
 - 'Property Value' = (any value = initial value)
3. MAKE (delete an object)
4. TRANSMIT ReadProperty-Request,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision
5. RECEIVE ReadProperty-ACK,
 - 'Object Identifier' = (the Device object),
 - 'Property Identifier' = Database_Revision,
 - 'Property Value' = (greater than initial value taking wrapping into account)

7.3.2.10.5 Successful increment of the Database_Revision property after changing the Object_Name property of an object

Purpose: To verify that the Database_Revision property of the Device object increments when the Object_Name property of an object is changed. If the Object_Name property of an object cannot be changed, this test shall be omitted.

Test Concept: The Database_Revision property of the Device object is read. The Object_Name property of an object is changed. The Database_Revision property of the Device object is read again to verify that it incremented.

Configuration Requirements: none.

Test Steps:

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

5. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Device object),
 'Property Identifier' = Database_Revision,
 'Property Value' = (greater than initial value taking wrapping into account)

7.3.2.10.6 Successful increment of the Database_Revision property after changing the Object_Identifier property of an object

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

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

Configuration Requirements: none.

Test Steps:

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

7.3.2.10.7 Max_Segments_Accepted at least the minimum

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

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

Test Steps:

1. IF Segmentation_Supported = SEGMENTED_TRANSMIT | SEGMENTED_NONE THEN
 VERIFY (Max_Segments_Accepted = 1)
 ELSE
 VERIFY (Max_Segments_Accepted > 1)

7.3.2.10.8 Ensure UTC_Offset is Configurable

Purpose: This test verifies UTC_Offset is configurable and accepts from -1440 to +1440.

Test Concept: For each value in a set of valid values across the acceptable range for UTC_Offset, verify that the UTC_Offset can be configured with the value.

Configuration Requirements: If the Protocol_Revision of the device is less than 18, this test shall be skipped.

Test Steps:

1. REPEAT UO = (-1440, -780, 0, +780, +1440, and 2 other values which are multiples of 15 minutes) DO {
 IF (UTC_Offset is writable) THEN
 WRITE UTC_Offset = UO

```

ELSE
    MAKE UTC_Offset = UO
    VERIFY (Device Object) UTC_Offset = UO
}

```

7.3.2.10.9 Ensure Device Object_Name is Configurable

Purpose: This test verifies Device Object_Name value is configurable as per BACnet Clause 12.1.1.4.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object_Name
2. MAKE (Configure the device Object_Name to a value other than S)
3. VERIFY (Device, IUT), Object_Name $\neq$ S

7.3.2.10.10 Ensure Device Object_Identifier is Configurable

Purpose: This test verifies Device Object_Identifier property value is configurable as per BACnet Clause 12.1.1.3.

Configuration Requirements: none.

Test Steps:

1. READ S = (Device, IUT), Object_Identifier
2. MAKE (Configure the device Object_Identifier to a value other than S)
3. VERIFY (Device, IUT), Object_Identifier $\neq$ S

7.3.2.11 Event Enrollment Object Test

In addition to tests in this section, additional tests to verify the functionality of the Event Enrollment object are covered by the tests for initiating event notifications in 8.4 and 8.5. If the IUT supports monitoring objects outside of the IUT one of the tests in 8.4 and in 8.5 shall be used to demonstrate this capability. This will require providing an additional BACnet device configured appropriately.

7.3.2.11.1 Event_Type Test

Purpose: This test case verifies that Event_Type property of an Event Enrollment object properly tracks changes to the Event_Parameters property.

Test Concept: Write to the Event_Parameters property and verify that the Event_Type tracks.

Configuration Requirements: The IUT shall be configured with an Event Enrollment object, E1, having a writable Event_Parameters property that will accept a value with a different algorithm choice. The tester shall choose a valid Event_Parameters value, EP1, with an event type value, ET1, that the IUT will accept where the ET1 differs from the Event_Type already existing in E1.

Test Steps:

1. VERIFY Event_Type $\neq$ ET1
2. TRANSMIT WriteProperty-Request,
 'Object Identifier' = E1,
 'Property Identifier' = Event_Parameters,
 'Property Value' = EP1
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY Event_Type = ET1

7.3.2.12 File Object Test

All necessary tests for functionality of the File object are covered by tests for execution of the file access services in 9.12 and 9.13.

7.3.2.13 Global Group Object Tests

7.3.2.13.1 Resizing Group_Member_Names by Writing Group_Members Property Test

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

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

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

Test Steps:

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

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

7.3.2.13.2 Resizing Group_Members by Writing Group_Member_Names Property Test

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

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

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

Test Steps:

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

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

7.3.2.13.3 Global Group Present_Value, Out_Of_Service and Status_Flags Test

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

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

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

Test Steps:

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

7.3.2.13.4 Reliability MEMBER_FAULT Test

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

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

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

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

Test Steps:

1. MAKE (M1 Status_Flags = {?, TRUE, ?, ?})
2. WAIT (W1)

3. VERIFY Member_Status_Flags = {?, TRUE, ?, ?}
4. IF (Reliability is present) THEN
 VERIFY Reliability = MEMBER_FAULT

7.3.2.13.5 Reliability COMMUNICATION_FAILURE Test

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

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

Configuration Requirements: The IUT shall be configured with a Global Group object with the Group_Members containing a member M1 at index N1 that can be made to discontinue communications. The Out_Of_Service property of the Global Group object must remain FALSE throughout the test. W1 is the maximum time it takes for the Global Group to receive an update from M1.

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

Test Steps:

1. MAKE (M1 discontinue communications)
2. WAIT (W1)
3. VERIFY Reliability = COMMUNICATION_FAILURE
4. IF (Reliability is present) THEN
 VERIFY Reliability = COMMUNICATION_FAILURE
5. VERIFY Member_Status_Flags = {?, FALSE, ?, ?}

7.3.2.13.6 Present_Value Tracking and Reliability Test

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

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

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

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

Test Steps:

1. VERIFY Reliability $\neq$ NO_FAULT_DETECTED
2. MAKE (M1 = X1)
3. WAIT (W1)
4. X2 = READ Present_Value, ARRAY INDEX = N1
5. VERIFY X1 = X2

7.3.2.13.7 Present_Value Tracking Test

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

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

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

Test Steps:

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

7.3.2.13.8 COVU_Period and COVU_Recipients Zero Test

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

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

Configuration Requirements: At the start of the test, O1 shall be configured with a non-zero COVU_Period and a non-empty COV_Recipient property.

Test Steps:

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

7.3.2.14 Group Object Test

Purpose: To verify that the Present_Value of a Group properly tracks the values of the properties of the objects that make up the group.

Test Concept: The Present_Value of a Group object is read. Each of the object, property combinations that make up the membership of the Group is also read. The values are compared to verify that they match. The value of one of the Group members is changed. The Present_Value of the Group is read again to verify that it correctly tracks the change.

Configuration Requirements: The IUT shall be configured with a Group object that has at least two members. One of the group members shall be changeable by the WriteProperty service or some other mechanism provided by the vendor. The value of the properties that make up the Group shall remain static for the duration of the test except for changes made as part of the test procedure.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = List_Of_Group_Members
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = List_Of_Group_Members,
 'Property Value' = (any valid list of group members)
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = Present_Value
4. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = List_Of_Group_Members,
 'Property Value' = (any valid set of values consistent with the properties that make up the group)
5. REPEAT X = (each object, property combination returned in the List_Of_Group_Members in step 2) DO {
 VERIFY X = (the same value that was returned for this group member in step 4)}
6. IF (a property value of a group member is changeable) THEN
 IF (the changeable group member property value is writable) THEN
 WRITE (the writable property that is a member of the group) = (a value different from its current value)
 ELSE
 MAKE (the changeable group member property value different from its current value)
7. WAIT **Internal Processing Fail Time**
8. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = Present_Value
9. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Group object being tested),
 'Property Identifier' = List_Of_Group_Members,
 'Property Value' = (the same set of values received in step 4 except for the value changed in step 6)
10. REPEAT X = (each object, property combination returned in the List_Of_Group_Members in step 2) DO {
 VERIFY X = (the same value that was returned for this group member in step 9)}

7.3.2.15 Life Safety Point Object Tests

7.3.2.15.1 Tracking Value Test

Purpose: To verify that the Present_Value and Tracking_Value properties of a Life Safety Point object track when the object is in a NORMAL state. This test can be omitted when the optional Tracking Value property is not supported.

Test Steps:

1. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
2. VERIFY Tracking Value = Present Value
3. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state but different from the previous value)
4. VERIFY Tracking Value = Present Value

7.3.2.15.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.15.3 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Life Safety Point objects are covered in 8.4.8.

7.3.2.15.4 Mode Tests

Tests to verify reporting capabilities for Life Safety Point objects when the Mode property changes are covered in 8.4.8.

7.3.2.15.5 Writable Tracking_Value

Purpose: This test case verifies that Tracking_Value is writable when Out_Of_Service is TRUE.

Test Concept: It verifies the interrelationship between the Tracking_Value, Status Flags, and Present_Value properties. If the Out_Of_Service property of the object under test is not writable, and the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to Life Safety Zone and Life Safety point object.

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

Test Steps:

1. MAKE (Out_Of_Service TRUE)
2. VERIFY Out_Of_Service = TRUE
3. VERIFY Status_Flags = (?, FALSE, ?, TRUE)
4. MAKE (Event_State = Normal)
5. VERIFY Event_State = Normal
6. WRITE Tracking_Value = (X: any value that corresponds to an Event_State of NORMAL)
7. VERIFY Tracking_Value = X
8. VERIFY Present_Value = X

7.3.2.15.6 Supports Writable Mode Property

Purpose: To verify that the Mode property takes one of the values found in the Accepted_Modes property.

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

Test Steps:

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

7.3.2.15.7 Support Operation_Expected Property

Purpose: To verify that the Operation_Expected property takes on the value of ConfirmedEventNotification-Request.

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

Test Steps:

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

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

3. VERIFY Operation_Expected = X (operation expected in the step 2)

7.3.2.15.8 Support Writable Member_Of Property

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

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

Test Steps:

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

7.3.2.15.9 Silenced Property Test

Purpose: This test verifies the behavior of Silenced property.

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

Since the result of any specific LifeSafetyOperation is a local matter, the expected actions when an operation is applied to an object is a local matter. In order to apply this test, the tester selects an initial Silenced state and a BACnetLifeSafetyOperation. The tester then verifies that the expected Silenced state, as specified by the vendor, is the result of the life safety operation on the object.

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

Test Steps:

1. InitialSilencedState = READ Silenced
2. MAKE (the object change its silenced state)
3. VERIFY Silenced = Other than InitialSilencedState
4. TRANSMIT LifeSafetyOperation-Request,

'Requesting Process Identifier' = (any valid identifier),
 'Requesting Source' = (any valid character string),
 'Request' = (any supported LifeSafetyOperation request transmitted to silence
 the sounder/strobe),
 'Object Identifier' = (the selected object)

5. RECEIVE BACnet-SimpleACK-PDU
6. CHECK (Sounder/Strobe inactive)
7. ResultingSilencedState = READ Silenced
8. CHECK (the ResultingSilencedState is equal to the InitialSilencedState, modified by the LifeSafetyOperation request transmitted)

7.3.2.16 Life Safety Zone Object Tests

7.3.2.16.1 Tracking Value Test

Purpose: To verify that the Present_Value and Tracking_Value properties of a Life Safety Zone object track when the object is in a NORMAL state. This test can be omitted when the optional Tracking Value property is not supported.

Test Steps:

1. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
2. VERIFY Tracking Value = Present Value
3. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state but different from the previous value)
4. VERIFY Tracking Value = Present Value

7.3.2.16.2 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.16.3 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Life Safety Point objects are covered in 8.4.8.

7.3.2.16.4 Mode Tests

Tests to verify reporting capabilities for Life Safety Point objects when the Mode property changes are covered in 8.4.8.

7.3.2.17 Loop Object Test

7.3.2.17.1 Manipulated_Variable_Reference Tracking

Purpose: To verify that the property referenced by Manipulated_Variable_Reference tracks the Present_Value of the Loop object.

Configuration Requirements: The IUT shall be configured so that the control output of the Loop object being tested remains constant for the duration of the test. If this is not possible then this test shall be omitted.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Manipulated_Variable_Reference
2. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Manipulated_Variable_Reference,
 'Property Value' = (any valid object property reference)
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Priority_For_Writing
4. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Priority_For_Writing,
 'Property Value' = (any priority from 1 to 16)

5. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Present_Value
6. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Present_Value,
 'Property Value' = (any valid value)
7. IF (the manipulated variable reference is commandable) THEN
 VERIFY (the manipulated variable reference object),
 (the referenced property) = (the Present_Value from step 6),
 ARRAY INDEX = (the Priority_For_Writing from step 4)
 ELSE
 VERIFY (the manipulated variable reference object),
 (the referenced property) = (the Present_Value from step 6)

7.3.2.17.2 Controlled_Variable_Reference Tracking

Purpose: To verify that Controlled_Variable_Value tracks the property referenced by Controlled_Variable_Reference.

Configuration Requirements: The IUT shall be configured so that the controlled variable value of the Loop object being tested remains constant for the duration of the test. If this is not possible then this test shall be omitted.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Controlled_Variable_Reference
2. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Controlled_Variable_Reference,
 'Property Value' = (any valid object property reference)
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Controlled_Variable_Value
4. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Controlled_Variable_Value,
 'Property Value' = (any valid value)
5. VERIFY (the controlled variable reference object),
 (the referenced property) = (the Controlled_Variable_Value from step 4)

7.3.2.17.3 Setpoint_Reference Tracking

Purpose: To verify that Setpoint tracks the property referenced by Setpoint_Reference.

Configuration Requirements: The Loop object shall be configured to determine the setpoint based on an object and property specified in the Setpoint_Reference property. This referenced setpoint shall remain constant for the duration of the test except as noted in the test steps. If such a control loop cannot be configured this test shall be omitted.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Setpoint_Reference
2. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Setpoint_Reference,
 'Property Value' = (any valid object property reference)
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Setpoint

4. RECEIVE BACnet-ComplexACK-PDU,
 'Object Identifier' = (the Loop object being tested),
 'Property Identifier' = Setpoint,
 'Property Value' = (any valid value)
5. VERIFY (the setpoint reference object),
 (the referenced property) = (the setpoint from step 4)
6. IF (the referenced property of the setpoint reference object is writable) THEN
 WRITE (the referenced property) = (a different value)
 ELSE
 MAKE (the referenced property take on a new value)
7. VERIFY (the Loop object being tested),
 Setpoint = (the new value of the referenced property)

7.3.2.17.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Loop objects are covered in 8.4.5.

7.3.2.18 Multi-state Input Object Test

7.3.2.18.1 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.18.2 Number_Of_States and State_Text

Purpose: To verify that the size of the State_Text array corresponds to the Number_Of_States.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = (any integer greater than 0)
3. VERIFY State_Text = (the number of states from step 2), ARRAY INDEX = 0

7.3.2.18.3 Intrinsic Reporting Tests

Tests to verify intrinsic reporting for Multi-state Input Objects are covered in 8.4.4.

7.3.2.18.4 Input Tracking Test

Purpose: To verify the ability to track and represent the value of a multi-state input.

Configuration Requirements: The IUT shall be configured with a multi-state input that can be externally controlled during the test. If the IUT cannot be configured with such an input, this test shall be omitted.

Test Steps:

1. REPEAT X = (at least two values meeting the functional range requirements of 7.2.1) DO {
 MAKE (the real multi-state input X)
 VERIFY Present_Value = X
 }

7.3.2.18.5 Number_Of_States and State_Text Size Change Test

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

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

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

Test Steps:

1. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = 2
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY Number_Of_States = 2
4. VERIFY State_Text = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = (some value greater than 2)
6. RECEIVE BACnet-SimpleACK-PDU
7. VERIFY Number_Of_States = (the value written in step 5)
8. VERIFY State_Text = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = State_Text,
 'Property Value' = (State_Text array of length 2)
10. RECEIVE BACnet-SimpleACK-PDU
11. VERIFY Number_Of_States = 2
12. VERIFY State_Text = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = State_Text,
 'Property Value' = (State_Text array of length greater than 2)
14. RECEIVE BACnet-SimpleACK-PDU
15. VERIFY Number_Of_States = (the length of the array written in step 13)
16. VERIFY State_Text = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = 2
18. RECEIVE BACnet-SimpleACK-PDU
19. VERIFY State_Text = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Number_Of_States = 2

7.3.2.19 Multi-State Output Object Test

7.3.2.19.1 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.19.2 Number_Of_States and State_Text

Purpose: To verify that the size of the State_Text array corresponds to the Number_Of_States.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Multi-state Output object being tested),
 'Property Identifier' = Number_Of_States
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Multi-state Output object being tested),
 'Property Identifier' = Number_Of_States,

'Property Value' = (any integer greater than 0)

3. VERIFY State_Text = (the number of states from step 2), ARRAY INDEX = 0

7.3.2.19.3 Prioritized Commands Tests

Tests to verify prioritized commands of Multi-state Output objects are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.19.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Multi-state Output objects are covered in 8.4.4.

7.3.2.19.5 Output Tracking Test

Purpose: To verify the ability to represent and implement a physical multi-state output.

Configuration Requirements: The IUT shall be configured with a multi-state output that can be observed during the test. If the IUT cannot be configured with such an output, this test shall be omitted.

Test Steps:

1. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {
 WRITE Present_Value = X
 VERIFY Present_Value = X
 CHECK (verify that the output is X)
 }

7.3.2.19.6 Number_Of_States and State_Text Size Change Test

The test to verify the Number_Of_States value and State_Text array size of Multi-state Output objects are defined in 7.3.2.18.5. Run the test using a Multi-state Output object.

7.3.2.20 Multi-State Value Object Test

7.3.2.20.1 Out_Of_Service, Status_Flags, and Reliability Tests

Tests to verify the functionality of the Out_Of_Service property and the links between Status_Flags, Reliability, and Out_Of_Service are covered in 7.3.1.1.

7.3.2.20.2 Number_Of_States and State_Text

Purpose: To verify that the size of the State_Text array corresponds to the Number_Of_States.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the Multi-state Value object being tested),
 'Property Identifier' = Number_Of_States
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the Multi-state Value object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = (any integer greater than 0)
3. VERIFY State_Text = (the number of states from step 2), ARRAY INDEX = 0

7.3.2.20.3 Prioritized Commands Tests

Tests to verify prioritized commands of Multi-state Value objects are covered in 7.3.1.2 and 7.3.1.3.

7.3.2.20.4 Intrinsic Reporting Tests

Tests to verify intrinsic reporting capabilities for Multi-state Value objects are covered in 8.4.2.

7.3.2.20.5 Number_Of_States and State_Text Size Change Test

Test Steps:

Tests to verify the Number_Of_States value and State_Text array size of Multi-state Value objects are defined in 7.3.2.18.5. Run the tests using a Multi-state Value object.

7.3.2.21 Notification Class Object

7.3.2.21.1 Priority Tests

Purpose: To verify that the IUT implements the functionality of the Priority property of the Notification Class object when initiating even notifications.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to the Notification Class object. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL, from OFFNORMAL back to NORMAL, from NORMAL to FAULT, and from FAULT back to NORMAL. For each state transition the appropriate use of priority in the resulting event notification will be verified. It must be possible to trigger the events of this test or the test result is considered to be a failure.

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The Notification Class object shall be configured with separate, distinct Priority values for TO-OFFNORMAL, TO-NORMAL, and TO-FAULT transitions. All Event_Enable bits shall be set to TRUE. The referenced event-triggering property shall be set to a value that results in a NORMAL condition.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that is OFFNORMAL)
ELSE
 MAKE (pMonitoredValue have a value that is OFFNORMAL)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = (values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentState = OFFNORMAL
7. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that is NORMAL)
ELSE
 MAKE (pMonitoredValue have a value that is NORMAL)
8. WAIT (pTimeDelayNormal)
9. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFNORMAL,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)

10. TRANSMIT BACnet-SimpleACK-PDU
 11. VERIFY pCurrentState = NORMAL
 12. IF (the event-triggering object can be placed into a fault condition) THEN {
 MAKE (a condition exist that will cause the object to generate a TO_FAULT transition)
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO_FAULT transition),
 'Event Type' = (IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = FAULT,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 VERIFY pCurrentState = FAULT
 MAKE (a condition exist that will cause the object to generate a TO_NORMAL transition)
 WAIT (pTimeDelayNormal)
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = FAULT,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 VERIFY pCurrentState = NORMAL
 }
 }

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

7.3.2.21.1.1 Network Priority Test

Purpose: To verify that the correct network priority is used when transmitting EventNotification requests.

Test Concept: The IUT is made to generate a ConfirmedEventNotification with a specific priority. The network priority in the NPCI is checked to ensure that it is correct based on the priority of the event.

Configuration Requirements: The IUT is configured with an event generating object E1 that refers to Notification Class object N1.

Test Steps:

1. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 0..63) THEN {
 MAKE (The IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 0..63)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Network Priority' = 3
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (a value in the range 0..63),
 'Event Type' = (any valid value),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = (any valid value),
 'To State' = (any valid value),
 'Event Values' = (any valid valid)
 TRANSMIT BACnet-SimpleACK-PDU
 }
2. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 64..127) THEN {
 MAKE (The IUT initiate a ConfirmedEventNotification-Request with an event priority in the range 64..127)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Network Priority' = 2
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (a value in the range 64..127),
 'Event Type' = (any valid value),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = (any valid value),
 'To State' = (any valid value),
 'Event Values' = (any valid valid)
 TRANSMIT BACnet-SimpleACK-PDU
 }
3. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 128..191) THEN {
 MAKE (The IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 128..191)
 BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Network Priority' = 1
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (a value in the range 128..191),

```

        'Event Type' = (any valid value),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = (any valid value),
        'To State' = (any valid value),
        'Event Values' = (any valid valid)
    TRANSMIT BACnet-SimpleACK-PDU
}
4. IF (the IUT can generate ConfirmedEventNotifications with an event priority in the range 192..255) THEN {
    MAKE (The IUT initiate a ConfirmedEventNotification-Request with a event priority in the range 192..255)
    BEFORE Notification Fail Time
    RECEIVE ConfirmedEventNotification-Request,
        'Network Priority' = 0
        'Process Identifier' = (any valid process ID),
        'Initiating Device Identifier' = IUT,
        'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
        'Time Stamp' = (any valid time stamp),
        'Notification Class' = (the configured notification class),
        'Priority' = (a value in the range 192..255),
        'Event Type' = (any valid value),
        'Message Text' = (optional, any valid message text),
        'Notify Type' = EVENT | ALARM,
        'AckRequired' = TRUE | FALSE,
        'From State' = (any valid value),
        'To State' = (any valid value),
        'Event Values' = (any valid valid)
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.21.2 Ack_Required Tests

These tests verify that the values of the Notification Class' Ack_Required property are properly reflected in the issuance of event notification messages.

7.3.2.21.2.1 Ack_Required False Test

Purpose: To verify that if the Ack_Required property indicates that event notifications do not require acknowledgment, then the AckRequired parameter of the notification message conveys that fact. If the IUT does not support unacknowledged event notifications this test shall be omitted.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to the Notification Class object. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL, from OFFNORMAL back to NORMAL, from NORMAL to FAULT, and from FAULT back to NORMAL. For each state transition the appropriate value for the 'AckRequired' parameter is verified.

Configuration Requirements: The configuration requirements are identical to those in 7.3.2.21.1 except for an additional requirement that the value of the Ack_Required property shall be B'000' indicating that no acknowledgments are required.

Test Steps: The test steps are identical to those in 7.3.2.21.1 with the additional constraint that the 'AckRequired' parameter shall be FALSE in all event notification messages.

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

7.3.2.21.3 Recipient_List Tests

The test cases defined in this clause verify the correct processing of the various parameters of the BACnetDestination entries in the Recipient_List property.

7.3.2.21.3.1 ValidDays Test

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

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

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

Test Steps:

1. (TRANSMIT TimeSynchronization-Request,
 'Time' = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days)) |
 (TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days, converted to UTC)) |
 MAKE (the local date and time = (any time within the window defined by From Time and To Time in the BACnetDestination that corresponds to one of the valid days))
2. WAIT (pTimeDelay + **Notification Fail Time**)
3. VERIFY pCurrentState = NORMAL
4. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that is OFFNORMAL)
 ELSE
 MAKE (pMonitoredValue have a value that is OFFNORMAL)
5. WAIT (pTimeDelay)
6. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = (values appropriate to the event type)
7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY pCurrentState = OFFNORMAL
9. (TRANSMIT TimeSynchronization-Request,
 'Time' = (any time within the window defined by From Time and To time in the BACnetDestination that

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

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

7.3.2.21.3.2 FromTime and ToTime Test

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

Test Concept: The case where the local date and time fall within the window defined by the From Time and To Time parameters is covered by the ValidDays test in 7.3.2.21.3.1. This test uses the same IUT configuration and sets the local time to a value that is one of the ValidDays but outside of the window defined by the From Time and To Time parameters. The objective is to verify that an event notification message is not transmitted when the event is triggered. For devices that implement a read-only Recipient_List property for all instances of Notification Class objects and are exclusively configured for all times (From Time set to 00:00:00.0, To Time set to 23:59:59.99), this test shall be omitted. For devices that implement a writeable Notification Class Recipient_List property for all instances of Notification Class objects, and exclusively accept all times as the only permitted configuration, this test shall be omitted.

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

Test Steps:

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

7.3.2.21.3.3 IssueConfirmedNotifications Test

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

Configuration Requirements: The IUT shall be configured with two or more instances of the Notification Class object and event-generating objects that are linked to the Notification Class objects. The event-generating objects may be objects that support intrinsic reporting or they may be Event Enrollment objects. The event-generating objects shall have the

Event Enable property configured to transmit notification messages for all event transitions. The event-generating objects shall be configured to be in a NORMAL event state at the start of the test. One Notification Class object, N_1 , shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, N_2 , shall be configured with Issue Confirmed Notifications equal to FALSE. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions. The local date and time shall be configured to be within the window defined by From Time and To Time on one of the ValidDays.

In the test description below "X1" and "X2" are used to designate the pMonitoredValue algorithm parameter linked to Notification objects "N1" and "N2" respectively.

Test Steps:

1. VERIFY (the event-generating object linked to N1), pCurrentState = NORMAL
2. VERIFY (the event-generating object linked to N2), pCurrentState = NORMAL
3. WAIT (pTimeDelay + **Notification Fail Time**)
4. IF (X1 is writable) THEN
WRITE X1 = (a value that is OFFNORMAL)
ELSE
MAKE (X1 a value that is OFFNORMAL)
5. WAIT (pTimeDelay)
6. BEFORE **Notification Fail Time**
RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object linked to N1),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = (values appropriate to the event type)
7. IF (X2 is writable) THEN
WRITE X2 = (a value that is OFFNORMAL)
ELSE
MAKE (X2 a value that is OFFNORMAL)
8. WAIT (pTimeDelay)
9. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object linked to N2),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = (values appropriate to the event type)

Notes to Tester: If the Recipient_List is writable and the Issue Confirmed Notifications can be changed then this test can be performed using only one Notification Class object by writing to the Recipient_List in order to change between confirmed and unconfirmed notifications.

7.3.2.21.3.4 Transitions Test

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

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

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

Test Steps:

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

'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = NORMAL,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 ELSE
 CHECK (verify that the IUT did not transmit an event notification message)
 10. VERIFY pCurrentState = NORMAL
 11. IF (the event-triggering object can be placed into a fault condition) THEN {
 MAKE (a condition exist that will cause the object to generate a TO_FAULT transition)
 BEFORE **Notification Fail Time**
 IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the event-generating object configured for this test),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the class corresponding to the object being tested),
 'Priority' = (the value configured to correspond to a TO-FAULT transition),
 'Event Type' = (IF (Protocol_Revision < 13) THEN
 (any valid event type),
 ELSE
 CHANGE_OF_RELIABILITY),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = FAULT,
 'Event Values' = (values appropriate to the event type)
 TRANSMIT BACnet-SimpleACK-PDU
 ELSE
 CHECK (verify that the IUT did not transmit an event notification message)
 VERIFY pCurrentState = FAULT
 }

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service, in which case the TD shall skip all of the steps in which a BACnet-SimpleACK-PDU is sent.

7.3.2.21.3.5 Recipient_List Property Supports Device Identifier Recipients Test

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

Test Concept: The tester shall select a single event generating object E in the IUT that references Notification Class object N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a Device Identifier, D, for a device that the IUT is not already aware of. The TD, acting as device D, shall be located on a different network than the IUT to ensure that the IUT is capable of binding to recipients located on any network. For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is an address form of the Recipient component in all instances, this test shall be omitted. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

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

Test Steps:

1. WRITE N.RecipientList = ({all days, all times, D, any process ID, FALSE, all transitions})
2. MAKE (a condition exist that will cause E to generate an event transition)
3. WAIT D1
4. BEFORE (**Notification Fail Time** plus the amount of time the IUT takes to perform device discovery)
 - RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' =	PID,
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	E,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(N's instance),
'Priority' =	(any valid priority),
'Event Type' =	(any valid event type),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM EVENT,
'AckRequired' =	TRUE FALSE,
'From State' =	(any valid event state),
'To State' =	(any valid event state),
'Event Values' =	(values appropriate to the event type)

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

7.3.2.21.3.6 Recipient_List Property Supports Network Address Recipients

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

Test Concept: The tester shall select a single event-generating object E in the IUT that references Notification Class object N. The tester shall add an entry into the Recipient_List of the associated Notification Class object that specifies a BACnetAddress A, where A is a unicast or is a local, remote, or global broadcast address. For devices that implement a read-only Recipient_List property for all instances of Notification Class objects, and there is a Device Identifier form of the Recipient component in all instances, this test shall be skipped. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Test Steps:

1. WRITE N.RecipientList = ({all days, all times, A, any process ID, FALSE, all transitions})
2. MAKE (a condition exist that will cause E to generate an event transition)
3. WAIT D1
4. BEFORE **Notification Fail Time**
 - RECEIVE
 - DESTINATION = A,
 - UnconfirmedEventNotification-Request,

'Process Identifier' =	PID,
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	E,
'Time Stamp' =	(any valid time stamp),
'Notification Class' =	(N's instance),
'Priority' =	(any valid priority),
'Event Type' =	(any valid event type),
'Message Text' =	(optional, any valid message text),
'Notify Type' =	ALARM EVENT,
'AckRequired' =	TRUE FALSE,
'From State' =	(any valid event state),
'To State' =	(any valid event state),
'Event Values' =	(values appropriate to the event type)

7.3.2.21.3.7 Recipient_List non-volatility test

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

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

Test Steps:

1. WRITE Recipient_List = (RL: any valid value)
2. IF (ReinitializeDevice – WARMSTART execution is supported, i.e. BIBB DM-RD-B) THEN {
 - TRANSMIT ReinitializeDevice-Request
 - Reinitialized State of Device = WARMSTART
 - Password = (any valid password)
 - RECEIVE BACnet-SimpleACK-PDU
 - CHECK (Did the IUT perform a WARMSTART reboot?)
 - VERIFY Recipient_List = RL
- ELSE {
 - MAKE (power cycle the IUT to make it reinitialize)
 - VERIFY Recipient_List = RL

7.3.2.21.3.8 Read-only Recipient_List with internal Notification Forwarder objects

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

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

Test Steps:

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

7.3.2.21.3.9 Read-only Recipient_List for external Notification Forwarder Objects

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

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

Test Steps:

1. VERIFY Recipient_List = {(1, 1, 1, 1, 1, 1) -- Valid Days
 - 00:00:00.0 -- From Time
 - 23:59:59.99 -- To Time
 - (BACnetAddress: network-number = 0, zero length mac-address)
 - 0 -- Process Identifier

```

False          -- Issue Confirmed Notifications
(True, True, True) -- Transitions
}

```

7.3.2.21.3.10 Read-only Recipient_List Without Notification Forwarder Test

Purpose: This test case verifies that each read-only Recipient_List for all instances of Notification Class objects, in a device that cannot contain a Notification Forwarder object, are correctly configured. This test is only applied to devices that have a read-only Recipient_List property for all instances of Notification Class objects in a device that cannot contain a Notification Forwarder object.

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

Test Steps:

1. REPEAT X = (each Notification Class object instance in the IUT) DO
 - {VERIFY (X), Recipient_List = {--list size of one
 - {(1, 1, 1, 1, 1, 1, 1) -- Valid Days
 - 00:00:00.0 -- From Time
 - 23:59:59.99 -- To Time
 - (BACnetAddress: network-number = 0, zero length mac-address)--
 - Recipient
 - 0 -- Process Identifier
 - False -- Issue Confirmed Notifications
 - (True, True, True) -- Transitions
 - } -- end of list entry
 - } -- end of list

7.3.2.22 Program Object Tests

The Program object utilizes parameter control through its writable Program_Change property.

7.3.2.22.1 Program_Change Property Test

Purpose: To verify writability of Program_Change property.

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

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

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

Test Steps:

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

7.3.2.23 Schedule Object Tests

The Schedule object has no properties required to be writable or otherwise configurable. The following tests are designed to be performed on such a Schedule object. However, if the Schedule object is in any way configurable it shall be configured to accommodate as many of the following tests as is possible for the implementation. If it is impossible to

configure the IUT in the manner required for a particular test that test shall be omitted. If the IUT supports Schedule objects that can write outside the device this shall be demonstrated in one of the Schedule tests.

Tests of the Schedule object center upon observing the write operations scheduled to occur at specific dates and times, verified by reading the Schedule object's Present_Value property. For the test to be performed in a reasonable amount of time it is necessary to be able to alter settings of the device's clock and calendar.

For each test using a scheduled write operation, a date and time ("Date") for the write operation is determined beforehand. Tables 7-1 through 7-9 give the criteria for the Dates, designated D₁, D₂, and so on, to be used in the tests. Dates meeting these criteria may be chosen from existing schedules, or a schedule may be developed by the manufacturer to meet these criteria.

Associated with each Date D_n is a value V_n, which is the value associated with the time member of Date in the BACnetTimeValue pair. V_n may take on any primitive datatype.

7.3.2.23.1 Effective_Period Test

Purpose: To verify that Effective_Period controls the range of dates during which the Schedule object is active.

Test Concept: Two Date values are chosen by the TD based on the criteria in Table 7-1 such that one is outside of the Effective_Period and the other corresponds to a known scheduled state inside the Effective_Period. The IUT's local date and time are changed between these dates and the Present_Value property is monitored to verify that write operations occur only within the Effective_Period.

Configuration Requirements: The IUT shall be configured with a schedule object such that the time periods defined in Table 7-1 have uniquely scheduled values. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-1. Criteria for Effective_Period Test Dates and Values

Date:	Criteria:	Value:
D ₁	1. Date occurs during Effective_Period, and 2. Date is active either in Weekly_Schedule or Exception_Schedule.	V ₁
D ₂	1. Date does not occur during Effective_Period, and 2. Date is active either in Weekly_Schedule or Exception_Schedule.	V ₂ different from V ₁ .

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
| MAKE (the local date and time = D₁)
3. WAIT Schedule Evaluation Fail Time
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
| MAKE (the local date and time = D₂)
6. WAIT Schedule Evaluation Fail Time
7. VERIFY Present_Value = V₁

7.3.2.23.2 Weekly_Schedule Property Test

Purpose: To verify that Weekly_Schedule contains distinguishable schedules for each day of the week, and that a day's entire schedule can be executed.

Test Concept: The IUT's local date and time are changed sequentially to represent each day of the week as shown in Table 7-2. The Present_Value property is monitored to verify that write operations occur for each separately scheduled day.

Configuration Requirements: The IUT shall be configured with a schedule object containing a weekly schedule with seven distinguishable daily schedules meeting the requirements of Table 7-2. The local date and time shall be set such that the Present_Value property has a value other than V₁. If no schedule exists that meets these requirements and none can be configured, this test shall be omitted.

Table 7-2. Criteria for Weekly Schedule Test Dates and Values

Date:	Criteria:	Value:
D ₁	1. Date occurs during Effective_Period, 2. Date occurs on a Monday, and 3. Date is not active in Exception_Schedule.	V ₁
D ₂	1. Date occurs during Effective_Period, 2. Date occurs on a Tuesday, and 3. Date is not active in Exception_Schedule.	V ₂ is different from V ₁ .
D ₃	1. Date occurs during Effective_Period, 2. Date occurs on a Wednesday, and 3. Date is not active in Exception_Schedule.	V ₃ is different from V ₂ .
D ₄	1. Date occurs during Effective_Period, 2. Date occurs on a Thursday, and 3. Date is not active in Exception_Schedule.	V ₄ is different from V ₃ .
D ₅	1. Date occurs during Effective_Period, 2. Date occurs on a Friday, and 3. Date is not active in Exception_Schedule.	V ₅ is different from V ₄ .
D ₆	1. Date occurs during Effective_Period, 2. Date occurs on a Saturday, and 3. Date is not active in Exception_Schedule.	V ₆ is different from V ₅ .
D ₇	1. Date occurs during Effective_Period, 2. Date occurs on a Sunday, and 3. Date is not active in Exception_Schedule.	V ₇ is different from V ₆ .

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V₂
8. (TRANSMIT TimeSynchronization-Request, 'Time' = D₃) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₃) |
MAKE (the local date and time = D₃)
9. WAIT **Schedule Evaluation Fail Time**
10. VERIFY Present_Value = V₃
11. (TRANSMIT TimeSynchronization-Request, 'Time' = D₄) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₄) |
MAKE (the local date and time = D₄)
12. WAIT **Schedule Evaluation Fail Time**
13. VERIFY Present_Value = V₄
14. (TRANSMIT TimeSynchronization-Request, 'Time' = D₅) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₅) |
MAKE (the local date and time = D₅)
15. WAIT **Schedule Evaluation Fail Time**
16. VERIFY Present_Value = V₅
17. (TRANSMIT TimeSynchronization-Request, 'Time' = D₆) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₆) |
MAKE (the local date and time = D₆)
18. WAIT **Schedule Evaluation Fail Time**
19. VERIFY Present_Value = V₆
20. (TRANSMIT TimeSynchronization-Request, 'Time' = D₇) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₇) |

- MAKE (the local date and time = D₇)
21. **WAIT Schedule Evaluation Fail Time**
 22. VERIFY Present_Value = V₇
 23. REPEAT X = (the time portion of the BACnetTimeValue entries for one of the daily schedules in Table 7-2) DO {
 - (TRANSMIT TimeSynchronization-Request, 'Time' = X) |
 - (TRANSMIT UTCTimeSynchronization-Request, 'Time' = X) |
 - MAKE (the local date and time = X)
 - WAIT Schedule Evaluation Fail Time**
 - VERIFY Present_Value = (the scheduled value corresponding to time X)

7.3.2.23.3 Exception_Schedule Property Tests

If the IUT cannot be suitably configured to perform one or more of the tests in this clause it shall be omitted. The inability to make such a configuration may be due to an absent or immutable Exception_Schedule property, to limited numbers of available BACnetSpecialEvents in the Exception_Schedule, or to the unavailability of Calendar objects.

7.3.2.23.3.1 Calendar Reference Test

See Clause 7.3.2.23.10.3.1 Revision 4 Calendar Reference Test.

7.3.2.23.3.2 Calendar Entry Date Test

Purpose: To verify that a specified date appearing in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-3. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a specific date. The criteria for the dates used in the test are given in Table 7-3. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-3. Criteria for Calendar Entry Date Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: Date, 2B. Date matches calendarEntry: Date, and 2C. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, and 2. Date does not appear in any BACnetSpecialEvents, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	Other than V ₁

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. **WAIT Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.3 Calendar Entry DateRange Test

Purpose: To verify that a date appearing in an Exception_Schedule's date range enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-4. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a date range. The criteria for the dates used in the test are given in Table 7-4. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-4. Criteria for Calendar Entry DateRange Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: DateRange, 2B. Date matches BACnetCalendarEntry: DateRange, and 2C. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, and 2. Date does not appear in any BACnetSpecialEvents, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	Other than V ₁

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.4 Calendar Entry WeekNDay Month Test

Purpose: To verify that a date matching a WeekNDay's Month field in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-5. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying a month. The criteria for the dates used in the test are given in Table 7-5. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-5. Criteria for Calendar Entry WeekNDay Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay: specifies Month, 2C. Date matches calendarEntry: WeekNDay: Month, and 2D. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, and 2. Date does not appear in any BACnetSpecialEvents, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	Other than V ₁

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)

2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. **WAIT Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.5 Calendar Entry WeekNDay Week Of Month Test

Purpose: To verify that a date matching a WeekNDay's WeekOfMonth field in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-6. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying a week of the month. The criteria for the dates used in the test are given in Table 7-6. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-6. Criteria for Calendar Entry WeekNDay Week Of Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth is in the range 1..5, 2D. Date matches calendarEntry: WeekNDay: WeekOfMonth, and 2E Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth is in the range 1..5, and 2D. Date does not match calendarEntry: WeekNDay: WeekOfMonth.	Other than V ₁

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. **WAIT Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.6 Calendar Entry WeekNDay Last Week Of Month Test

Purpose: To verify that a date matching a WeekNDay's WeekOfMonth field in an Exception_Schedule enables the referencing Schedule object.

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

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the last week of the month. The criteria for the dates used in the test are given in Table 7-7. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-7. Criteria for Calendar Entry WeekNDay Last Week Of Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value 6, 2D. Date is in the last week of the month, and 2E. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value 6, and 2D. Date is not in the last week of the month.	Other than V ₁

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.7 Calendar Entry WeekNDay Day Of Week Test

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

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-8. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the day of the week. The criteria for the dates used in the test are given in Table 7-8. The local date and time shall be set such that the Present_Value property has a value other than V₁. This test shall not be performed if the Protocol_Revision property is present in the Device object and has a value of 4 or greater.

Table 7-8. Criteria for Calendar Entry WeekNDay Day of Week Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, 2C. Date falls on the specified day of the week, and 2D. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, and 2C. Date does not fall on the specified day of the week.	

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. **WAIT Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₁)

7.3.2.23.3.8 Event Priority Test

Purpose: To verify that a BACnetSpecialEvent of a higher priority takes precedence over one of lower priority when both specify the same date.

Configuration Requirements: The IUT shall be configured with a Schedule object containing two or more BACnetSpecialEvents, all active on the same date, with different eventPriority values, with distinguishable BACnetTimeValue entries. If possible all BACnetSpecialEvents shall have a BACnetTimeValue entry with identical time but different values. In the test description D₁ represents a date and time where all of the special events are active.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
2. **WAIT Schedule Evaluation Fail Time**
3. VERIFY Present_Value = (the value corresponding to the special event with the highest eventPriority)

7.3.2.23.3.9 List of BACnetTimeValue Test

Purpose: To verify that a Special_Event's entire schedule can be executed.

Test Concept: A special event is scheduled that contains multiple BACnetTimeValue entries. The local date and time are changed to values that match each of the BACnetTimeValue entries and the Present_Value property is read to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured with a Schedule object containing a BACnetSpecialEvent with two or more BACnetTimeValue entries and no BACnetSpecialEvents with a higher priority. Each BACnetTimeValue entry shall have a distinguishable value.

Test Steps:

1. REPEAT D_i = (the times used in the BACnetTimeValue pairs of the special event) DO {
(TRANSMIT TimeSynchronization-Request, 'Time' = D_i) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_i) |
MAKE (the local date and time = D_i)
WAIT Schedule Evaluation Fail Time
VERIFY Present_Value = (the value corresponding to the special event with the highest eventPriority)
}

7.3.2.23.4 Weekly_Schedule and Exception_Schedule Interaction Test

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

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

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

Test Steps:

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

7.3.2.23.5 Exception_Schedule Restoration Test

Purpose: To verify the restoration behavior in an Exception_Schedule.

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

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

Test Steps:

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

ELSE

VERIFY Present_Value = V_{last}

7.3.2.23.6 Weekly_Schedule Restoration Test

Purpose: To verify the restoration behavior in a Weekly_Schedule.

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

Configuration Requirements: The IUT shall be configured with a Schedule object that contains a Weekly_Schedule that has more than one scheduled write operation for a particular day. None of the write operations shall be scheduled for time 00:00 and there shall be no higher priority coincident BACnetSpecialEvents. In the test description D₁ represents a time between 00:00 and the time of the first scheduled write operation in the BACnetDailySchedule. V_{last} represents the value that is scheduled to be written in the last BACnetTimeValue pair for the day.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
2. WAIT **Schedule Evaluation Fail Time**
3. IF (Protocol_Revision is present AND Protocol_Revision >= 4) THEN
VERIFY Present_Value = Schedule_Default
ELSE
VERIFY Present_Value = V_{last}
4. IF (ReinitializeDevice execution is supported) THEN
TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART,
'Password' = (any valid password)
RECEIVE BACnet-SimpleACK-PDU
ELSE
MAKE (the IUT reinitialize)
5. CHECK (Did the IUT perform a WARMSTART reboot?)
6. WAIT **Schedule Evaluation Fail Time**
7. IF (Protocol_Revision is present AND Protocol_Revision >= 4) THEN
VERIFY Present_Value = Schedule_Default
ELSE
VERIFY Present_Value = V_{last}

7.3.2.23.7 List_Of_Object_Property_Reference Internal Test

Purpose: To verify that the Schedule object writes to objects and properties contained within the IUT.

Test Concept: The Schedule object is configured to write to a property of another object within the same device. The IUT's clock is then set to a time between a pair of scheduled write operations, and verification of the first write operation's data value is performed. The time is advanced to the second time, the Schedule object's Present_Value is checked, and verifications of the write operations are performed. If the IUT does not support writing to object properties within the IUT, then this test shall not be performed.

Configuration Requirements: The IUT is configured with a Schedule object containing a List_Of_Object_Property_References property that references, if possible, at least one property in another object within the IUT. The Schedule object is configured with either a Weekly_Schedule or an active Exception_Schedule, during a period where Effective_Period is active, with at least two consecutive entries with distinguishable values in the List of BACnetTimeValues, and with no Exception_Schedules at a higher priority. D₁ represents the date and time of the first of these two BACnetTimeValues, with corresponding value V₁, while D₂ and V₂ (a value distinguishable from V₁) represent the second BACnetTimeValue. A time D_t is defined to occur between D₁ and D₂.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
2. **WAIT Schedule Evaluation Fail Time**
3. VERIFY Present_Value = V₁
4. VERIFY (value of referenced property in IUT) = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V₂
8. VERIFY (value of referenced property in IUT) = V₂

7.3.2.23.8 List_Of_Object_Property_Reference External Test

Purpose: To verify that the Schedule object writes to object properties contained in a device other than the IUT.

Test Concept: The Schedule object is configured to write to a property of another object in the same device and a property of an object in the TD. The IUT's clock is then set to a time between a pair of scheduled write operations, and verification of the first write operation's data value is performed. The time is advanced to the second time, the Schedule object's Present_Value is checked, and verifications of the write operation are performed. If the IUT does not support writes to object properties contained in a device other than the IUT, then this test shall not be performed.

Configuration Requirements: The TD is configured to indicate that it supports the WriteProperty-Request service but not WritePropertyMultiple-Request. The IUT is configured with a Schedule object containing a List_Of_Object_Property_References property that references a property of an object contained in the TD. The Schedule object is configured with either a Weekly_Schedule or an active Exception_Schedule, during a period where Effective_Period is active, with at least two consecutive entries with distinguishable values in the List of BACnetTimeValues, and with no Exception_Schedules at a higher priority. D₁ represents the date and time of the first of these two BACnetTimeValues, with corresponding value V₁, while D₂ and V₂ (a value distinguishable from V₁) represent the second BACnetTimeValue. A time D₁ is defined to occur between D₁ and D₂.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
2. **WAIT Schedule Evaluation Fail Time**
3. VERIFY Present_Value = V₁
4. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
5. **BEFORE Schedule Evaluation Fail Time**
RECEIVE WriteProperty-Request,
 'Object Identifier' = (the referenced object in the TD),
 'Property Identifier' = (the referenced property in the TD),
 'Property Value' = V₂,
 'Priority' = (the value of the Schedule object's Priority_For_Writing property)
6. TRANSMIT BACnet-SimpleACK-PDU
7. **WAIT Schedule Evaluation Fail Time**
8. VERIFY Present_Value = V₂

Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

Note to Tester: The Priority parameter for the WriteProperty-Request may be left out if the Schedule is configured with a value of 16 in its Priority_For_Writing property or if the target property is a standard property of a standard object for which commandability is not an option.

7.3.2.23.9 Exception_Schedule Size Change Test

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

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

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

Test Steps:

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

7.3.2.23.10 Schedule Object Protocol_Revision 4 Tests

The Schedule object was revised in Addendum a to ANSI/ASHRAE 135-2001, which increased Protocol_Revision to 4. Although the basic structure of the Schedule object changed little, its operations are sufficiently different that the existing tests for the original Schedule object need revision in some cases and complete replacement in others, and new tests for some additional changes were needed. This clause presents specific tests to be run for Schedule objects in devices that claim Protocol_Revision 4 or higher.

The Schedule object has no properties required to be writable or otherwise configurable. The following tests are designed to be performed on such a Schedule object. However, if the Schedule object is in any way configurable, then it shall be configured to accommodate as many of the following tests as is possible for the implementation. If it is impossible to configure the IUT in the manner required for a particular test, then that test shall be omitted. If the IUT supports Schedule objects that can write outside the device this shall be demonstrated in one of the Schedule tests.

Tests of the Schedule object center upon observing the write operations scheduled to occur at specific dates and times, verified by reading the Schedule object's Present_Value property. For the test to be performed in a reasonable amount of time it is necessary to be able to alter settings of the device's clock and calendar.

For each test, a date and (as required) time ("Date") for the test is determined beforehand. The tables in the Configuration Requirements section of each test give the criteria for the Date/Time value, designated D1, D2, and so on, to be used in the tests. Date/Time values meeting these criteria may be chosen from existing schedules, or a schedule may be developed by the manufacturer to meet these criteria.

Associated with each Date/Time value, Dn, and defining the time of a schedule write operation is a value Vn, which is the value associated with the time member of Dn in the BACnetTimeValue pair. Vn may take on any primitive datatype.

7.3.2.23.10.1 Revision 4 Effective_Period Test

Purpose: To verify that Effective_Period controls the range of dates during which the Schedule object is active.

Test Concept: Two Date values are chosen by the TD based on the criteria in Table 7-9 such that one is outside of the Effective_Period and the other corresponds to a known scheduled state inside the Effective_Period. The IUT's local date and time are changed between these dates and a property referenced by the List_Of_Object_Property_References property is monitored to verify that write operations occur only within the Effective_Period.

Configuration Requirements: The IUT shall be configured with a schedule object such that the time periods defined in Table 7-9 have uniquely scheduled values. The local date and time shall be set such that the Present_Value property has a value other than V₁. The List_Of_Object_Property_References property shall contain at least one reference either to a property within the IUT alterable by the Schedule object or a writable property in another device (in either case: the "referenced property"); if the List_Of_Object_Property_References property cannot be thus configured, then this test shall be skipped.

Table 7-9. Criteria for Effective_Period Test Dates and Values

Date:	Criteria:	Value:
D ₁	1. Date occurs during Effective_Period, and 2. Date appears either in Weekly_Schedule or Exception_Schedule.	V ₁
D ₂	1. Date does not occur during Effective_Period, and 2. Date appears either in Weekly_Schedule or Exception_Schedule.	V ₂ different from V ₁ .

Test Steps:

1. VERIFY "referenced property" = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
| MAKE (the local date and time = D₁)
3. WAIT Schedule Evaluation Fail Time
4. VERIFY "referenced property" = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
| MAKE (the local date and time = D₂)
6. WAIT Schedule Evaluation Fail Time
7. VERIFY "referenced property" = V₁

7.3.2.23.10.2 Revision 4 Weekly_Schedule Property Test

Purpose: To verify that Weekly_Schedule contains distinguishable schedules for each day of the week and that a day's entire schedule can be executed.

Test Concept: The IUT's local date and time are changed sequentially to represent each day of the week as shown in Table 7-10. The Present_Value property is monitored to verify that write operations occur for each separately scheduled day.

Configuration Requirements: The IUT shall be configured with a schedule object containing a weekly schedule with seven distinguishable daily schedules meeting the requirements of Table 7-10. The local date and time shall be set such that the Present_Value property has a value other than V_1 . If no schedule exists that meets these requirements and none can be configured, then this test shall be omitted. An "active period" is defined as a period of time when the Exception_Schedule determines the value appearing in Present_Value.

Table 7-10. Criteria for Weekly Schedule Test Dates and Values

Date:	Criteria:	Value:
D ₁	1. Date occurs during Effective_Period, 2. Date occurs on a Monday, and 3. Date does not occur during an active period in Exception_Schedule.	V_1
D ₂	1. Date occurs during Effective_Period, 2. Date occurs on a Tuesday, and 3. Date does not occur during an active period in Exception_Schedule.	V_2 is different from V_1 .
D ₃	1. Date occurs during Effective_Period, 2. Date occurs on a Wednesday, and 3. Date does not occur during an active period in Exception_Schedule.	V_3 is different from V_2 .
D ₄	1. Date occurs during Effective_Period, 2. Date occurs on a Thursday, and 3. Date does not occur during an active period in Exception_Schedule.	V_4 is different from V_3 .
D ₅	1. Date occurs during Effective_Period, 2. Date occurs on a Friday, and 3. Date does not occur during an active period in Exception_Schedule.	V_5 is different from V_4 .
D ₆	1. Date occurs during Effective_Period, 2. Date occurs on a Saturday, and 3. Date does not occur during an active period in Exception_Schedule.	V_6 is different from V_5 .
D ₇	1. Date occurs during Effective_Period, 2. Date occurs on a Sunday, and 3. Date does not occur during an active period in Exception_Schedule.	V_7 is different from V_6 .

Test Steps:

1. VERIFY Present_Value = (any value other than V_1)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
 | MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V_1
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V_2
8. (TRANSMIT TimeSynchronization-Request, 'Time' = D₃)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₃)
 | MAKE (the local date and time = D₃)
9. WAIT **Schedule Evaluation Fail Time**
10. VERIFY Present_Value = V_3
11. (TRANSMIT TimeSynchronization-Request, 'Time' = D₄)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₄)
 | MAKE (the local date and time = D₄)
12. WAIT **Schedule Evaluation Fail Time**
13. VERIFY Present_Value = V_4
14. (TRANSMIT TimeSynchronization-Request, 'Time' = D₅)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₅)
 | MAKE (the local date and time = D₅)

15. **WAIT Schedule Evaluation Fail Time**
16. VERIFY Present_Value = V_r
17. (TRANSMIT TimeSynchronization-Request, 'Time' = D_6)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_6)
 | MAKE (the local date and time = D_6)
18. **WAIT Schedule Evaluation Fail Time**
19. VERIFY Present_Value = V_6
20. (TRANSMIT TimeSynchronization-Request, 'Time' = D_7)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_7)
 | MAKE (the local date and time = D_7)
21. **WAIT Schedule Evaluation Fail Time**
22. VERIFY Present_Value = V_7
23. REPEAT X = (the time portion of the BACnetTimeValue entries for one of the daily schedules in Table 7-10) DO {
 (TRANSMIT TimeSynchronization-Request, 'Time' = X)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = X)
 | MAKE (the local date and time = X)
 WAIT Schedule Evaluation Fail Time
 VERIFY Present_Value = (the scheduled value corresponding to time X)
 }

7.3.2.23.10.3 Revision 4 Exception_Schedule Property Tests

If the IUT cannot be made to meet the configuration requirements of one or more of the tests in this clause, then that test shall be omitted. The inability to make such a configuration may be due to an absent or immutable Exception_Schedule property to limited numbers of available BACnetSpecialEvent records in the Exception_Schedule or to the unavailability of Calendar objects.

7.3.2.23.10.3.1 Revision 4 Calendar Reference Test

Purpose: To verify that a date appearing in a referenced Calendar object enables the referencing Schedule object. This test applies to Protocol_Revision 3 and prior, as well as to Protocol_Revision 4 and later schedule objects.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-11. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object that references a Calendar object with a non-empty Date_List. The criteria for the dates used are given in Table 7-11. The local date and time shall be set such that the Present_Value property has a value other than V_1 .

Table 7-11. Criteria for Calendar Reference Dates and Values

Date	Criteria	Value
D_1	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent references Calendar object via calendarReference, 2B. Date appears in that Calendar's Date_List property, 2C. Time is on or after the time of the entry with V_1 , but before any other entry in the Exception_Schedule, and 2D. BACnetSpecialEvent has a higher eventPriority than any other coincident BACnetSpecialEvent records.	V_1
D_2	1. Date occurs during Effective_Period, 2. Date does not appear in any BACnetSpecialEvent records, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	V_2 different from V_1

Test Steps:

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

5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. **VERIFY** Present_Value = V₂

7.3.2.23.10.3.2 Revision 4 Calendar Entry Date Test

Purpose: To verify that a specified date appearing in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-12. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception_Schedule containing a BACnetCalendarEntry with a specific date. The criteria for the dates used in the test are given in Table 7-12. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-12. Criteria for Calendar Entry Date Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: Date, 2B. Date matches calendarEntry: Date, 2C. Time is on or after the time of the entry with V ₁ , but before any other entry in the Exception_Schedule, and 2D. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records.	V ₁
D ₂	1. Date occurs during Effective_Period, 2. Date does not appear in any BACnetSpecialEvent records, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	V ₂ different from V ₁

Test Steps:

1. **VERIFY** Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
 | MAKE (the local date and time = D₁)
3. **WAIT Schedule Evaluation Fail Time**
4. **VERIFY** S, Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. **WAIT Schedule Evaluation Fail Time**
7. **VERIFY** S, Present_Value = V₂

7.3.2.23.10.3.3 Revision 4 Calendar Entry DateRange Test

Purpose: To verify that a date appearing in an Exception_Schedule's date range enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-13. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception_Schedule containing a BACnetCalendarEntry with a date range. The criteria for the dates used in the test are given in Table 7-13. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-13. Criteria for Calendar Entry DateRange Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: DateRange, 2B. Date matches BACnetCalendarEntry: DateRange, 2C. Time is on or after the time of the entry with V ₁ , but before any other entry in the Exception_Schedule, 2D. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records, and 2E. DateRange shall not include wildcards.	V ₁
D ₂	1. Date occurs during Effective_Period, 2. Date does not appear in any BACnetSpecialEvent records, 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object, and 4. DateRange shall not include wildcards.	V ₂ different from V ₁

Test Steps:

1. VERIFY (S), Present_Value = any value other than V₁
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
 | MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY S, Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY S, Present_Value = V₂

7.3.2.23.10.3.4 Revision 4 Calendar Entry WeekNDay Month Test

Purpose: To verify that a date matching a WeekNDay's Month field, specifying a specific month in an Exception_Schedule, enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-14. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying a specific month, from January (1) to December (12). The criteria for the dates used in the test are given in Table 7-14. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-14. Criteria for Calendar Entry WeekNDay Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay: specifies Month, 2C. Date matches calendarEntry: WeekNDay, 2D. Time is on or after the time of the entry with V ₁ , but before any other entry in the Exception_Schedule, and 2E. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records.	V ₁
D ₂	1. Date occurs during Effective_Period, 2. Date does not appear in any BACnetSpecialEvent records, and 3. Date occurs on the same date as, but with time following, an entry in a BACnetDailySchedule in the referencing Schedule object.	V ₂ different from V ₁

Test Steps:

1. VERIFY (S), Present_Value = any value other than V₁
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
 | MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY S, Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY S, Present_Value = V₂

7.3.2.23.10.3.5 Revision 4 Calendar Entry WeekNDay Week Of Month Test

Purpose: To verify that a date matching a WeekNDay's WeekOfMonth field in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-15. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying a week of the month. The criteria for the dates used in the test are given in Table 7-15. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-15. Criteria for Calendar Entry WeekNDay Week Of Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth is in the range 1..5, 2D. Date matches calendarEntry: WeekNDay, 2E. Time is on or after the time of the entry with V ₁ , but before any other entry in the Exception_Schedule, and 2F. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth is in the range 1..5, and 2D. Date does not match calendarEntry: WeekNDay: WeekOfMonth.	V ₂ different from V ₁

Test Steps:

1. VERIFY (S), Present_Value = any value other than V₁
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁)
 | MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY S, Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂)
 | MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY S, Present_Value = V₂

7.3.2.23.10.3.6 Revision 4 Calendar Entry WeekNDay Special Week Of Month Test

Purpose: To verify that a date matching a WeekNDay's WeekOfMonth field in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-16. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object, S, with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying one of the special week of month values, WM [6,7,8,9]. The criteria for the dates used in the test are given in Table 7-16. The local date and time shall be set such that the Present_Value property has a value other than V₁.

Table 7-16. Criteria for Calendar Entry WeekNDay Last Week Of Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value WM, 2D. Date is in the WeekOfMonth specified by 2C, 2E. WeekNDay:Month matches the specified month, 2F. WeekNDay:dayOfWeek matches the specified day of the week, 2G. Time is on or after the time of the entry with V ₁ , but before any other entry in the Exception_Schedule, and 2H. BACnetSpecialEvent has a higher eventPriority than any coincident BACnetSpecialEvent records.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies WeekOfMonth, 2C. calendarEntry: WeekNDay: WeekOfMonth has the value WM, and 2D. Date is not in the WeekOfMonth specified by 2C.	V ₂ different from V ₁

Test Steps:

1. VERIFY (S), Present_Value = any value other than V₁
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY S, Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY S, Present_Value = V₂

7.3.2.23.10.3.7 Revision 4 Calendar Entry WeekNDay Day Of Week Test

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

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-16.1. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying the day of the week. The criteria for the dates used in the test are given in Table 7-16.1. The local date and time shall be set such that the Present_Value property has a value other than V₁.

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

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, 2C. Date falls on the specified day of the week, and 2D. Higher EventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay specifies only DayOfWeek, and 2C. Date does not fall on the specified day of the week.	V ₂ (a value different from the Present_Value expected at D ₂)

Test Steps:

1. VERIFY Present_Value = (any value other than V₁)
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than V₂)

7.3.2.23.10.3.8 Revision 4 Event Priority Test

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

Configuration Requirements: The IUT shall be configured with a Schedule object containing two or more BACnetSpecialEvents, all active on the same date, with different EventPriority values (if possible, all 16 priority levels should be represented), and with overlapping BACnetTimeValue entries distributed thus: the entry with the lowest priority shall have the earliest time-value pair (D₁) with a non-NULL value, and the last time-value pair (D_N) with a NULL value; the next higher priority shall have a time-value pair D₂ occurring after D₁ with a different non-NULL value, and a time-value pair D_{N-1} with a NULL value and occurring before D_N; and so on. The result is that the time-value pairs shall be ordered chronologically thus: D₁, D₂, D₃, ..., D_{N-1}, D_N. An example of such a configuration testing five priority levels is shown in Table 7-16.2.

Table 7-16.2. Example of event and value prioritization

Event Priority:	Time:								
	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	D ₈	D ₉
1	-	-	-	-	V ₅	NULL	-	-	-
2	-	-	-	V ₄	-	-	NULL	-	-
3	-	-	V ₃	-	-	-	-	NULL	-
4	-	V ₂	-	-	-	-	-	-	NULL
5	V ₁	-	-	-	-	-	-	-	-
Present Value:	V ₁	V ₂	V ₃	V ₄	V ₅	V ₄	V ₃	V ₂	V ₁

Note: Each event priority in the table above represents one BACnetSpecialEvent. The BACnetSpecialEvent shall contain the time value pairs listed in the table (D_x, V_x). There shall be only one BACnetSpecialEvent per priority for this test.

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

Test Steps:

1. REPEAT D = (the times in the configured time-value pairs with non-NULL values) DO {

- ```

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

```

### 7.3.2.23.10.3.9 Revision 4 List of BACnetTimeValue Test

Purpose: To verify that a BACnetSpecialEvent's entire schedule can be executed.

Test Concept: A special event is scheduled that contains multiple BACnetTimeValue entries with distinguishable non-NULL values. The local date and time are changed to values that match each of the BACnetTimeValue entries, and the Present\_Value property is read to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured with a Schedule object containing a BACnetSpecialEvent with two or more BACnetTimeValue entries. Each BACnetTimeValue entry shall be non-NULL and have a distinguishable value. The BACnetSpecialEvent selected for this test shall be the highest priority event active for the day selected for testing.

Test Steps:

- ```

1. REPEAT Di = (the times used in the BACnetTimeValue pairs of the special event) DO {
    (TRANSMIT TimeSynchronization-Request, 'Time' = Di)
    | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = Di)
    | MAKE (the local date and time = Di)
    WAIT Schedule Evaluation Fail Time
    VERIFY Present_Value = (the value corresponding to the special event)
  }

```

7.3.2.23.10.3.10 Revision 4 Calendar Entry WeekNDay Odd-Numbered Month Test

Purpose: To verify that a date matching a WeekNDay's Month field, specifying odd-numbered months (BACnetWeekNDay month enumeration value 13), in an Exception_Schedule enables the referencing Schedule object.

Test Concept: The IUT's local date and time are changed such that all months of the year are tested. The value of the Present_Value property is monitored to verify that the scheduled write operations occur.

Configuration Requirements: The IUT shall be configured to contain a Schedule object with an Exception_Schedule containing a BACnetCalendarEntry with a WeekNDay entry specifying odd-numbered months in a specific year and a second, lower priority, BACnetCalendarEntry with a WeekNDay entry specifying all days of the year specified in the first. The criteria for the dates used in the test are given in Table 7-16.3.

Table 7-16.3. Criteria for Calendar Entry WeekNDay Month Test Dates and Values

Date	Criteria	Value
D ₁	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay: specifies odd-numbered months with a specific year (Y), and 2C. Higher eventPriority than any coincident BACnetSpecialEvents.	V ₁
D ₂	1. Date occurs during Effective_Period, 2A. BACnetSpecialEvent incorporates calendarEntry: WeekNDay, 2B. calendarEntry: WeekNDay:Month specifies all days of the year used in D ₁ , and 2C. Lower eventPriority than that used for D ₁ .	V ₂ (different from V ₁)

Test Steps:

1. REPEAT X = (All months, 1-12) {
 (TRANSMIT TimeSynchronization-Request, 'Time' = D, where month = X and year is Y)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D, where month = X and year is Y)
 | MAKE (the local date and time = D, where month = X and year is Y)
 WAIT **Schedule Evaluation Fail Time**
 IF (X is odd) THEN
 VERIFY Present_Value = V₁
 ELSE
 VERIFY Present_Value = V₂
 }

7.3.2.23.10.3.11 Revision 4 Calendar Entry WeekNDay Even-Numbered Month Test

This test is identical to 7.3.2.23.10.3.10, except that even-numbered months (BACnetWeekNDay month enumeration value 14) are used instead of odd-numbered months.

7.3.2.23.10.3.12 Revision 4 Lower Event Priority Change Test

Purpose: To verify that when a BACnetSpecialEvent of a higher priority takes precedence over one of lower priority, that a change in the lower priority level is not observed in Present_Value until control is relinquished to it.

Configuration Requirements: A Schedule object is configured with two BACnetSpecial Events, thus: the first event is at lower priority than the second and contains two time-value pairs: the first, D₁, has a non-NULL value V₁ and the second, D₄, has a non-NULL value V₄ which is different from V₁ and different from V₃. The second event contains three time-value pairs: the first, D₂, occurs after D₁ and before D₃ and has a non-NULL value V₂ different from the value V₁; the second, D₃, occurs after D₂ and has a non-NULL value V₃ different from the value V₂; the third, D₅, occurs after D₄ and has a NULL value. (This arrangement of events facilitates testing Schedule objects that schedule only BOOLEAN or two-state enumerations.) Table 7-16.4 illustrates the time and value pairs in this test.

Table 7-16.4. Event and value prioritization test times and value

Event Priority:	Time:				
	D ₁	D ₂	D ₃	D ₄	D ₅
Higher	-	V ₂	V ₃	-	NULL
Lower	V ₁	-	-	V ₄	-
Present Value:	V ₁	V ₂	V ₃	V ₃	V ₄

Note: Each event priority in the table above represents one BACnetSpecialEvent. The BACnetSpecialEvent shall contain the time value pairs listed in the table (D_x, V_x). There shall be only one BACnetSpecialEvent per priority for this test.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
 (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
 MAKE (the local date and time = D₁)
2. VERIFY Present_Value = V₁
3. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
 (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |

- MAKE (the local date and time = D₂)
4. VERIFY Present_Value = V₂
 5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₃) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₃) |
MAKE (the local date and time = D₃)
 6. VERIFY Present_Value = V₃
 7. (TRANSMIT TimeSynchronization-Request, 'Time' = D₄) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₄) |
MAKE (the local date and time = D₄)
 8. VERIFY Present_Value = V₃
 9. (TRANSMIT TimeSynchronization-Request, 'Time' = D₅) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₅) |
MAKE (the local date and time = D₅)
 10. VERIFY Present_Value = V₄

7.3.2.23.10.3.13 Revision 4 Schedule_Default Test

Purpose: To verify that the value in Schedule_Default is applied when no weekly or exception schedule is in effect.

Configuration Requirements: The IUT shall be configured with a Schedule object with a Schedule_Default value V_{default} and containing at least one of, and if possible, both (non-overlapping):

- a Weekly_Schedule containing a time-value pair at time D₁ with a non-NULL value V₁ different from V_{default} and a subsequent time-value pair with a NULL value at time D₂.
- an Exception_Schedule with no overlap with the time frame D₁ to D₂, a time-value pair at time D₃ with a non-NULL value V₃ different from V_{default}, and a subsequent time-value pair with a NULL value at time D₄.

Test Steps:

1. IF (the Schedule object is configured with a Weekly_Schedule) THEN
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D₁)
3. WAIT Schedule Evaluation Fail Time
4. VERIFY Present_Value = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
6. WAIT Schedule Evaluation Fail Time
7. VERIFY Present_Value = V_{default}
8. IF (the Schedule object is configured with an Exception_Schedule) THEN
9. (TRANSMIT TimeSynchronization-Request, 'Time' = D₃) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₃) |
MAKE (the local date and time = D₃)
10. WAIT Schedule Evaluation Fail Time
11. VERIFY Present_Value = V₃
12. (TRANSMIT TimeSynchronization-Request, 'Time' = D₄) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₄) |
MAKE (the local date and time = D₄)
13. WAIT Schedule Evaluation Fail Time
14. VERIFY Present_Value = V_{default}

7.3.2.23.10.4 Revision 4 Weekly_Schedule and Exception_Schedule Interaction Test

Purpose: To verify that an Exception_Schedule takes precedence over a coincident Weekly_Schedule; to verify that Weekly_Schedule automatically takes control once it becomes active and after the Exception_Schedule is expired; and to verify that the value in Schedule_Default is applied when no weekly or exception schedule is in effect.

Test Concept: The IUT is configured with a Weekly_Schedule and an Exception_Schedule that apply to the same date. The local time is changed to the time when the Exception_Schedule takes control, and the Present_Value is read to verify that the scheduled write operation occurs. The local time is changed again to a value that would cause another change if the Weekly_Schedule were in control. The Present_Value is read to verify the Exception_Schedule is still in control. The local time is changed again to a value where Exception_Schedule is not in effect, and the Present_Value is read to verify

that Weekly_Schedule is in control. The local time is changed again to a value where both Exception_Schedule and Weekly_Schedule are not in effect, and the Present_Value is read to verify that Schedule_Default is written into it.

Configuration Requirements: The IUT shall be configured with a Schedule object containing a Weekly_Schedule and an Exception_Schedule that both apply to a particular day, D_1 , within the Effective_Period. There shall be no other BACnetSpecial Events in the Exception_Schedule. The Weekly_Schedule for the day of week entry related to D_1 is configured with the time value pairs (T_2, V_2) and (T_4, NULL) , and the Exception_Schedule is configured with the time value pairs (T_1, V_1) and (T_3, NULL) where $T_1 < T_2 < T_3 < T_4$. The values V_1 and V_2 shall be different, and if possible, different from V_{default} , the Schedule_Default. If possible, a non-NULL Schedule_Default is used.

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = T_1 on day D_1)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = T_1 on day D_1)
| MAKE (the local time = T_1 on day D_1)
2. WAIT **Schedule Evaluation Fail Time**
3. VERIFY Present_Value = V_1
4. (TRANSMIT TimeSynchronization-Request, 'Time' = T_2 on day D_1)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = T_2 on day D_1)
| MAKE (the local time = T_2 on day D_1)
5. WAIT **Schedule Evaluation Fail Time**
6. VERIFY Present_Value = V_1
7. (TRANSMIT TimeSynchronization-Request, 'Time' = T_3 on day D_1)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = T_3 on day D_1)
| MAKE (the local time = T_3 on day D_1)
8. WAIT **Schedule Evaluation Fail Time**
9. VERIFY Present_Value = V_2
10. (TRANSMIT TimeSynchronization-Request, 'Time' = T_4 on day D_1)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = T_4 on day D_1)
| MAKE (the local time = T_4 on day D_1)
11. WAIT **Schedule Evaluation Fail Time**
12. VERIFY Present_Value = V_{default}

7.3.2.23.10.5 Revision 4 Exception_Schedule Restoration Test

Purpose: To verify the restoration behavior in an Exception_Schedule.

Test Steps: The Revision 4 version of this test is identical to the test in Clause 7.3.2.23.5 Exception_Schedule Restoration Test.

7.3.2.23.10.6 Revision 4 Weekly_Schedule Restoration Test

Purpose: To verify the restoration behavior in an Exception_Schedule.

Test Steps: The Revision 4 version of this test is identical to the test in Clause 7.3.2.23.6.

7.3.2.23.10.7 Revision 4 List_Of_Object_Property_Reference Internal Test

Purpose: To verify that the Schedule object writes to objects and properties contained within the IUT.

Test Steps: The Revision 4 version of this test is identical to the test in Clause 7.3.2.23.7.

7.3.2.23.10.8 Revision 4 List_Of_Object_Property_Reference External Test

Purpose: To verify that the Schedule object writes to object properties contained in a device other than the IUT.

Test Steps: The Revision 4 version of this test is identical to the test in Clause 7.3.2.23.8.

7.3.2.23.11 Written Datatypes Tests

The following tests verify that the Schedule properly writes datatypes that it claims to support.

7.3.2.23.11.1 Internally Written Datatypes Test, non-NULL values

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

Test Concept: Two Date/Time, values, D_1 and D_2 , are chosen by the TD based on the criteria in Table 7-17 such that D_1 is sufficiently different from, and later than, the current time to cause a Schedule evaluation when the time is changed to D_1 , and such that setting the time to D_2 (later than D_1) from D_1 will cause a Schedule evaluation that will cause it to write value V_2 . These values may be chosen based on the Schedule object's existing configuration, or the Schedule object, S, may be configured with such values.

Configuration Requirements: The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-17 can be configured with uniquely scheduled values. The Schedule object shall be configured with a List_Of_Object_Property_References, including at least one reference to a writable property within the device, if possible. Step 4 and Step 8 would REPEAT zero times if the referenced property is empty or not present. Properties in the Schedule object shall be consistent in both datatypes and values in a manner permitting this test to be executed.

Table 7-17. Criteria for Test Date and Times

Date and Time:	Value:
D_1	V_1
D_2	V_2 different from V_1 .

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

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D_1)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'= D_1)
| MAKE (the local date and time = D_1)
2. WAIT(Schedule Evaluation Fail Time)
3. VERIFY S, Present_Value = V_1
4. REPEAT P = (writable property in List_Of_Property_References)
VERIFY P = V_1
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D_2)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'= D_2)
| MAKE (the local date and time = D_2)
6. WAIT(Schedule Evaluation Fail Time)
7. VERIFY S, Present_Value = V_2
8. REPEAT P = (writable property in List_Of_Property_References)
VERIFY P = V_2

7.3.2.23.11.2 Internally Written Datatypes Test, NULL Values and Priority_Arrays

Purpose: This test verifies that the Schedule object writes NULLs to priority arrays (via Present_Value) within the same device.

Test Concept: Two Date/Time values, D_1 and D_2 , are chosen by the TD based on the criteria in Table 7-18 such that D_1 is sufficiently different from current time to cause a Schedule evaluation when the time is changed to D_1 , and such that setting the time to D_2 from D_1 will cause a Schedule evaluation that will cause it to write value V_2 . These values may be chosen based on the Schedule object's existing configuration, or the Schedule object, S, may be configured with such values, and either V_1 or V_2 , but not both, has datatype NULL. The values are written to a Present_Value property with the priority designated by the Schedule object's Priority_For_Writing property, X. For devices of protocol revision 4 or higher, the Schedule_Default shall be set to NULL and the schedule shall be configured such that at time D_1 or D_2 , but not both, the schedule shall take on the value of Schedule_Default.

Configuration Requirements: The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-18 can be configured with uniquely scheduled values. The Schedule object shall be configured with a List_Of_Object_Property_References, including at least one reference within the device to a Present_Value property, P, in an object containing a Priority_Array property, PA. If the IUT cannot be configured to these requirements, then this test shall be omitted.

Table 7-18. Criteria for Test Date and Times

Date and Time:	Value:
D ₁	V ₁
D ₂	V ₂ different from V ₁ .

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D1)
| MAKE (the local date and time = D1)
2. WAIT(Schedule Evaluation Fail Time)
3. VERIFY S, Present_Value = V₁
4. VERIFY PA[X] = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D2)
| MAKE (the local date and time = D2)
6. WAIT(Schedule Evaluation Fail Time)
7. VERIFY S, Present_Value = V₂
8. VERIFY PA[X] = V₂

7.3.2.23.11.3 Externally Written Datatypes Test, non-NULL values

Purpose: This test verifies that the Schedule object writes to properties in other devices with all datatypes required and claimed for the external write operation. If the IUT supports Schedule objects that have differences in supported datatypes, then this test should be performed on at least one example of each type.

Test Concept: Two Date/Time values, D₁ and D₂, are chosen by the TD based on the criteria in Table 7-19 such that D₁ is sufficiently different from, and later than, the current time to cause a Schedule evaluation when the time is changed to D₁, and such that setting the time to D₂ (later than D₁) from D₁ will cause a Schedule evaluation that will cause it to write value V₂. These values may be chosen based on the Schedule object's existing configuration, or the Schedule object may be configured with such values.

Configuration Requirements: The TD shall be configured to support the WriteProperty-Request service but not WritePropertyMultiple-Request in the Protocol Services Supported property of its Device object. The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-19 can be configured with uniquely scheduled values. The Schedule object shall be configured with a List_Of_Object_Property_Reference property, including at least one reference to a property in the TD. Other properties shall be consistent in both datatypes and values in a manner permitting this test to be executed.

Table 7-19. Criteria for Test Date and Times

Date and Time:	Value:
D ₁	V ₁
D ₂	V ₂ different from V ₁ .

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D1)
| MAKE (the local date and time = D1)
2. BEFORE Schedule Evaluation Fail Time
REPEAT X = (every reference to the TD in List_Of_Object_Property_References) DO {
 RECEIVE WriteProperty-Request,
 'Object Identifier' = (the object identifier of X),
 'Property Identifier' = (the property of X),
 'Property Value' = V₁
 'Priority' = (the value of the Schedule object's Priority_For_Writing property)
 TRANSMIT BACnet-SimpleACK-PDU
}
3. VERIFY S, Present_Value = V₁
4. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D2)

- | MAKE (the local date and time = D2)
5. **BEFORE Schedule Evaluation Fail Time**
 REPEAT X = (every reference to the TD in List_Of_Object_Property_References) DO {
 RECEIVE WriteProperty-Request,
 'Object Identifier' = (the object identifier of X),
 'Property Identifier' = (the property of X),
 'Property Value' = V₂,
 'Priority' = (the value of the Schedule object's Priority_For_Writing property)
 TRANSMIT BACnet-SimpleACK-PDU
 }
 6. **VERIFY** S, Present_Value = V₂

Note to Tester: The Priority parameter for the WriteProperty-Request may be left out if the Schedule is configured with a value of 16 in its Priority_For_Writing property or if the target property is a standard property of a standard object for which commandability is not an option. The test shall pass regardless of the order in which the IUT generates the WriteProperty-Requests in steps 2 and 5.

7.3.2.23.11.4 Externally Written Datatypes Test, NULL values and Priority_Arrays

Purpose: This test verifies that the Schedule object writes NULLs to priority arrays (via Present_Value) in other devices.

Test Concept: Two Date/Time values, D₁ and D₂, are chosen by the TD based on the criteria in Table 7-20 such that D₁ is sufficiently different from current time to cause a Schedule evaluation when the time is changed to D₁, and such that setting the time to D₂ from D₁ will cause a Schedule evaluation that will cause it to write value V₂. These values may be chosen based on the Schedule object's existing configuration, or the Schedule object may be configured with such values, and either V₁ or V₂, but not both, has datatype NULL. The values are written to a Present_Value property with the priority designated by the Schedule object's Priority_For_Writing property. For devices of protocol revision 4 or higher, the Schedule_Default shall be set to NULL and the schedule shall be configured such that at time D₁ or D₂, but not both, the schedule shall take on the value of Schedule_Default.

Configuration Requirements: The TD shall be configured to support the WriteProperty-Request service but not WritePropertyMultiple-Request in the Protocol_Services Supported property of its Device object. The IUT shall be configured with a Schedule object, S, such that the time periods defined in Table 7-20 can be configured with uniquely scheduled values. The Schedule object shall be configured with a Priority_For_Writing value other than 16, and with a List_Of_Object_Property_References, including at least one reference to a Present_Value property in an object in the TD containing a Priority_Array property.

Table 7-20. Criteria for Test Date and Times

Date and Time:	Value:
D ₁	V ₁
D ₂	V ₂ different from V ₁ .

Test Steps:

- (TRANSMIT TimeSynchronization-Request, 'Time' = D₁)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D₁)
 | MAKE (the local date and time = D₁)
- BEFORE Schedule Evaluation Fail Time**
 REPEAT X = (every reference to the TD in List_Of_Object_Property_References) DO {
 RECEIVE WriteProperty-Request,
 'Object Identifier' = (the object identifier of X),
 'Property Identifier' = (the property of X),
 'Property Value' = V₁,
 'Priority' = (the value of the Schedule object's Priority_For_Writing property)
 TRANSMIT BACnet-SimpleACK-PDU
 }
 3. **VERIFY** S, Present_Value = V₁
- (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D₂)
 | MAKE (the local date and time = D₂)
- BEFORE Schedule Evaluation Fail Time**

```

REPEAT X = (every reference to the TD in List_Of_Object_Property_References) DO {
  RECEIVE WriteProperty-Request,
    'Object Identifier' = (the object identifier of X),
    'Property Identifier' = (the property of X),
    'Property Value' = V2,
    'Priority' = (the value of the Schedule object's Priority_For_Writing property)
  TRANSMIT BACnet-SimpleACK-PDU
}

```

6. VERIFY S, Present_Value = V₂

Note to Tester: The Priority parameter for the WriteProperty-Request may be left out if the Schedule is configured with a value of 16 in its Priority_For_Writing property or if the target property is a standard property of a standard object for which commandability is not an option. The test shall pass regardless of the order in which the IUT generates the WriteProperty-Requests in steps 2 and 5.

7.3.2.23.12 Revision 4 Midnight Evaluation Test

Purpose: To verify that the Schedule object evaluates its schedule as it passes through midnight (00:00).

Configuration Requirements: The IUT shall be configured with a Schedule object with a Schedule_Default value V_{default}, and containing at least one of, and if possible, both (non-overlapping):

- a Weekly_Schedule containing a time-value pair at time D₁ (not 00:00) with a non-NULL value V₁ different from V_{default}, and no scheduled write operations during the day after D₁, and none on the day following.
- an Exception_Schedule with an event occurring on a day different from D₁, containing a time-value pair at time D₃ with a non-NULL value V₃ different from V_{default}, and no scheduled write operations during the day after D₃, and none on the day following.

Two additional times, used in the execution of the test, are defined as follows:

- D₂ occurring on the same day as D₁, after D₁, and before midnight.
- D₄ occurring on the same day as D₃, after D₃, and before midnight.

It is recommended that to minimize testing time, D₁ through D₄ be chosen to be close to midnight. However, all times used in this test shall be separated by at least **Schedule Evaluation Fail Time**.

An illustration of the test times and values configured and observed is shown in **Table 7-21**.

Table 7-21. Test Times and Values

Time:	D ₁	D ₂	00:00	D ₃	D ₄	00:00
Exception_Schedule:	-	-	-	V ₃	-	-
Weekly_Schedule:	V ₁	-	-	-	-	-
Present_Value:		V ₁	V _{default}		V ₃	V _{default}

Test Steps:

1. IF (the Schedule object is configured with a Weekly_Schedule) THEN
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₂) |
MAKE (the local date and time = D₂)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present_Value = V₁
5. WAIT (until 00:00)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V_{default}
8. IF (the Schedule object is configured with an Exception_Schedule) THEN
9. (TRANSMIT TimeSynchronization-Request, 'Time' = D₄) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₄) |
MAKE (the local date and time = D₄)
10. WAIT **Schedule Evaluation Fail Time**
11. VERIFY Present_Value = V₃

12. WAIT (until 00:00)
13. WAIT **Schedule Evaluation Fail Time**
14. VERIFY Present_Value = V_{default}

7.3.2.23.13 Forbid Duplicate Time Values

Purpose: To verify that the IUT does not accept duplicate BACnetTimeValue entries.

Test Concept: Transmit a request in which the property value contains duplicate BACnetTimeValue entries.

Configuration Requirements: Select an object, Sched1, which contains a writable Weekly_Schedule or Exception_Schedule property, P1. If the IUT claims a Protocol_Revision less than 16, this test shall be skipped.

Test Steps:

1. READ V1 = (Sched1), P1
2. TRANSMIT WriteProperty-Request,
 'Object Identifier' = Sched1,
 'Property Identifier' = P1,
 'Property Value' = (a value that contains a list of BACnetTimeValue with two entries at the same time within a BACnetDailySchedule or BACnetSpecialEvent)
3. RECEIVE BACnet-Error-PDU,
 Error Class = PROPERTY,
 Error Code = DUPLICATE_ENTRY
4. VERIFY P1 = V1

7.3.2.24 Logging Object Tests

Logging objects have only a few properties required to be writable or otherwise configurable. Logging objects shall be configured to accommodate as many of the following tests as is possible for the implementation. If it is impossible to configure the IUT in the manner required for a particular test that test shall be omitted.

Tests of logging objects center upon the collection of records in the Log_Buffer and the issuance of notifications when a predetermined number of records have been collected since startup or the last preceding notification.

7.3.2.24.1 Enable Test

Purpose: To verify that the Enable property enables and disables the logging of data by the logging object.

Test Concept: The logging object is configured to acquire data by each means available to the implementation. Enable is enabled and the collection of one or more records in the Log_Buffer is confirmed. Enable is then disabled and non-collection of records is confirmed.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with a time that will occur after the completion of the test. Stop_When_Full, if configurable, shall be set to FALSE.

Test Steps:

1. READ I = Log_Interval
2. WRITE Enable = FALSE
3. WRITE Record_Count = 0
4. WAIT **Internal Processing Fail Time**
5. WRITE Enable = TRUE
6. READ X = Total_Record_Count
7. MAKE (IUT collect another record)
8. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
9. VERIFY Total_Record_Count > X
10. WRITE Enable = FALSE
11. READ Y = Total_Record_Count
12. MAKE (IUT collect another record)
13. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
14. VERIFY Total_Record_Count = Y

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

```

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

```

7.3.2.24.2 Start_Time Test

Test Concept: The logging object is configured to acquire data by each means available to the implementation. The test is begun at some time prior to the time specified in Start_Time and non-collection of records is confirmed. Collection of records after the time specified by Start_Time is then confirmed.

Configuration Requirements: Start_Time shall be configured with a date and time such that steps 1 through 6 will be concluded before that time. Stop_Time, if present, shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE; Enable shall be set to TRUE.

Test Steps:

1. WRITE Record_Count = 0
2. WAIT **Internal Processing Fail Time**
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Total_Record_Count
4. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Total_Record_Count
 'Property Value' = (any valid value, X)
5. MAKE (IUT collect another record)
6. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
7. VERIFY Total_Record_Count = (value X returned in step 4)
8. WHILE (IUT clock is earlier than Start_Time) DO {
 VERIFY Total_Record_Count = (value X returned in step 4)
 }
9. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
10. MAKE (IUT collect another record)
11. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
12. VERIFY Total_Record_Count > (value X returned in step 4)

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

```

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

```

7.3.2.24.3 Stop_Time Test

Purpose: To verify that logging is disabled at the time specified by Stop_Time.

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

Configuration Requirements: Stop_Time shall be configured with a date and time such that steps 1 through 9 will be concluded before that time. Start_Time, if present shall be configured with date and time preceding the initiation of the test. Stop_When_Full, if configurable, shall be set to FALSE.

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

```

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

Test Steps:

1. WRITE Enable = FALSE
2. WAIT **Internal Processing Fail Time**
3. WRITE Record_Count = 0
4. WRITE Enable = TRUE
5. READ X = Total_Record_Count
6. WAIT **Internal Processing Fail Time**
7. MAKE (IUT collect another record)
8. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
9. VERIFY Total_Record_Count > X
10. WHILE (IUT clock is earlier than Stop_Time) DO {}
11. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
12. READ X = Total_Record_Count
13. MAKE (IUT collect another record)
14. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)
15. VERIFY Total_Record_Count = X

7.3.2.24.4 Log_Interval Test

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

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

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

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

Test Steps:

1. WRITE Log_Interval = (LI1, some non-zero value)
2. WRITE Record_Count = 0
3. WAIT (**Internal Processing Fail Time** + 10* Log_Interval hundredths-seconds)
4. VERIFY (Log_Buffer record timestamp intervals are LI1)

5. WRITE Log_Interval = (LI2, a non-zero value different from LI1)
6. WRITE Record_Count = 0
7. WAIT (**Internal Processing Fail Time** + 10* Log_Interval hundredths-seconds)
8. VERIFY (Log_Buffer record timestamp intervals are LI2)

7.3.2.24.5 COV_Resubscription_Interval Test

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

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

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE. Enable shall be set to TRUE. Non-zero values shall be chosen for COV_Resubscription_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

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

7. TRANSMIT ConfirmedCOVNotification-Request,
 - 'Subscriber Process Identifier' = (corresponding value in step 1),
 - 'Initiating Device Identifier' = (Device object identifier of the TD),
 - 'Monitored Object Identifier' = (corresponding value in step 1),
 - 'Issue Confirmed Notifications' = (corresponding value in step 1),
 - 'Time Remaining' = (any value $\leq$ the Lifetime from step 5),
 - 'List of Values' = (appropriate BACnetPropertyValue(s))
8. RECEIVE BACnet-SimpleACK-PDU
9. WAIT (COV_Resubscription_Interval - **Re-subscription Interval Tolerance**)
10. BEFORE (2 * **Re-subscription Interval Tolerance**)
 - IF (the IUT uses SubscribeCOV)
 - RECEIVE SubscribeCOV-Request,
 - 'Subscriber Process Identifier' = (corresponding value in step 1),
 - 'Monitored Object Identifier' = (corresponding value in step 1),
 - 'Issue Confirmed Notifications' = (TRUE),
 - 'Lifetime' = (corresponding value in step 1)
 - ELSE
 - RECEIVE SubscribeCOVProperty-Request,
 - 'Subscriber Process Identifier' = (corresponding value in step 1),
 - 'Monitored Object Identifier' = (corresponding value in step 1),
 - 'Issue Confirmed Notifications' = (TRUE),
 - 'Lifetime' = (corresponding value in step 1),
 - 'Monitored Property Identifier' = (corresponding value in step 1),
 - 'COV Increment' = (corresponding value in step 1)
 - 11. TRANSMIT BACnet-SimpleACK-PDU

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

7.3.2.24.6 Stop_When_Full Tests

Two tests are performed on Stop_When_Full. The first is performed only when Stop_When_Full can be configured to TRUE, the second when Stop_When_Full can be configured to FALSE.

7.3.2.24.6.1 Stop_When_Full TRUE Test

Purpose: To verify that Stop_When_Full set to TRUE properly indicates that the logging object ceases collecting data when its Log_Buffer acquires Buffer_Size data items.

Test Concept: The logging object is configured to acquire data by whatever means. Data is collected until more than Buffer_Size records have been collected and Enable is verified to be FALSE. Attempt to write TRUE to Enable and verify that the IUT does not accept it due to Log_Buffer being full.

Configuration Requirements: The IUT shall be configured with Object1 where Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to TRUE. Enable shall be set to FALSE.

Test Steps:

1. WRITE Record_Count = 0
2. WRITE Enable = TRUE
3. WHILE (Record_Count < Buffer_Size) DO { }
4. WAIT **Internal Processing Fail Time**
5. VERIFY Enable = FALSE
6. TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = Object1,
 - 'Property Identifier' = Enable,
 - 'Property Value' = TRUE

7. RECEIVE BACnet-Error-PDU,
 'Error Class' = OBJECT,
 'Error Code' = LOG_BUFFER_FULL.
8. VERIFY Enable = FALSE

7.3.2.24.6.2 Stop_When_Full FALSE Test

Purpose: To verify that Stop_When_Full set to FALSE properly indicates that the logging object continues collecting data after its Log_Buffer acquires Buffer_Size data items.

Test Concept: The logging object is configured to acquire data by whatever means. Data is collected until more than Buffer_Size records have been collected and Enable is verified to be TRUE.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE. Enable shall be set to FALSE.

Test Steps:

1. WRITE Record_Count = 0
2. WRITE Enable = TRUE
3. WHILE (Record_Count < Buffer_Size) DO { }
4. WAIT **Internal Processing Fail Time**
5. VERIFY Enable = TRUE

7.3.2.24.7 Buffer_Size Test

Purpose: To verify that Buffer_Size properly indicates the number of records that can be stored in the Log_Buffer.

Test Concept: The logging object is configured to acquire data by whatever means. Data is collected until at least Buffer_Size records have been collected, then the Log_Buffer is read and the presence of Buffer_Size discrete records is verified.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occurs after the end of the test. Enable shall be set to TRUE.

Test Steps:

1. WHILE (Record_Count < Buffer_Size) DO { }
2. WRITE Enable = FALSE
3. WAIT **Internal Processing Fail Time**
4. CHECK (that Log_Buffer has Buffer_Size discrete records)

7.3.2.24.8 Record_Count Test

Test Concept: The logging object is configured to acquire data by whatever means. Record_Count is set to zero and Log_Buffer is read to verify that only one record is present and that it is the buffer-purged event. Collection of data proceeds until Record_Count is about Buffer_Size/2, collection is halted and Log_Buffer is read to verify the Record_Count value. Collection then resumes until Buffer_Size records are read; collection is then halted and Log_Buffer read to verify the Record_Count again.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occurs after the end of the test. Enable shall be set to FALSE.

Test Steps:

1. WRITE Record_Count = 0
2. WAIT **Internal Processing Fail Time**
3. CHECK (Log_Buffer contains one entry, and it is the buffer-purged event)
4. WRITE Enable = TRUE
5. WHILE (Record_Count < Buffer_Size/2) DO { }

6. WRITE Enable = FALSE
7. WAIT **Internal Processing Fail Time**
8. CHECK (that Log_Buffer has the number of records indicated by Record_Count)
9. WRITE Enable = TRUE
10. WHILE (Record_Count < Buffer_Size) DO { }
11. WRITE Enable = FALSE
12. WAIT **Internal Processing Fail Time**
13. CHECK (that Log_Buffer has the number of records indicated by Record_Count)

7.3.2.24.9 Total_Record_Count Test

Purpose: To verify that the Total_Record_Count property increments for each record added to the Log_Buffer, even after Buffer_Size records have been added. (Note: it is not reasonable to test for the requirement of BACnet Clause 12.25.16 that the value wrap from $2^{32}-1$ to one; even if a record was collected every 100th of a second it could take more than 497 days to complete the test.)

Test Concept: The logging object is configured to acquire data by whatever means. Total_Record_Count is read to determine an initial value. Record_Count is set to zero and Total_Record_Count is read. It is verified that Total_Record_Count is incremented and not reset to 0 when Record_Count is written to 0. Collection of data proceeds until Record_Count changes, collection is halted and Total_Record_Count is checked that it has incremented by Record_Count. If, for whatever reason, the IUT cannot be configured such that the TD is able to halt collection before Buffer_Size records are collected this test shall not be performed.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Enable shall be set to FALSE.

Test Steps:

1. READ TRC1 = Total_Record_Count
2. WRITE Record_Count = 0
3. WAIT **Internal Processing Fail Time**
4. VERIFY Total_Record_Count = TRC1 + 1
5. READ X = Total_Record_Count
6. READ Y = Record_Count
7. WRITE Enable = TRUE
8. WHILE (Record_Count = Y + 1) DO { }
9. WRITE Enable = FALSE
10. WAIT **Internal Processing Fail Time**
11. IF (Total_Record_Count - X != Record_Count - Y) THEN
ERROR "Total_Record_Count has incorrect value."

7.3.2.24.10 Notification_Threshold Test

Purpose: To verify that the Notification_Threshold property reflects the number of records collected since a previous notification, or since logging started, that causes a Buffer_Ready notification to be sent.

Test Concept: The logging object is configured to acquire data by whatever means. Record_Count is set to zero. Collection of data proceeds until a notification is seen, and the value of Record_Count is checked. Collection continues until the second notification, when Record_Count is verified again.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Enable shall be set to FALSE. Notification_Threshold shall be set to a non-zero value.

Test Steps:

1. WRITE Record_Count = 0
2. WAIT **Internal Processing Fail Time**
3. READ X = Total_Record_Count
4. WRITE Enable = TRUE
5. MAKE (LO1 collect the number of records specified by pThreshold)

6. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = LO1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the configured notification class),
 - 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 - 'Event Type' = BUFFER_READY,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = EVENT | ALARM,
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = NORMAL,
 - 'Event Values' = (pLogBuffer),
 - (pPreviousCount: valid value < X),
 - (pMonitoredValue: any value Y_1 where $Y_1 \geq X$ and $Y_1 \leq X + pThreshold$)
7. TRANSMIT BACnet-SimpleACK-PDU
8. VERIFY Total_Record_Count $\geq Y_1$
9. MAKE (the logging object collect number of records specified by pThreshold)
10. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = (LO1),
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (the configured notification class),
 - 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 - 'Event Type' = BUFFER_READY,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = EVENT | ALARM,
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = NORMAL,
 - 'To State' = NORMAL,
 - 'Event Values' = (pLogBuffer),
 - (pPreviousCount: Y_1),
 - (pMonitoredValue: $Y_1 + pThreshold$)
11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY Total_Record_Count $\geq Y_1 + pThreshold$
13. WRITE Enable = FALSE

7.3.2.24.11 Notification Time Tests

Purpose: To verify that the Previous_Notify_Time and Current_Notify_Time parameters reflect the values sent in the most recent notification. This test shall be performed only if the Protocol_Revision property is present in the Device object and has a value of 1 or 2.

Test Concept: The Trend Log is configured to acquire data by whatever means. Record_Count is set to zero. Collection of data proceeds until two notifications are seen, collection is halted and the values of Previous_Notify_Time and Current_Notify_Time are checked.

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Log_Enable shall be set to FALSE.

Test Steps:

1. WRITE Log_Enable = TRUE
2. MAKE (Trend Log object collect number of records specified by pThreshold)
3. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process ID),
 - 'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the Trend Log object being tested),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = BUFFER_READY,
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = NORMAL,
 'Event Values' = (BACnetObjectIdentifier of the IUT's Device object),
 (BACnetObjectIdentifier of the Trend Log object),
 (any BACnetDateTime),
 (current local BACnetDateTime)

4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (Trend Log object collect number of records specified by pThreshold)
6. RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the Trend Log object being tested),
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = BUFFER_READY,
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = NORMAL,
 'Event Values' = (BACnetObjectIdentifier of the IUT's Device object),
 (BACnetObjectIdentifier of the Trend Log object),
 (BACnetDateTime sent in step 3),
 (current local BACnetDateTime)
7. TRANSMIT BACnet-SimpleACK-PDU
8. WRITE Log_Enable = FALSE
9. IF (Previous_Notify_Time != Event Value parameter 3) THEN
ERROR "Previous_Notify_Time value is incorrect."
10. IF (Current_Notify_Time != Event Value parameter 4) THEN
ERROR "Current_Notify_Time value is incorrect."
11. IF (Event_Time_Stamps TO-NORMAL element != Event Value parameter 4) THEN
ERROR "Event_Time_Stamps value is incorrect."

7.3.2.24.12 COV Subscription Failure Test

Purpose: To verify that a failed COV subscription causes a TO-FAULT transition.

Test Concept: A Trend Log configured to acquire samples via COV is monitored to ensure that when a COV subscription fails, the object will go into fault.

Configuration Requirements: The Trend Log is configured to acquire data by COV subscription from the TD. Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Enable shall be set to TRUE.

Test Steps:

1. IF (the IUT uses SubscribeCOV for this Trend Log) THEN
BEFORE the lifetime of the COV subscription expires
RECEIVE SubscribeCOV-Request,

'Subscriber Process Identifier' = (any value),
 'Monitored Object Identifier' = (any object),
 'Issue Confirmed Notifications' = (TRUE|FALSE),
 'Lifetime' = (2 * COV_Resubscription_Interval)

ELSE

BEFORE the lifetime of the COV subscription expires

RECEIVE SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (any value),
 'Monitored Object Identifier' = (any object),
 'Issue Confirmed Notifications' = (TRUE|FALSE),
 'Lifetime' = (2 * COV_Resubscription_Interval),
 'Monitored Property Identifier' = (the property to be monitored),
 'COV Increment' = (Client_COV_Increment -- optional)

2. TRANSMIT BACnet-Error-PDU

'Error Class' = (any valid error class),
 'Error Code' = (any valid error code)

3. VERIFY Event_State = FAULT

7.3.2.24.13 Log-Status Test

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

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

Configuration Requirements: The logging object, O1, is configured to acquire data by whatever means available. Configure the logging such that the entire test may be run without the trend buffer overflowing.

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

Test Steps:

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

7.3.2.24.14 Time_Change Test

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

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

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

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Record_Count = 0
3. VERIFY (Log_Buffer contains 1 entry, and it is the buffer-purged event)
4. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (device that contains log object)
 'Property Identifier' = Local_Time
5. RECEIVE ReadProperty-Ack,
 'Object Identifier' = (device that contains log object)
 'Property Identifier' = Local_Time
 'Property Value' = (currentTime)
6. WRITE Enable = TRUE

7. MAKE (the time change on the device by a reasonable amount (deltaTime); change by one hour or more)
8. WRITE Enable = FALSE
9. VERIFY Record_Count => 4
10. CHECK (Log_Buffer contains a log-status entry of time-change)
11. VERIFY (time-change value ~= deltaTime)
12. VERIFY TimeStamp on the time-change entry ~= (currentTime + deltaTime)

7.3.2.24.15 COV-Sampling Verification Test

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

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

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

Test Steps:

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

7.3.2.24.16 Interval Gathering of External Trends Test

Purpose: To verify the IUT uses ReadProperty to pull external data at the specified intervals.

Test Concept: The log is configured to acquire data from the TD using polling. The TD verifies that the receipt of ReadProperty requests are at the Log_Interval set.

Configuration Requirements: The log shall be configured to be polling for external trends during the entire time of this test. The Stop_When_Full property, if configurable, shall be set to FALSE. Enable shall be set to TRUE. The TD shall be configured so that it does not support execution of ReadPropertyMultiple.

Test Steps:

1. BEFORE (Log_Interval)
 REPEAT X = (for each property logged) DO
 RECEIVE ReadProperty-Request,
 'Object Identifier' = (object that contains the monitored property)
 'Property Identifier' = (external property that is being trended)
 TRANSMIT ReadProperty-Ack
 'Object Identifier' = (object that contains the monitored property)
 'Property Identifier' = (property being monitored)
 'Property Value' = (any value)
2. WAIT (Log Interval)
3. REPEAT X = (for each property logged) DO
 RECEIVE ReadProperty-Request,
 'Object Identifier' = (object that contains the monitored property)
 'Property Identifier' = (external property that is being trended)
 TRANSMIT ReadProperty-Ack
 'Object Identifier' = (object that contains the monitored property)
 'Property Identifier' = (property being monitored)

- 'Property Value' = (any value)
 4. CHECK (to ensure all properties logged are requested)

7.3.2.24.17 Last_Notify_Record Test

Purpose: To verify that the Last_Notify_Record property reflects the values sent in the most recent notification.

Test Concept: The log buffer is cleared. The Log is allowed to collect records until it issues a BUFFER_READY notification. The value of the Last_Notify_Record property is checked.

Configuration Requirements: The log is configured to send BUFFER_READY notifications to the TD.

Test Steps:

1. WRITE Record_Count = 0
2. READ prev = Last_Notify_Record
3. WRITE Enable = TRUE
4. MAKE (Log object collect number of records specified by pThreshold)
5. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (the configured process identifier)
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = (Log object being tested),
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (configured notification class),
 - 'Priority' = (value configured to correspond to a TO-NORMAL),
 - 'Event Type' = BUFFER_READY,
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = EVENT | ALARM,
 - 'AckRequired' = TRUE | FALSE,
 - 'FromState' = NORMAL,
 - 'To State' = NORMAL, 'Event Values' = (pLogBuffer,
 - (pPreviousCount: prev),
 - (pMonitoredValue: C1))
6. TRANSMIT BACnet-SimpleACK-PDU
7. WRITE Enable = FALSE
8. VERIFY Last_Notify_Record = C1

7.3.2.24.18 Records_Since_Notification Test

Purpose: To verify that the Records_Since_Notification property reflects the number of records recorded by the log which have not yet been reported via a BUFFER_READY notification.

Test Concept: The log buffer is cleared. The log is allowed to collect records until it issues a BUFFER_READY notification and is halted before a second notification is generated. The value of the Records_Since_Notification property is checked.

Configuration Requirements: The logging object, LO1, is configured to send BUFFER_READY notifications to the TD.

Test Steps:

1. WRITE Enable = TRUE
2. WRITE Record_Count = 0
3. MAKE (LO1 collect a sufficient number of records in order to trigger a notification)
4. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (the configured process identifier)
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = LO1,
 - 'Time Stamp' = (any valid time stamp),
 - 'Notification Class' = (configured notification class),
 - 'Priority' = (value configured to correspond to a TO-NORMAL),
 - 'Event Type' = BUFFER_READY,
 - 'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'FromState' = NORMAL,
 'To State' = NORMAL,
 'Event Values' = (pLogBuffer,
 pPreviousCount,
 pMonitoredValue: C1)

5. TRANSMIT BACnet-SimpleACK-PDU
6. MAKE (LO1 collect N records, such that $N < pThreshold-1$)
7. WRITE Enable = FALSE
8. READ T1 = Total_Record_Count
9. VERIFY Records_Since_Notification = T1 - C1

7.3.2.24.19 Trigger Verification Test

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

Test Concept: The log, O1 is configured to log based on TRIGGERED. Logging is enabled. After a period of time the buffer is checked to verify that the data in the buffer is based on triggered values.

Configuration Requirements: The object being tested shall be configured with Logging_Type set to TRIGGERED.

Test Steps:

1. VERIFY Logging_Type = TRIGGERED
2. VERIFY Log_Interval = 0
3. WRITE Enable = FALSE
4. WRITE Record_Count = 0
5. WRITE Enable = TRUE
6. WAIT (10 seconds)
7. WRITE Trigger = TRUE
8. WAIT (20 seconds)
9. WRITE Trigger = TRUE
10. WAIT (40 seconds)
11. WRITE Trigger = TRUE
12. WAIT (30 seconds)
13. WRITE Enable = FALSE
14. READ N = Record_Count
15. REPEAT X = (1 through 3)
 - TRANSMIT ReadRange-Request
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Log_Buffer,
 - 'Reference Index' = N-4+X,
 - 'Count' = 1
 - RECEIVE ReadRange-ACK
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Log_Buffer,
 - 'Result Flags' = (False, False, False),
 - 'Item Count' = 1,
 - 'Item Data' = (one data record storing the timestamp in TS[X])
16. TRANSMIT ReadRange-Request
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Log_Buffer,
 - 'Reference Index' = N,
 - 'Count' = 1
17. RECEIVE ReadRange-ACK
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Log_Buffer,
 - 'Result Flags' = (False, True, False),
 - 'Item Count' = 1,

'Item Data' = (one data record storing the timestamp in TS[4])

18. CHECK (TS[2] - TS[1] $\approx$ 20 seconds)
19. CHECK (TS[3] - TS[2] $\approx$ 40 seconds)
20. CHECK (TS[4] - TS[3] $\approx$ 30 seconds)

7.3.2.24.20 Status/Failure Logging

Purpose: To verify that a failure is logged when an error is encountered in an attempt to read a data value from the monitored object.

Test Concept: Configure the Log_DeviceObjectProperty of the logging object with an unknown object such that collection of records fails. Wait until the IUT attempts to read a sample for the Log_Buffer then check the Log_Buffer to verify that there is a failure entry that consists of the ErrorClass and ErrorCode of the error. Repeat with Log_DeviceObjectProperty referencing an object in a device that does not exist.

Configuration Requirements: Configure the logging object so that collection of records will fail (such as by referencing a non-existent object).

Test Steps:

1. MAKE (Log_DeviceObjectProperty reference a non-existent object in the local device or in an existing remote device)
2. WAIT (until IUT attempts to read a sample for the Log_Buffer)
3. CHECK (Log_Buffer contains a failure entry of with an error class/error code of OBJECT/UNKNOWN_OBJECT)
4. IF the IUT supports logging remote values THEN {
 - MAKE (Log_DeviceObjectProperty reference an object in a non-existing device)
 - WAIT (until IUT attempts to read a sample for the Log_Buffer)
 - CHECK (Log_Buffer contains a failure entry with an error class/code COMMUNICATION/UNKNOWN_DEVICE)

7.3.2.24.21 Clock-Aligned Logging

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

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

Configuration Requirements: Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present, shall be configured in order that it occurs after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Interval_Offset is set to zero. Align_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. Logging_Mode is POLLED. X1 and X2 are each a value which the IUT supports for which the standard mandates the alignment behavior.

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

Test Steps:

1. CHECK (Log_Buffer contains 1 entry, and it is the buffer-purged event)
2. WRITE Log_Interval = X1
3. WRITE Enable = TRUE
4. MAKE (logging object collect at least 2 records)
5. WRITE Enable = FALSE
6. CHECK (Log_Buffer contains at least 5 entries, and at least two data records)
7. CHECK (that the timestamp of each data record, since the Log_Interval was written, is a multiple of X1)
8. WRITE Log_Interval = (X2, any value which requires alignment behavior, that was not already chosen)
9. WRITE Enable = TRUE
10. MAKE (logging object collect at least 2 more records)

11. WRITE Enable = FALSE
12. CHECK (Log_Buffer has collected two or more additional data records and two or more log-status records)
13. CHECK (that the timestamp are multiples of X2 for all data records collected, since the write with X2)

7.3.2.24.22 Logging Interval_Offset

Purpose: To verify that timestamps abide by the Interval_Offset.

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

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

Configuration Requirements: Align_Intervals is set to TRUE. The Log_DeviceObjectProperty property in a Trend Log or in a Trend Log Multiple, is configured to the property or properties monitored. Start_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop_Time, if present shall be configured in order that it occur after the end of the test. Stop_When_Full, if configurable, shall be set to FALSE. Enable is initially FALSE. Align_Intervals is set to TRUE. Triggering of non periodic log records should not occur during this test. If the Interval_Offset cannot be set to a value which the IUT supports greater than Log_Interval, then steps 11 through the end of this test are skipped. Logging_Mode is POLLED. An evenly divisible value is a value for which the standard mandates the alignment behavior.

Notes to tester: Interval_Offset in logging objects, and Log_Interval are each an Unsigned number of hundredths of seconds. Excellent choices are 400, 500, 600, 1000, or 1200. When Interval_Offset is larger than Log_Interval, then Interval_Offset modulo Log_Interval, is smaller than Log_Interval.

Test Steps:

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

7.3.2.24.23 Buffer_Size Write Test

Purpose: To verify the content of the log buffer after a write to the Buffer_Size property.

Test Concept: The logging object, O1, is configured to acquire data by whatever means. Logging is disabled and Buffer_Size set to a different valid value, V1. The content of the Log_Buffer is read to confirm a single entry that is a buffer purged event.

Configuration Requirements: The logging object, O1, is configured to acquire data by whatever means. If a write to the Buffer_Size does not delete all records in the log, this test shall be skipped.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Buffer_Size = V1

3. WAIT **Internal Processing Fail Time**
4. CHECK (Log_Buffer contains one entry, and it is a buffer-purged event)

7.3.2.25 Event Log Tests

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

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

7.3.2.25.1 Internal Logging of Notifications

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

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

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

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

Test Steps:

1. WRITE Enable = TRUE
2. MAKE (a condition exist that will cause the device to generate an event transition)
3. WAIT D1
4. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = (any valid object),
 - 'Time Stamp' = T1, any valid timestamp),
 - 'Notification Class' = (any valid notification class),
 - 'Priority' = (any valid priority),
 - 'Event Type' = (any standard event type),
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = ALARM | EVENT,
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = (state S1, any valid state for this event type),
 - 'To State' = (state S2, any valid state for this event type that can follow S1),
 - 'Event Values' = (any values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. MAKE (a condition exist that will cause the device to generate an event transition)
7. WAIT D2
8. RECEIVE ConfirmedEventNotification-Request,
 - 'Process Identifier' = (any valid process identifier),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = (any valid object),
 - 'Time Stamp' = (T2, any valid timestamp),
 - 'Notification Class' = (any valid notification class),
 - 'Priority' = (any valid priority),
 - 'Event Type' = (any standard event type),
 - 'Message Text' = (optional, any valid message text),
 - 'Notify Type' = ALARM | EVENT,
 - 'AckRequired' = TRUE | FALSE,
 - 'From State' = (state S3, any valid state for this event type),
 - 'To State' = (state S4, any valid state for this event type that can follow S3),
 - 'Event Values' = (any values appropriate to the event type)
9. TRANSMIT BACnet-SimpleACK-PDU
10. READ RC = LO1, Record_Count

11. TRANSMIT ReadRange-Request,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = RC,
 'Count' = -2
12. RECEIVE ReadRange-ACK,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = {?, ?, FALSE},
 'Item Count' = 2,
 'Item Data' = (logged data that matches the information received in steps 3 and 6,
 except that Process Identifier may be any value and is not required to match)
13. CHECK (T2 > T1, and that the notifications were logged in order)

7.3.2.25.2 Remote Logging of Notifications

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

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

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

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

Test Steps:

1. WRITE Enable = TRUE
2. WAIT **Internal Processing Fail Time**
2. TRANSMIT ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process identifier),
 'Initiating Device Identifier' = TD,
 'Event Object Identifier' = (any valid object identifier),
 'Time Stamp' = (T1, any valid timestamp),
 'Notification Class' = (any valid notification class),
 'Priority' = (any valid priority),
 'Event Type' = (any standard event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = (state S1, any valid state for this event type),
 'To State' = (state S2, any valid state for this event type that can follow S1),
 'Event Values' = (any values appropriate to the event type)
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process identifier),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (any valid object identifier),
 'Time Stamp' = (T2, any valid timestamp),
 'Notification Class' = (any valid notification class),
 'Priority' = (any valid priority),
 'Event Type' = (any standard event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = (state S3, any valid state for this event type),
 'To State' = (state S4, any valid state for this event type that can follow S3),
 'Event Values' = (any values appropriate to the event type)
5. RECEIVE BACnet-SimpleACK-PDU

6. READ RC = LO1, Record_Count
7. TRANSMIT ReadRange-Request,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = RC,
 'Count' = -2
8. RECEIVE ReadRange-ACK,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = {?, ?, FALSE},
 'Item Count' = 2,
 'Item Data' = (logged data that matches the information received in steps 2 and 4,
 except that Process_Identifier can be any value and is not required to match)
9. CHECK (that the events were logged in the order in which they were received)

7.3.2.25.3 Internal Logging of ACK_NOTIFICATIONs

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

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

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

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

Test Steps:

1. WRITE Enable = TRUE
2. MAKE (the IUT generate a notification)
3. RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (PI1, any valid process identifier),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (T1, any valid timestamp),
 'Notification Class' = (N1, any valid notification class),
 'Priority' = (P1, any valid priority),
 'Event Type' = (ET1, any standard event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE | FALSE,
 'From State' = (S1, any valid state for this event type),
 'To State' = (S2, any valid state for this event type),
 'Event Values' = (any values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT AcknowledgeAlarm-Request,
 'Acknowledging Process Identifier' = (any valid value),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S2,
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgment' = (the current time)
6. RECEIVE BACnet-SimpleACK-PDU
7. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = PI1,
 'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,
 'Time Stamp' = (T2, any valid timestamp > T1),
 'Notification Class' = N1,
 'Priority' = P1,
 'Event Type' = ET1,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ACK_NOTIFICATION,
 'From State' = S1

8. TRANSMIT BACnet-SimpleACK-PDU
9. READ RC = LO1, Record_Count
10. TRANSMIT ReadRange-Request,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = RC,
 'Count' = -1
11. RECEIVE ReadRange-ACK,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = {?, ?, FALSE},
 'Item Count' = 1,
 'Item Data' = (logged data that matches the information received in step 7,
 except that Process_Identifier can be any value and is not
 required to match)

7.3.2.25.4 Remote Logging of ACK_NOTIFICATIONs

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

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

Configuration Requirements: LO1 is an Event Log object in the IUT which logs ACK_NOTIFICATIONs. Stop_When_Full in LO1 shall be FALSE or absent.

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

Test Steps:

1. WRITE Enable = TRUE
2. WAIT **Internal Processing Fail Time**
3. TRANSMIT ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process identifier),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (any valid object identifier),
 'Time Stamp' = (T1, any valid timestamp),
 'Notification Class' = (any valid notification class),
 'Priority' = (any valid priority),
 'Event Type' = (any standard event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = ACK_NOTIFICATION,
 'From State' = (state S1, any valid state for this event type)
4. RECEIVE BACnet-SimpleACK-PDU
5. READ RC = LO1, Record_Count
6. TRANSMIT ReadRange-Request,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = RC,
 'Count' = -1

7. RECEIVE ReadRange-ACK,
 'Object Identifier' = LO1,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = {?, ?, FALSE},
 'Item Count' = 1,
 'Item Data' = (logged data that matches the information received in step 3,
 except that Process_Identifier can be any value and is not required to match)

7.3.2.26 Moved to 7.3.2.25.2

7.3.2.27 Moved to 7.3.2.25.3

7.3.2.28 Moved to 7.3.2.25.4

7.3.2.29 Structured View Tests

7.3.2.29.1 Subordinate_List Size Changes Subordinate_Annotations

Purpose: This test verifies that when the size of the Subordinate_List array is changed, the size of the Subordinate_Annotations array is changed accordingly to the same size. In addition, the test case verifies that the Instance Number in any uninitialized objectIdentifiers in the Subordinate_List has the value 4194303. If the size of the Subordinate_List and Subordinate_Annotations arrays cannot be changed, then this test shall not be performed.

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

Configuration Requirements: The IUT shall be configured with a Structured View object with resizable Subordinate_List and Subordinate_Annotations arrays.

Test Steps:

1. WRITE Subordinate_List = 2, ARRAY INDEX = 0
2. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
3. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
4. WRITE Subordinate_List = (L1, some value greater than 2), ARRAY INDEX = 0
5. VERIFY Subordinate_List = L1, ARRAY INDEX = 0
6. REPEAT X = (values greater than 2 up to L1) DO {
 VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
 }
7. VERIFY Subordinate_Annotations = L1, ARRAY INDEX = 0
8. WRITE Subordinate_List = (Subordinate_List array of length 2)
9. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
10. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
11. WRITE Subordinate_List = (Subordinate_List array of length L2 greater than 2)
12. VERIFY Subordinate_List = L2, ARRAY INDEX = 0
13. VERIFY Subordinate_Annotations = L2, ARRAY INDEX = 0
14. WRITE Subordinate_List = 2, ARRAY INDEX = 0
15. VERIFY Subordinate_List = (an array consisting of elements 1 & 2 from the array written in step 11)
16. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0

7.3.2.29.2 Subordinate_Annotations Size Changes Subordinate_List

Purpose: This test verifies that when the size of the Subordinate_Annotations array is changed, the size of the Subordinate_List array is changed accordingly to the same size. In addition, the test case verifies that the Instance Number in any uninitialized objectIdentifiers in the Subordinate_List has the value 4194303. If the size of the Subordinate_Annotations and Subordinate_List arrays cannot be changed, then this test shall not be performed.

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

Configuration Requirements: The IUT shall be configured with a Structured View object with resizable Subordinate_Annotations and Subordinate_List arrays.

Test Steps:

1. WRITE Subordinate_Annotations = 2, ARRAY INDEX = 0
2. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
3. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
4. WRITE Subordinate_Annotations = (L1, some value greater than 2), ARRAY INDEX=0
5. VERIFY Subordinate_Annotations = L1, ARRAY INDEX = 0
6. REPEAT X = (values greater than 2 up to L1) DO {
 VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
 }
7. VERIFY Subordinate_List = L1, ARRAY INDEX = 0
8. WRITE Subordinate_Annotations = (Subordinate_Annotations array of length 2)
9. VERIFY Subordinate_Annotations = 2, ARRAY INDEX = 0
10. VERIFY Subordinate_List = 2, ARRAY INDEX = 0
11. WRITE Subordinate_Annotations = (Subordinate_Annotations array of length L2 greater than 2)
12. VERIFY Subordinate_Annotations = L2, ARRAY INDEX = 0
13. VERIFY Subordinate_List = L2, ARRAY INDEX = 0
14. REPEAT X = (values greater than 2 up to L2) DO {
 VERIFY (Instance Number within the Subordinate_List objectIdentifier) = 4194303, ARRAY INDEX = X
 }
15. WRITE Subordinate_Annotations = 2, ARRAY INDEX = 0
16. VERIFY Subordinate_Annotations = (an array consisting of elements 1 & 2 from the array written in step 11)
17. VERIFY Subordinate_List = 2, ARRAY INDEX = 0

7.3.2.30 Notification Forwarder Object Tests

This clause defines the tests necessary to demonstrate Notification Forwarder Object functionality.

7.3.2.30.1 Common values and configurations used in all Notification Forwarder object tests

All tests use the value names listed below for configuration and verification of functionality. Most of the tests begin with either base setup 1 configuration or base setup 2 configuration defined below. Notifications originate from the notification source object and are distributed by the IUT Notification Forwarder object to the destinations. A notification source object may be any BACnet event-initiating object.

7.3.2.30.1.1 Values used in all Notification Forwarder object tests

Values are split into two categories indicated by the first characters which indicate SRC (Source) or DEST (Destination). After the category are additional characters indicating the meaning of the parameter:

SRC (Source) category. Used by notification source and Notification Class objects sending messages directed to the IUT Notification Forwarder object(s); or in the case that Local_Forwarding_Only is TRUE for the IUT, then the objects within the IUT device that are sending messages to the IUT Notification Forwarder object(s).

SRC_NOTIF_DEV	Device containing the notification source object
SRC_NOTIF_OBJ	Object used as the notification source object
SRC_NOTIF_CLS	Notification Class object used by the notification source object
SRC_PROCESS_ID	Process Identifier value used in the Recipient_List of the Notification Class object used by the notification source object
SRC_CONF_NOTIF	Issue Confirmed Notifications value used in the Recipient_List of the Notification Class object used by the notification source object
SRC_NOTIF_TYP	Notify Type value used by the source notification object

DEST (Destination) category. Used by the IUT Notification Forwarder object(s) when forwarding messages.

DEST_OBJ_ID	Recipient used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test
DEST_PROCESS_ID	Process_Identifier value used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test
DEST_CONF_NOTIF	Issue Confirmed Notifications value used in the Recipient_List or Subscribed_Recipients list of the IUT Notification Forwarder object under test

Thus SRC_CONF_NOTIF indicates the (source) confirmed notifications selection for event notifications directed to the IUT Notification Forwarder object(s) while DEST_CONF_NOTIF indicates the (destination) confirmed notification selection for event notifications forwarded by the IUT Notification Forwarder object(s) to destination devices.

Other commonly used values include:

TR	Time Remaining
SR	Subscribed Recipients

Values indicated by these name codes can be selected by the tester within the requirements of the specification, and the test but must be a single unchanging value for the duration of each test. Values may be changed between tests.

Devices commonly specified include:

DS	Device notification Source, referenced by the value of SRC_NOTIF_DEV previously defined.
D1	Destination Device 1, referenced by the value of DEST_OBJ_ID previously defined.
D2, D3...DX	Other distinct destination devices that may be specified in a test.

When multiple destination devices are required for a test, those devices shall be distinct unless the test specifically indicates otherwise. Destination devices may be connected to be reachable through any combination of available ports on the IUT device appropriate for the test.

TD shall be connected to monitor the port or ports used by the IUT to transmit to the destination devices.

7.3.2.30.1.2 Base setup 1 for Notification Forwarder object tests

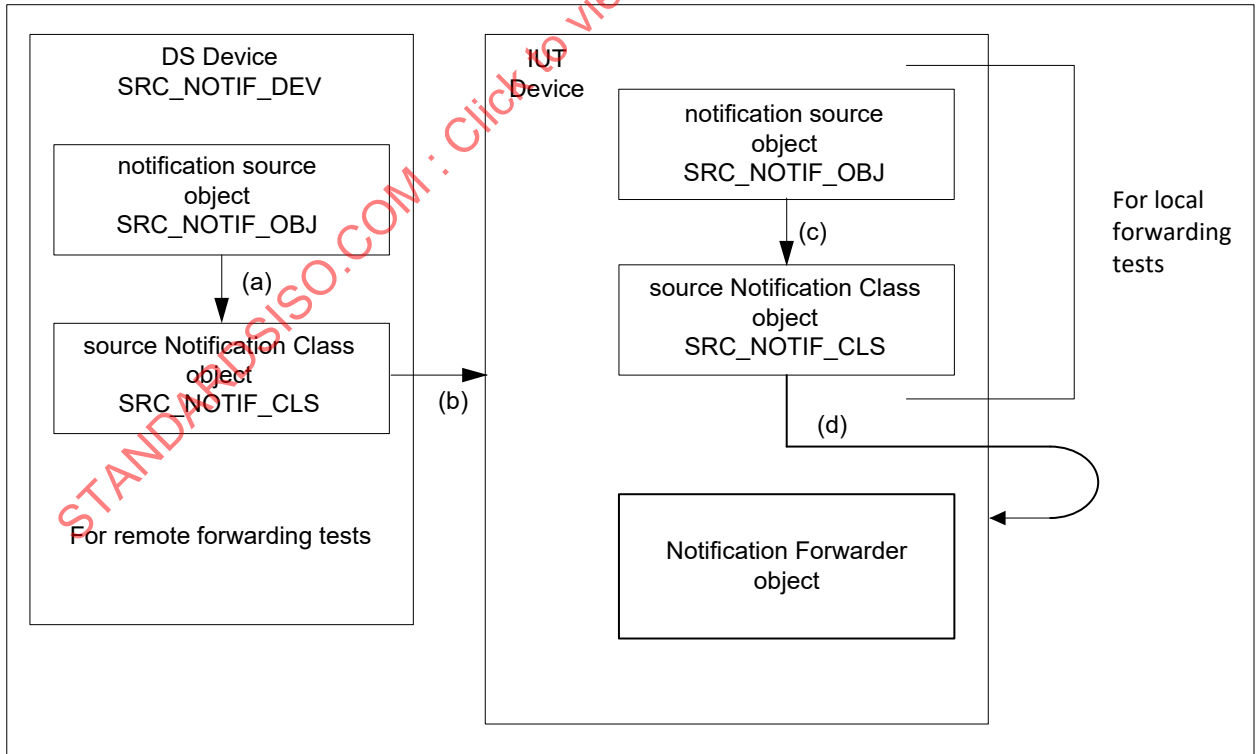


Figure 7-31-1. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 1.

Base setup 1 as shown in Figure 7-31-1 requires that DS include two objects, a Notification source object (SRC_NOTIF_OBJ) and a Source notification class object (SRC_NOTIF_CLS). The DS Notification source object is an object capable of generating an event notification such as an analog value object with intrinsic alarming. The Notification

source object's Notification_Class property references (a) the DS's Source notification class object which is a BACnet notification class object containing a BACnetDestination (b) referencing the IUT. The DS Notification source object is used to produce event notifications for distributing by the IUT Notification Forwarder Object unless the test indicates otherwise or unless Local_Forwarding_Only in the Notification Forwarder Object is TRUE and the test does not indicate otherwise.

Base setup 1 also requires that the IUT include three objects, a Notification source object (SRC_NOTIF_OBJ), a Source notification class object (SRC_NOTIF_CLS), and the Notification Forwarder Object being tested. The IUT Notification source object is an object capable of generating an event notification such as an analog value object with intrinsic alarming. The Notification source object's Notification_Class property references (c) the IUT's Source notification class object which is a BACnet notification class object containing a BACnetDestination (d) referencing the IUT. The IUT's Notification source object is used to produce event notifications for distributing by the IUT Notification Forwarder Object when Local_Forwarding_Only is TRUE and the test does not indicate otherwise.

Configure a Notification Forwarder object in the IUT as follows:

Out_Of_Service = FALSE.

Recipient_List = empty

Subscribed_Recipients list = empty

Process_Identifier_Filter = NULL.

Enable all Port_Filter array entries (if present).

If the IUT has no Notification Forwarder object with a writable, NULL or zero Process_Identifier_Filter property, then select a Process_Identifier value SRC_PROCESS_ID in the source notification class below to match the Process_Identifier_Filter value.

Configure a Notification source object in DS (or in the IUT when Local_Forwarding_Only is TRUE) as follows:

Source notification device = SRC_NOTIF_DEV, -- DS (or IUT if Local_Forwarding_Only is TRUE)

Source notification object = SRC_NOTIF_OBJ,

Notification_Class = SRC_NOTIF_CLS -- (a) or (c)

Event_Enable = {T, T, T}

Configure a Source notification class SRC_NOTIF_CLS in DS (or in the IUT when Local_Forwarding_Only is TRUE) as follows:

Notification Class = instance number of notification class object,

Priority = (any valid priority)

Ack_Required = (any valid Ack_Required value)

Recipient_List = {(all), -- Valid Days

(all), -- From Time, To Time

(IUT as a recipient (b) or (d) compatible with the value of

Local_Forwarding_Only in the Notification Forwarder

Object under test) -- Recipient

} -- One list element

Process Identifier = SRC_PROCESS_ID

Issue Confirmed Notifications = SRC_CONF_NOTIF

Transitions = {T, T, T}

SRC_CONF_NOTIF is assumed to be FALSE unless specified within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected unless specified otherwise. Behaviors can alternately be tested using SRC_CONF_NOTIF set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Local_Forwarding_Only is assumed to be FALSE unless specified within the test. Thus event notifications are assumed to originate from DS and are directed to the IUT unless specified within the test. Behaviors can alternately be tested with Local_Forwarding_Only set TRUE if no value is specified in the test and the IUT is capable of containing this value. In this case, event notifications shall originate from a Notification source object within the IUT device and are directed to the IUT object, and it is necessary for the tester to MAKE the IUT Notification source object generate an UnconfirmedEventNotification-Request with the specified parameters rather than TRANSMIT an UnconfirmedEventNotification-Request. It is not necessary to test both settings unless specifically directed by the test.

7.3.2.30.1.3 Base setup 2 for Notification Forwarder object tests

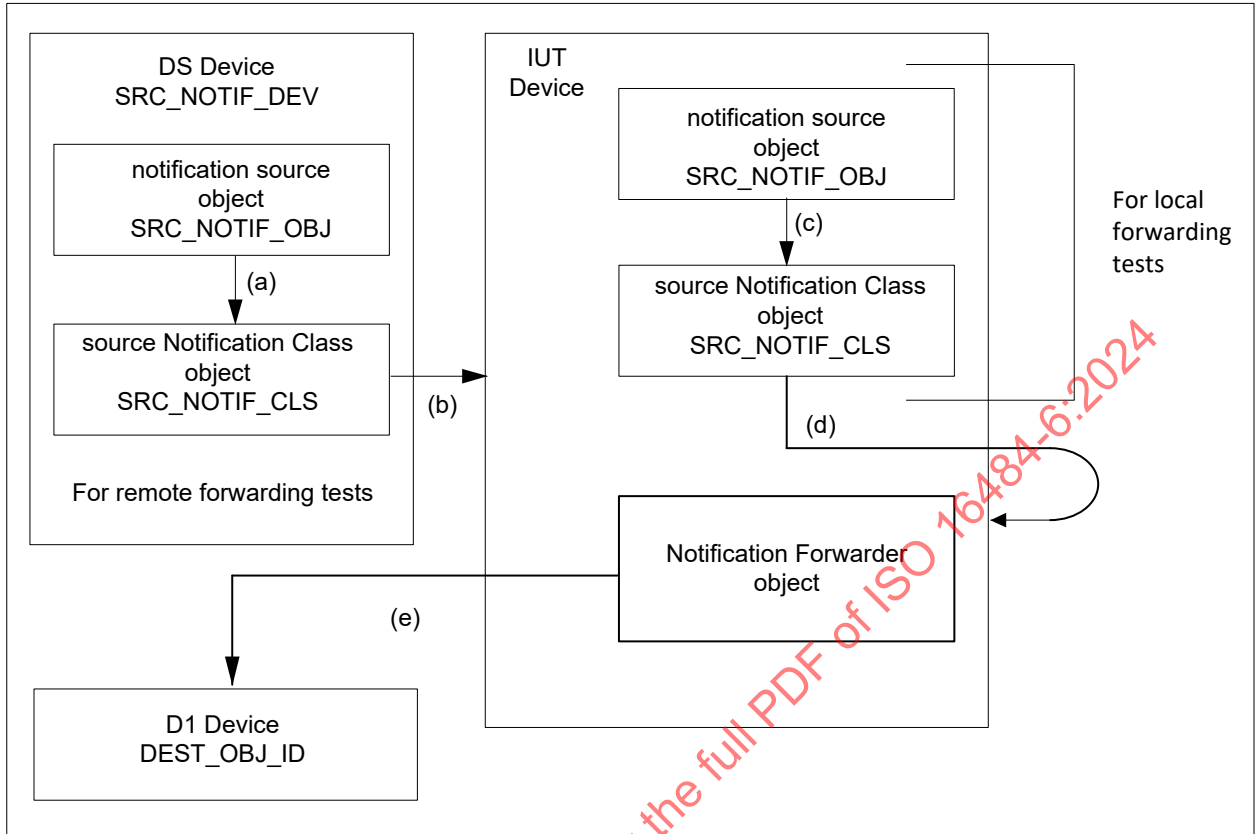


Figure 7-31-2. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 2.

Base setup 2 as shown in Figure 7-31-2 is the same as base setup 1 except that the IUT's Notification Forwarder Object has a Subscribed_Recipient (e) added referencing D1 while base setup 1 has no defined recipients. Base setup 2 is used when a single subscribed recipient can be used to complete the test or is a logical starting point for the test.

This is achieved as follows:

Beginning with Base setup 1, add D1 as a subscribed recipient (e) to the IUT Notification Forwarder Object by performing the following additional steps:

1. TRANSMIT AddListElement-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Subscribed_Recipients,
 List of Elements = {DEST_OBJ_ID, -- Recipient (e)
 DEST_PROCESS_ID, -- Any Process Identifier
 DEST_CONF_NOTIF, -- Issue Confirmed Notifications
 TR -- Time Remaining where TR > test duration
 } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU

DEST_CONF_NOTIF is assumed to be FALSE unless specified within the test. Thus UnconfirmedEventNotification-Request messages from the Notification Forwarder object are expected unless specified otherwise. Behaviors can alternately be tested using DEST_CONF_NOTIF set to TRUE resulting in ConfirmedEventNotification service messages from the NotificationForwarder object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

7.3.2.30.2 Recipient_List Forwarding Test

Purpose: Verify that an event notification can be forwarded to a Recipient device in the Recipient_List. This test is used as the basis for several additional tests that require more specific value choices.

Test Concept: Write or configure a Recipient_List entry and then send an event notification to the IUT. Insure that the event notification is forwarded to the recipient device with the correct parameters. Parameter values shall be selected for the test as a consistent set indicated in the test steps.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests

Test Steps:

1. MAKE (Recipient_List = { (all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
2. IF (SRC_CONF_NOTIF is FALSE) THEN
 TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 ELSE
 TRANSMIT SOURCE = DS, ConfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 RECEIVE DESTINATION = DS, BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 IF (DEST_CONF_NOTIF is FALSE) THEN
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)
 ELSE
 RECEIVE DESTINATION = D1, ConfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)
TRANSMIT SOURCE = D1, BACnet-SimpleACK-PDU

7.3.2.30.3 Subscribed_Recipients Forwarding Test

Purpose: Verify that an event notification can be forwarded to a Recipient device in the Subscribed_Recipients. This test is used as the basis for several additional tests that require more specific value choices.

Test Concept: Add a Subscribed_Recipients list entry and then send an event notification to the IUT. Insure that the event notification is forwarded to the recipient device with the correct parameters. Parameter values shall be selected for the test as a consistent set indicated in the test steps.

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

Test Steps:

1. IF (SRC_CONF_NOTIF is FALSE) THEN

TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier' = SRC_PROCESS_ID,

'Initiating Device Identifier' = SRC_NOTIF_DEV,

'Event Object Identifier' = SRC_NOTIF_OBJ,

'Time Stamp' = (any valid time stamp),

'Notification Class' = SRC_NOTIF_CLS,

'Priority' = (any valid priority),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = SRC_NOTIF_TYP,

'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),

'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

ELSE

TRANSMIT SOURCE = DS, ConfirmedEventNotification-Request,

'Process Identifier' = SRC_PROCESS_ID,

'Initiating Device Identifier' = SRC_NOTIF_DEV,

'Event Object Identifier' = SRC_NOTIF_OBJ,

'Time Stamp' = (any valid time stamp),

'Notification Class' = SRC_NOTIF_CLS,

'Priority' = (any valid priority),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = SRC_NOTIF_TYP,

'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),

'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

RECEIVE DESTINATION = DS, BACnet-SimpleACK-PDU

2. BEFORE Notification Fail Time

IF (DEST_CONF_NOTIF is FALSE) THEN

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request

'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)

ELSE

RECEIVE DESTINATION = D1, ConfirmedEventNotification-Request

'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)
TRANSMIT SOURCE = D1, BACnet-SimpleACK-PDU

7.3.2.30.4 Confirmed And Unconfirmed Forwarding Test

7.3.2.30.4.1 Confirmed Source And Confirmed Destination Forwarding Test

Purpose: Check that a confirmed source event notification is forwarded as a confirmed event notification when specified in the IUT Notification Forwarder recipient settings as confirmed.

Test Concept: Perform tests 7.3.2.31.2 Recipient_List Forwarding Test and 7.3.2.31.3 Subscribed_Recipients Forwarding Test for SRC_CONF_NOTIF set to TRUE (confirmed), and DEST_CONF_NOTIF set to TRUE (confirmed).

7.3.2.30.4.2 Confirmed Source And Unconfirmed Destination Forwarding Test

Purpose: Check that a confirmed source event notification is forwarded as an unconfirmed event notification when specified in the IUT Notification Forwarder recipient settings as unconfirmed.

Test Concept: Perform tests 7.3.2.31.2 Recipient_List Forwarding Test and 7.3.2.31.3 Subscribed_Recipients Forwarding Test for SRC_CONF_NOTIF set to TRUE (confirmed), and DEST_CONF_NOTIF set to FALSE (unconfirmed).

7.3.2.30.4.3 Unconfirmed Source And Confirmed Destination Forwarding Test

Purpose: Check that an unconfirmed source event notification is forwarded as a confirmed event notification when specified in the IUT Notification Forwarder recipient settings as confirmed.

Test Concept: Perform tests 7.3.2.31.2 Recipient_List Forwarding Test and 7.3.2.31.3 Subscribed_Recipients Forwarding Test for SRC_CONF_NOTIF set to FALSE (unconfirmed), and DEST_CONF_NOTIF set to TRUE (confirmed).

7.3.2.30.4.4 Unconfirmed Source And Unconfirmed Destination Forwarding Test

Purpose: Check that an unconfirmed source event notification is forwarded as an unconfirmed event notification when specified in the IUT Notification Forwarder recipient settings as unconfirmed.

Test Concept: Perform tests 7.3.2.31.2 Recipient_List Forwarding Test and 7.3.2.31.3 Subscribed_Recipients Forwarding Test for SRC_CONF_NOTIF set to FALSE (unconfirmed), and DEST_CONF_NOTIF set to FALSE (unconfirmed).

7.3.2.30.5 Character Encoding Test

Purpose: Verify that no event notifications are ignored due to the choice of character set in an event notification.

Test Concept: Send event notifications using character set encoding X to a Notification Forwarder. Character set encoding X should be chosen to be different than the character set encoding selected or used by the IUT device for at least one execution of this test. One or more Message Text characters shall be chosen as multi-byte characters. In the test steps, the event notification is sent twice, once testing the Notification Forwarder Subscribed_Recipients property and once testing the Notification Forwarder Recipient_List property. For each event notification sent, the forwarded event notification is examined. The test shall be performed with Local_Forwarding_Only set to FALSE. If the IUT is not capable of having Local_Forwarding_Only set to FALSE, then this test shall be omitted.

Configuration Requirements:

Base setup 1 for Notification Forwarder object tests.

Local_Forwarding_Only = FALSE.

Notification source Message Text = (any valid message text with character encoding X and not empty)

Test Steps:

1. TRANSMIT AddListElement-Request,
 - 'Object Identifier' = (the object being tested),
 - 'Property Identifier' = Subscribed_Recipients,
 - 'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration until WAIT statement
} -- One list element
2. RECEIVE BACnet-SimpleACK-PDU

3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (required, any valid message text with character encoding X and not empty),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
4. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
 'Message Text' = (message text from previous step with character encoding = X)
 (Other parameters omitted for clarity)
5. WAIT (TR+1) minutes for Time Remaining to expire
6. MAKE (Recipient_List = { (all), -- Valid Days
 (all), -- From Time, To Time
 DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 FALSE, -- Issue Confirmed Notifications
 {T, T, T} -- Transitions
 }) -- One list element
7. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (any valid message text with character encoding X and not empty), --
 required for this test
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
8. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request,
 'Message Text' = (message text from previous step with character encoding = X)
 (Other parameters omitted for clarity)

7.3.2.30.6 Out_Of_Service Property Test

Purpose: This test case verifies that event forwarding is not done while Out_Of_Service is TRUE.

Test Concept: Set up both Recipient_List and Subscribed_Recipient recipient entries with no filters specified and then send event notifications to the Notification Forwarder while the value of the Out_Of_Service property is TRUE. Subscribed_Recipients are configured as part of base setup 2 for Notification Forwarder object tests. Verify that forwarding of the event notifications is not performed.

Configuration Requirements: The selected object is configured such that its Out_Of_Service shall be set to FALSE and Reliability set to NO_FAULT_DETECTED. Base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Test Steps:

1. MAKE (Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
DEST_OBJ_ID2, -- Recipient D2
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
{T, T, T} -- Transitions
}) -- One list element
2. MAKE (Out_Of_Service = TRUE)
3. VERIFY Out_Of_Service = TRUE
4. VERIFY Status_Flags = (FALSE, FALSE, FALSE, TRUE)
5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
6. WAIT **Notification Fail Time**
7. CHECK (the IUT did not transmit an event notification)
8. MAKE (Out_Of_Service = FALSE)
9. VERIFY Out_Of_Service = FALSE
10. VERIFY Status_Flags = (FALSE, FALSE, FALSE, FALSE)
11. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
12. BEFORE **Notification Fail Time** --The following can be in any order
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request
13. MAKE (Out_Of_Service = TRUE)

14. VERIFY Out_Of_Service = TRUE
15. VERIFY Status_Flags = (FALSE, FALSE, FALSE, TRUE)
16. IF (Reliability is writable) THEN
 - REPEAT X = (all values of the Reliability enumeration appropriate to the object type except NO_FAULT_DETECTED) DO {
 - WRITE Reliability = X
 - VERIFY Reliability = X
 - VERIFY Status_Flags = (?, TRUE, ?, TRUE)
 - WRITE Reliability = NO_FAULT_DETECTED
 - VERIFY Reliability = NO_FAULT_DETECTED
 - VERIFY Status_Flags = (?, FALSE, ?, TRUE)
17. IF (Out_Of_Service is writable) THEN
 - WRITE Out_Of_Service = FALSE
 - ELSE
 - MAKE (Out_Of_Service = FALSE)
18. VERIFY Out_Of_Service = FALSE
19. VERIFY Status_Flags = (?, FALSE, ?, FALSE)

7.3.2.30.7 Recipient_List Property Tests

7.3.2.30.7.1 Destination Date Filtering Test

Purpose: Test destination date filtering operation by checking that forwarding of event notifications obeys the recipient validDays settings.

Test Concept: Configure a Recipient_List entry in the Notification Forwarder object such that the validDays is enabled for some days and disabled for some days. Set the IUT device date to an enabled day TIME_E. Send an event notification to the Notification Forwarder object and check that the event notification is forwarded. Change the IUT device's day to a disabled day TIME_D and send another event notification. Check that the event notification is not forwarded.

TIME_E is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

TIME_D is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the disabled days.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {(at least one day of week has a value of TRUE, at least one day of week has a value of FALSE), --Valid Days
 - (any range sufficient for the duration of the test), -- From Time, To Time
 - DEST_OBJ_ID, -- Recipient D1
 - DEST_PROCESS_ID, -- Process Identifier
 - FALSE, -- Issue Confirmed Notifications
 - {T, T, T} -- Transitions
 - } -- One list element
2. IF (IUT supports the TimeSynchronization service) THEN
 - TRANSMIT TimeSynchronization-Request,
 - 'Time' = TIME_E
 - ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 - TRANSMIT UTCTimeSynchronization-Request,
 - 'Time' = TIME_E
 - ELSE
 - MAKE (the local date and time = TIME_E)
3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 - 'Process Identifier' = SRC_PROCESS_ID,
 - 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 - 'Event Object Identifier' = SRC_NOTIF_OBJ,

'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

4. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)

5. IF (IUT supports the TimeSynchronization service) THEN
 TRANSMIT TimeSynchronization-Request,
 'Time' = TIME_D
 ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 TRANSMIT UTCTimeSynchronization-Request,
 'Time' = TIME_D
 ELSE
 MAKE (the local date and time = TIME_D)

7. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

8. WAIT **Notification Fail Time**

9. CHECK (the IUT did not transmit an event notification)

10. IF (IUT supports the TimeSynchronization service) THEN
 TRANSMIT TimeSynchronization-Request,
 'Time' = (current date and time)
 ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 TRANSMIT UTCTimeSynchronization-Request,
 'Time' = (current date and time)
 ELSE
 MAKE (the local date and time = (current date and time))

7.3.2.30.7.2 Destination Time Filtering Test

Purpose: Test destination time filtering operation by checking that forwarding of event notifications obeys the recipient fromTime and toTime settings.

Test Concept: Configure a Recipient_List entry in the Notification_Forwarder object such that the fromTime and toTime includes some period of time. Set the IUT device time to a value TIME_E between fromTime and toTime. Send an event notification to the Notification Forwarder object and check that the event notification is forwarded. Change the IUT

device's time to TIME_D so that it is outside the range of fromTime and toTime and send another event notification. Check that the event notification is not forwarded.

TIME_E is any time within the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

TIME_D is any time outside the window defined by From Time and To Time in the Recipient_List that corresponds to one of the enabled days.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = { (at least one day of week has a value of TRUE), -- Valid Days
 (any range sufficient for the duration of the test), -- From Time, To Time
 DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 FALSE, -- Issue Confirmed
- Notifications {T, T, T} -- Transitions
 }) -- One list element
2. IF (IUT supports the TimeSynchronization service) THEN
 TRANSMIT TimeSynchronization-Request,
 'Time' = TIME_E
 ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 TRANSMIT UTCTimeSynchronization-Request,
 'Time' = TIME_E
 ELSE
 MAKE (the local date and time = TIME_E)
 3. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
- ACK_NOTIFICATION 'From State' = (any valid From_State), -- absent if Notify Type is
- ACK_NOTIFICATION 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
- ACK_NOTIFICATION
4. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)
 5. IF (IUT supports the TimeSynchronization service) THEN
 TRANSMIT TimeSynchronization-Request,
 'Time' = TIME_D
 ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 TRANSMIT UTCTimeSynchronization-Request,
 'Time' = TIME_D
 ELSE
 MAKE (the local date and time = TIME_D)

6. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
8. WAIT **Notification Fail Time**
9. CHECK (the IUT did not transmit an event notification)
10. IF (IUT supports the TimeSynchronization service) THEN
 TRANSMIT TimeSynchronization-Request,
 'Time' = (current date and time)
 ELSE IF (IUT supports the UTCTimeSynchronization service) THEN
 TRANSMIT UTCTimeSynchronization-Request,
 'Time' = current date and time)
 ELSE
 MAKE (the local date and time = (current date and time))

7.3.2.30.7.3 Process Identifier Test

Purpose: Check that the process identifier values used in forwarded event notifications is equal to the values used in the recipient configurations contained in the IUT.

Test Concept: Perform tests 7.3.2.30.2 Recipient_List Forwarding Test and 7.3.2.30.3 Subscribed_Recipients Forwarding Test for different values of SRC_PROCESS_ID, and DEST_PROCESS_ID. The following test cases are required at a minimum.

Case 1 - SRC_PROCESS_ID <> DEST_PROCESS_ID and both are non-zero.

Case 2 - SRC_PROCESS_ID = DEST_PROCESS_ID and both are non-zero.

7.3.2.30.7.4 Destination Transition Filtering Test

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

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

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

- MAKE (Recipient_List = {(all), -- Valid Days
 (all), -- From Time, To Time
 DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 FALSE, -- Issue Confirmed Notifications
 any valid value} -- Transitions
 }) -- One list element

2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
 'AckRequired' = (any valid value),
 'From State' = normal,
 'To State' = offnormal, -- offnormal transition
 'Event Values' = (any valid event values)
3. IF (offnormal transitions are enabled in the Recipient_List)
 BEFORE Notification Fail Time
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)
- ELSE
 WAIT Notification Fail Time
 CHECK (the IUT did not transmit an event notification)
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
 'AckRequired' = (any valid value),
 'From State' = offnormal,
 'To State' = normal, -- normal transition
 'Event Values' = (any valid event values)
5. IF (normal transitions are enabled in the Recipient_List) THEN
 BEFORE Notification Fail Time
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)
- ELSE
 WAIT Notification Fail Time
 CHECK (the IUT did not transmit an event notification)
6. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP, -- choose ALARM or EVENT
 'AckRequired' = (any valid value),
 'From State' = normal,
 'To State' = fault, -- fault transition
 'Event Values' = (any valid event values)
7. IF (fault transitions are enabled in the Recipient_List) THEN
 BEFORE Notification Fail Time

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)

ELSE

WAIT **Notification Fail Time**

CHECK (the IUT did not transmit an event notification)

7.3.2.30.8 Subscribed_Recipients Property Test

7.3.2.30.8.1 Time Count Down Test

Purpose: Verify Subscribed_Recipients entries count Time Remaining down.

Test Concept: Add a subscription to a Notification Forwarder and check that the Subscribed_Recipients Time Remaining value decreases after time has passed.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

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

Test Steps:

1. READ SR1 = Subscribed_Recipients
2. CHECK (SR1 = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR1 -- Time Remaining where (TR-1) <= TR1 <= TR
}) -- One list element
3. WAIT greater of 2 minutes or twice timing resolution
4. READ SR2 = Subscribed_Recipients
5. CHECK (SR2 = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR2 -- Time Remaining where TR2 < TR1
}) -- One list element

7.3.2.30.8.2 Expiration Test

Purpose: Verify Subscribed_Recipients entries expire when Time Remaining reaches zero.

Test Concept: Add a subscription to a Notification Forwarder and verify that the subscription was successful. Then wait for the Subscribed_Recipients Time Remaining value to expire and check that the subscription has been removed from the Subscribed_Recipients list.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
2 -- Time Remaining minutes
} -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed_Recipients

4. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR1 -- Time Remaining where $1 \leq TR1 \leq 2$
}) -- One list element
5. WAIT more than greater of 2 minutes or twice timing resolution
6. VERIFY Subscribed_Recipients = { } -- Empty list

7.3.2.30.8.3 Time Renewal Test

Purpose: Verify that a Subscribed_Recipients resubscription renews the Time Remaining value.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder and wait until the Time Remaining value should have decreased. Renew the subscription and check that the Time Remaining has been restored to near the original value and that no additional list entries have been made.

Note to tester: If the device timing resolution is significantly greater than 1 minute, the wait times called for by the test steps may be extended.

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

Test Steps:

1. WAIT greater of 2 minutes or twice timing resolution
2. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining
} -- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. READ SR = Subscribed_Recipients
5. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR1 -- Time Remaining where $(TR-1) \leq TR1 \leq TR$
}) -- One list element

7.3.2.30.8.4 Resubscription Update Test

Purpose: Verify that a Subscribed_Recipients resubscription replaces Time Remaining and Confirmed Notification values.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder. Then renew the subscription with a different Time Remaining value and Confirmed Notification value and check that the Time Remaining and Confirmed Notification values have been replaced and no additional list entries have been made.

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

Test Steps:

1. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF1, -- Issue Confirmed
Notifications < DEST_CONF_NOTIF
TR2 -- Time Remaining where $TR2 > \text{test duration and } TR2 < TR$
} -- One list element

2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed_Recipients
4. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
DEST_CONF_NOTIF1, -- Issue Confirmed Notifications
TR3 -- Time Remaining where (TR2-1) <= TR3 <= TR2
}) -- One list element

7.3.2.30.8.5 Delete Test

Purpose: Verify RemoveListElement can delete a Subscribed_Recipients list entry.

Test Concept: Add a Subscribed_Recipients subscription to a Notification Forwarder. Then delete the subscription and check that the subscription has been removed.

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

Test Steps:

1. TRANSMIT RemoveListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID -- Process Identifier
} -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY Subscribed_Recipients = { } -- Empty list

7.3.2.30.8.6 Subscription Of Similar Entries Test

Purpose: Verify multiple Subscribed_Recipients subscriptions with minimal differences can be added as separate entries.

Test Concept: Add a Subscribed_Recipients subscription, then add it again with a different Recipient, then add it again with a different Process Identifier. Verify that the Subscribed_Recipients property list size is three and that all three subscriptions were added.

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

Test Steps:

1. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID2, -- Recipient D2
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration
} -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT AddListElement-Request,
'Object Identifier' = (the object being tested),
'Property Identifier' = Subscribed_Recipients,
'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
B, -- Process Identifier where B <> DEST_PROCESS_ID
FALSE, -- Issue Confirmed Notifications
TR -- Time Remaining where TR > test duration
} -- One list element
4. RECEIVE BACnet-SimpleACK-PDU
5. READ SR = Subscribed_Recipients
6. CHECK (SR = {{DEST_OBJ_ID, -- Recipient D1
DEST_PROCESS_ID, -- Process Identifier
FALSE, -- Issue Confirmed Notifications

TR1	-- Time Remaining where TR1 <= TR
},	-- List element in any order
{DEST_OBJ_ID2,	-- Recipient D2
DEST_PROCESS_ID,	-- Process Identifier
FALSE,	-- Issue Confirmed Notifications
TR2	-- Time Remaining where TR2 <= TR
},	-- List element in any order
{DEST_OBJ_ID,	-- Recipient D1
B,	-- Process Identifier
FALSE,	-- Issue Confirmed Notifications
TR3	-- Time Remaining where TR3 <= TR
}	-- List element in any order
})	-- Three list elements in any order

7.3.2.30.9 Process_Identifier_Filter Property Test

7.3.2.30.9.1 NULL And Unsigned32 Choice Test

Purpose: Insure writable Process_Identifier_Filter properties allow a choice of both NULL and Unsigned32 values.

Test Concept: Write a NULL Process_Identifier_Filter property value and verify that it was saved. Then write an Unsigned32 value = X Process_Identifier_Filter property value and verify that it was saved where X is any non-null legal process identifier value.

If no Notification Forwarder object in the IUT has a writable Process_Identifier_Filter property, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. WRITE Process_Identifier_Filter = NULL
2. VERIFY Process_Identifier_Filter = NULL
3. WRITE Process_Identifier_Filter = X
4. VERIFY Process_Identifier_Filter = X

7.3.2.30.9.2 NULL Unfiltered Process Identifier Test

Purpose: Insure a NULL value Process_Identifier_Filter property allows forwarding of all event notification process identifier values.

Test Concept: Perform tests 7.3.2.31.2 Recipient_List Forwarding Test and 7.3.2.31.3 Subscribed_Recipients Forwarding Test for non-zero values of SRC_PROCESS_ID. Select a non-zero value of SRC_PROCESS_ID within the BACnet allowable range and perform each test listed.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made NULL, then omit this test.

7.3.2.30.9.3 Zero Unfiltered Process Identifier Test

Purpose: Insure a zero value Process_Identifier_Filter property allows forwarding of all event notification process identifier values.

Test Concept: Define a Subscribed_Recipient entry and set the Process_Identifier_Filter to zero. Then send an event notification with a non-zero process identifier value to the notification forwarder. Verify that forwarding of the event notification is performed.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made zero, then omit this test.

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

Test Steps:

1. MAKE (Process_Identifier_Filter = zero)
2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV,	
'Event Object Identifier'	= SRC_NOTIF_OBJ,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	-- choose ALARM or EVENT
'AckRequired'	= (any valid value),	
'From State'	= (any valid From_State),	
'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	
3. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)

7.3.2.30.9.4 Specific Value Process Identifier Test

Purpose: Insure a non-NULL and non-zero value Process_Identifier_Filter property allows forwarding of only event notifications with process identifier values matching the Process_Identifier_Filter value.

Test Concept: Define a Subscribed_Recipient entry and set the Process_Identifier_Filter to a non-zero value. Then send an event notification with a non-zero process identifier value = X to the notification forwarder. Verify that forwarding of the event notification is only performed if the IUT Process_Identifier_Filter value matches the event notification Process_Identifier value.

If no Notification Forwarder object in the IUT has a Process_Identifier_Filter value that can be made non-NULL and non-zero, then omit this test.

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

Test Steps:

1. MAKE (Process_Identifier_Filter = X where X is not NULL and not zero)
2. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV,	
'Event Object Identifier'	= SRC_NOTIF_OBJ,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	-- choose ALARM or EVENT
'AckRequired'	= (any valid value),	
'From State'	= (any valid From_State),	
'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	
3. IF (SRC_PROCESS_ID = X) THEN
 BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)

ELSE

WAIT **Notification Fail Time**

CHECK (the IUT did not transmit an event notification)

7.3.2.30.9.5 Fixed Process_Identifier_Filter Test

Purpose: Test required configurability of property Process_Identifier_Filter.

Test Concept: If the Process_Identifier_Filter is not configurable or writable, check that at least one Notification Forwarder instance in the device under test has a NULL or zero Process_Identifier_Filter property value.

If Process_Identifier_Filter is configurable or writable, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. RESULT = 0
2. REPEAT ObjectX = (use each Notification Forwarder object in the IUT) DO
 - {READ PID = ObjectX, Process_Identifier_Filter
 - IF (PID = zero or NULL) THEN
 - RESULT = 1
 - }
3. CHECK (RESULT = 1)

7.3.2.30.10 Port_Filter Test

Purpose: Test that Notification Forwarder object Port_Filter properties in routers are writable and that event notifications are only forwarded if the Port_Filter property is set to allow the receiving port to forward the event.

Test Concept: Port P has a Port_ID value P_ID and is the router port where an event notification can be received by the IUT when sent from DS. Define D1 as a recipient in both the Recipient_List and the Subscribed_Recipients properties. D1 can be reachable through any port on the IUT device. Read the existing Port_Filter attribute and its array size and search the attribute array Port_ID entries for the entry matching P_ID indicating this is the correct port for a source event notification to reach the IUT from DS. Enable all ports except this one and send an event notification from DS to be forwarded. The IUT should not forward the event since the receiving port is disabled. Disable all ports except port P with Port_ID value P_ID and send an event notification to be forwarded. The IUT should now forward the event since the receiving port is enabled.

If the IUT device is not a router, then omit this test. If the IUT device cannot contain a Notification Forwarder object with a Local_Forwarding_Only property that has a value of FALSE, then omit this test.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. MAKE (Recipient_List =
 - {(all), -- Valid Days
 - (all), -- From Time, To Time
 - DEST_OBJ_ID, -- Recipient D1
 - DEST_PROCESS_ID, -- Process Identifier
 - FALSE, -- Issue Confirmed Notifications
 - {T, T, T} -- Transitions
 - }) -- One list element
3. TRANSMIT AddListElement-Request,
 - 'Object Identifier' = (the object being tested),
 - 'Property Identifier' = Subscribed_Recipients,
 - 'List of Elements' =
 - {DEST_OBJ_ID, -- Recipient D1
 - DEST_PROCESS_ID, -- Process Identifier
 - FALSE, -- Issue Confirmed Notifications
 - TR -- Time Remaining where TR > test duration
 - } -- One list element

4. RECEIVE BACnet-SimpleACK-PDU
 5. READ PF1 = Port_Filter
 6. READ PF1_SIZE = Port_Filter, ARRAY INDEX = 0
 7. WHILE (PF1 index X Port_ID $\neq$ P_ID) DO
 (Next index X)
 8. WHILE (Z = 1 to PF1_SIZE) DO
 IF (Z $\neq$ X) THEN
 WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled TRUE), ARRAY INDEX = Z
 ELSE
 WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled FALSE), ARRAY INDEX = Z
 9. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 10. WAIT **Notification Fail Time**
 11. CHECK (the IUT did not transmit an event notification)
 12. WHILE (Z = 1 to PF1_SIZE) DO
 IF (Z $\neq$ X) THEN
 WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled FALSE), ARRAY INDEX = Z
 ELSE
 WRITE Port_Filter = (Port_ID from PF1 index Z, Enabled TRUE), ARRAY INDEX = Z
 13. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 14. BEFORE **Notification Fail Time**
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
 'Process Identifier' = DEST_PROCESS_ID,
 (other parameter values match the parameter values used in the previous step)

7.3.2.30.11 Local_Forwarding_Only Property Tests**7.3.2.30.11.1 Only Forwards Locally When True**

Purpose: Verify a Notification Forwarder object with Local_Forwarding_Only set TRUE will only forward locally generated events.

Test Concept: Set the Local_Forwarding_Only property in the IUT to TRUE and add a Notification Forwarder subscription with D1 as the recipient. Send an Event Notification from DS to the IUT and check that the event is not forwarded. Send a different Event Notification from a source within the IUT and check that the event was forwarded.

If the IUT device cannot contain a Notification Forwarder object with a Local_Forwarding_Only property that has a value of TRUE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Notification source object in DS configured as follows:

Source notification device = SRC_NOTIF_DEV1,
Source notification object = SRC_NOTIF_OBJ1,
Notification_Class = SRC_NOTIF_CLS1
Event_Enable = {T, T, T}

Notification source object in IUT configured as follows:

Source notification device = IUT,
Source notification object = SRC_NOTIF_OBJ2,
Notification_Class = SRC_NOTIF_CLS2
Event_Enable = {T, T, T}

Source notification class object SRC_NOTIF_CLS1 in DS configured as follows:

Notification Class = (instance number of notification class object),
Priority = (any valid priority)
Ack_Required = (any valid Ack_Required value)
Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
IUT -- Recipient
} --One list element
Process Identifier = SRC_PROCESS_ID
Issue Confirmed Notifications = FALSE
Transitions = {T, T, T}

Source notification class object SRC_NOTIF_CLS2 in IUT configured as follows:

Notification Class = (instance number of notification class object),
Priority = (any valid priority)
Ack_Required = (any valid Ack_Required value)
Recipient_List = {(all), -- Valid Days
(all), -- From Time, To Time
IUT -- Recipient
} -- One list element
Process Identifier = SRC_PROCESS_ID
Issue Confirmed Notifications = FALSE
Transitions = {T, T, T}

Note to Tester: Issue Confirmed Notifications is specified as FALSE within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected. Behaviors can alternately be tested using Issue Confirmed Notifications set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Test Steps:

1. MAKE (Local_Forwarding_Only = TRUE)
2. TRANSMIT AddListElement-Request,

'Object Identifier' =	(the object being tested),	
'Property Identifier' =	Subscribed_Recipients,	
'List of Elements' =	{DEST_OBJ_ID,	-- Recipient D1
	DEST_PROCESS_ID,	-- Process Identifier
	FALSE,	-- Issue Confirmed Notifications
	TR	-- Time Remaining where TR > test duration
	}	-- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV1,	-- DS
'Event Object Identifier'	= SRC_NOTIF_OBJ1,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS1,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	
'AckRequired'	= (any valid value),	-- absent if Notify Type is
ACK_NOTIFICATION		
'From State'	= (any valid From_State),	-- absent if Notify Type is
ACK_NOTIFICATION		
'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	-- absent if Notify Type is
ACK_NOTIFICATION		
5. WAIT **Notification Fail Time**
6. CHECK (the IUT did not transmit an event notification)
7. MAKE (IUT Notification source generate an UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV2,	--IUT
'Event Object Identifier'	= SRC_NOTIF_OBJ2,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS2,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	
'AckRequired'	= (any valid value),	-- absent if Notify Type is
ACK_NOTIFICATION		
'From State'	= (any valid From_State),	-- absent if Notify Type is
ACK_NOTIFICATION		
'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	-- absent if Notify Type is
ACK_NOTIFICATION		
8. BEFORE **Notification Fail Time**

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request
'Process Identifier' = DEST_PROCESS_ID,
(other parameter values match the parameter values used in the previous step)

7.3.2.30.11.2 Forwards Locally And Remotely When False

Purpose: Verify a Notification Forwarder object with Local_Forwarding_Only set FALSE will forward both locally generated events and externally generated events.

Test Concept: Set the Local_Forwarding_Only property in the IUT to FALSE and add a Notification Forwarder subscription with D1 as the recipient. Send an Event Notification from DS to the IUT and check that the event is forwarded. Send a different Event Notification from a source within the IUT and check that the event was forwarded.

If the IUT device cannot contain a Notification Forwarder object with either a writable Local_Forwarding_Only property, or a Local_Forwarding_Only property that has a value of FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Notification source object in DS configured as follows:

Source notification device	= SRC_NOTIF_DEV1,
Source notification object	= SRC_NOTIF_OBJ1,
Notification_Class	= SRC_NOTIF_CLS1
Event_Enable	= {T, T, T}

Notification source object in IUT configured as follows:

Source notification device	= IUT,
Source notification object	= SRC_NOTIF_OBJ2,
Notification_Class	= SRC_NOTIF_CLS2
Event_Enable	= {T, T, T}

Source notification class object SRC_NOTIF_CLS1 in DS configured as follows:

Notification Class	= (instance number of notification class object),
Priority	= (any valid priority)
Ack_Required	= (any valid Ack_Required value)
Recipient_List	= {(all), -- Valid Days (all), -- From Time, To Time IUT -- Recipient } -- One list element
Process Identifier	= SRC_PROCESS_ID
Issue Confirmed Notifications	= FALSE
Transitions	= {T, T, T}

Source notification class object SRC_NOTIF_CLS2 in IUT configured as follows:

Notification Class	= (instance number of notification class object),
Priority	= (any valid priority)
Ack_Required	= (any valid Ack_Required value)
Recipient_List	= {(all), -- Valid Days (all), -- From Time, To Time IUT -- Recipient } -- One list element
Process Identifier	= SRC_PROCESS_ID
Issue Confirmed Notifications	= FALSE
Transitions	= {T, T, T}

Note to Tester: Issue Confirmed Notifications is specified as FALSE within the test. Thus UnconfirmedEventNotification-Request messages from the Notification source object are expected. Behaviors can alternately be tested using Issue Confirmed Notifications set to TRUE resulting in ConfirmedEventNotification service messages from the Notification source object that must be acknowledged at the communication level, but it is not necessary to test both settings unless specifically directed otherwise by the test.

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,

'Object Identifier' =	(the object being tested),
'Property Identifier' =	Subscribed_Recipients,
'List of Elements' =	{DEST_OBJ_ID, -- Recipient D1 DEST_PROCESS_ID, -- Process Identifier FALSE, -- Issue Confirmed Notifications TR -- Time Remaining where TR > test duration } -- One list element

3. RECEIVE BACnet-SimpleACK-PDU

4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV1, -- DS
 'Event Object Identifier' = SRC_NOTIF_OBJ1,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS1,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),

'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

5. BEFORE Notification Fail Time

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request

'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)

6. MAKE (IUT Notification source object send an UnconfirmedEventNotification-Request,

'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV2, --IUT
 'Event Object Identifier' = SRC_NOTIF_OBJ2,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS2,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is

ACK_NOTIFICATION

'From State' = (any valid From_State), -- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),

'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION)

7. BEFORE Notification Fail Time

RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request

'Process Identifier' = DEST_PROCESS_ID,

(other parameter values match the parameter values used in the previous step)

7.3.2.30.12 Preventing endless cycling / duplication of event forwarding for the same notification

7.3.2.30.12.1 Local Broadcast To Receiving Port Restriction Test

Purpose: Check that locally broadcast event forwarding is not allowed at the same port the source event notification is received at.

Test Concept: Network number NN is the BACnet network number the IUT is connected to at IUT device port PT. Set Local_Forwarding_Only to FALSE in the IUT. Set a recipient in the Subscribed_Recipients property and the Recipient_List property to the BACnetAddress for a local broadcast to network number NN. Configure the notification source to send an event notification to the IUT device at network NN. Monitoring the IUT shall be done at network number NN. Transmit the source event notification directly to the IUT device at network number NN. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Notification source object is in DS.

Source notification class is in DS.

Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

```
Recipient_List = { (all),           -- Valid Days
                  (all),           -- From Time, To Time
                  (IUT at network NN) -- Recipient
                  }               -- One list element
```

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,

'Object Identifier' =	(the object being tested),	
'Property Identifier' =	Subscribed_Recipients,	
'List of Elements' =	{(BACnetAddress; network-number 00, mac-address length 0),	--Recipient
	DEST_PROCESS_ID,	--Process Identifier
	FALSE,	--Issue Confirmed Notifications
	TR	--Time Remaining where TR > test duration
	}	--One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. MAKE (Recipient_List =

{(all),	-- Valid Days	
(all),	-- From Time, To Time	
(BACnetAddress; network-number NN, mac-address length 0),		--Recipient
DEST_PROCESS_ID,	-- Process Identifier	
FALSE,	-- Issue Confirmed Notifications	
{T, T, T}	-- Transitions	
})	--One list element	
5. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV,	
'Event Object Identifier'	= SRC_NOTIF_OBJ,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	
'AckRequired'	= (any valid value),	-- absent if Notify Type is
- ACK_NOTIFICATION

'From State'	= (any valid From_State),	-- absent if Notify Type is
- ACK_NOTIFICATION

'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	-- absent if Notify Type is
- ACK_NOTIFICATION
6. WAIT Notification Fail Time
7. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.2 Globally Broadcast Event Notification Received Restriction Test

Purpose: Verify global broadcast event notifications are not forwarded.

Test Concept: Set D1 as a recipient in the Subscribed_Recipients property. Send a globally broadcast event notification to the IUT and check that it is not forwarded.

Configuration Requirements:

Begin with base setup 2 for Notification Forwarder object tests with TR lifetime sufficient for this test.

Perform the following additional steps:

Reconfigure the source notification class SRC_NOTIF_CLS in DS (or IUT if Local_Forwarding_Only is TRUE) as follows:

```

Recipient_List = { (all), --Valid Days
                  (all), --From Time, To Time
                  (BACnetAddress; network-number 65535, mac-address length 0) --Recipient = global
broadcast
                  } --One list element

```

Test Steps:

- TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,				
'Initiating Device Identifier'	= SRC_NOTIF_DEV,				
'Event Object Identifier'	= SRC_NOTIF_OBJ,				
'Time Stamp'	= (any valid time stamp),				
'Notification Class'	= SRC_NOTIF_CLS,				
'Priority'	= (any valid priority),				
'Event Type'	= (any valid event type),				
'Message Text'	= (optional, any valid message text),				
'Notify Type'	= SRC_NOTIF_TYP,				
'AckRequired'	= (any valid value),	--	absent	if	Notify Type is
ACK_NOTIFICATION					
'From State'	= (any valid From_State),	--	absent	if	Notify Type is
ACK_NOTIFICATION					
'To State'	= (any valid To_State),				
'Event Values'	= (any valid event values)	--	absent	if	Notify Type is
ACK_NOTIFICATION					
- WAIT **Notification Fail Time**
- CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.3 Forwarding As Global Broadcast Restriction Test

Purpose: Verify forwarded event notifications cannot be globally broadcast by the IUT.

Test Concept: Attempt to configure an IUT Subscribed Recipients value to forward an event as a global broadcast. Check that the configuration is either not accepted by the IUT or does not send a global broadcast when so configured. Attempt to configure an IUT Recipient_List value to forward an event as a global broadcast. Check that the configuration is either not accepted by the IUT or does not send a global broadcast when so configured.

Configuration Requirements:

Base setup 1 for Notification Forwarder object tests.

Test Steps:

- TRANSMIT AddListElement-Request,

'Object Identifier' =	(the object being tested),
'Property Identifier' =	Subscribed_Recipients,
'List of Elements' =	{(BACnetAddress; network-number 65535, mac-address length 0), --Recipient =
global broadcast	
	DEST_PROCESS_ID, -- Process Identifier
	FALSE, -- Issue Confirmed Notifications
	TR -- Time Remaining where TR > test duration
	} -- One list element
- IF (RECEIVE BACnet-SimpleACK-PDU) THEN

TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
'Process Identifier' = SRC_PROCESS_ID,
'Initiating Device Identifier' = SRC_NOTIF_DEV,
'Event Object Identifier' = SRC_NOTIF_OBJ,
'Time Stamp' = (any valid time stamp),
'Notification Class' = SRC_NOTIF_CLS,
'Priority' = (any valid priority),
'Event Type' = (any valid event type),
'Message Text' = (optional, any valid message text),

'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value) -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State) -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 3. WAIT Notification Fail Time
 4. CHECK (the IUT did not transmit an event notification)
 5. If (The IUT can contain a global broadcast address in the Recipient_List) THEN
 MAKE (Recipient_List = {(all), --Valid Days
 (all), --From Time, To Time
 (BACnetAddress; network-number 65535, mac-address length 0), --Recipient = global
 broadcast
 DEST_PROCESS_ID, -- Process Identifier
 FALSE, -- Issue Confirmed Notifications
 {T, T, T} -- Transitions
 } -- One list element
 TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION
 'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is
 ACK_NOTIFICATION
 WAIT Notification Fail Time
 CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.4 Directed Broadcast Received Forwarding To BACnetAddress Restriction Test

Purpose: Check that a directed broadcast event notification received at a particular port is not forwarded to an external device identified as a BACnetRecipient using the BACnetAddress choice when it is residing on the same network that the port is attached to.

Test Concept: Network number NN is the BACnet network number the IUT is connected to. Configure the notification source to send an event notification as a directed broadcast to Network number NN. Set Local_Forwarding_Only to FALSE, and set an IUT recipient in the Subscribed_Recipients property to the BACnetAddress of a device connected to network number NN. Monitoring the IUT shall be done at network number NN. Transmit the source event notification as a directed broadcast to network number NN. The event notification source may be on a local or remote network. Check that the IUT did not forward the event notification.

Set an IUT recipient in the Recipient_List property to the BACnetAddress of a device connected to network number NN. Transmit the source event notification as a directed broadcast to network number NN. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:
 Notification source object must be in DS.

Source notification class must be in DS.

Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

```
Recipient_List = { (all),          -- Valid Days
                  (all),          -- From Time, To Time
                  (BACnetAddress; network-number NN, mac-address length 0)
                  }
                  -- Recipient is directed broadcast
                  --One list element
```

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,

'Object Identifier' =	(the object being tested),	
'Property Identifier' =	Subscribed_Recipients,	
'List of Elements' =	{(BACnetAddress; network-number NN, mac-address MA),	--Recipient
	DEST_PROCESS_ID,	-- Process Identifier
	FALSE,	-- Issue Confirmed Notifications
	TR	-- Time Remaining where TR > test duration
	}	-- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV,	
'Event Object Identifier'	= SRC_NOTIF_OBJ,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	
'AckRequired'	= (any valid value),	-- absent if Notify Type is

ACK_NOTIFICATION

'From State'	= (any valid From_State),	-- absent if Notify Type is

ACK_NOTIFICATION

'To State'	= (any valid To_State),	
'Event Values'	= (any valid event values)	-- absent if Notify Type is

ACK_NOTIFICATION
5. WAIT **Notification Fail Time**
6. CHECK (the IUT did not transmit an event notification)
7. MAKE (Recipient_List = {(all), --Valid Days

(all),	-- From Time, To Time
(BACnetAddress; network-number NN, mac-address MA),	-- Recipient
DEST_PROCESS_ID,	-- Process Identifier
FALSE,	-- Issue Confirmed Notifications
{T, T, T}	-- Transitions
}	-- One list element
8. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,

'Process Identifier'	= SRC_PROCESS_ID,	
'Initiating Device Identifier'	= SRC_NOTIF_DEV,	
'Event Object Identifier'	= SRC_NOTIF_OBJ,	
'Time Stamp'	= (any valid time stamp),	
'Notification Class'	= SRC_NOTIF_CLS,	
'Priority'	= (any valid priority),	
'Event Type'	= (any valid event type),	
'Message Text'	= (optional, any valid message text),	
'Notify Type'	= SRC_NOTIF_TYP,	
'AckRequired'	= (any valid value),	-- absent if Notify Type is

ACK_NOTIFICATION

'From State'	= (any valid From_State),	-- absent if Notify Type is

ACK_NOTIFICATION

'To State' = (any valid To_State),
 'Event Values' = (any valid event values) -- absent if Notify Type is

ACK_NOTIFICATION

9. WAIT Notification Fail Time

10. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.5 Directed Broadcast Received Forwarding To Object Identifier Restriction Test

Purpose: Check that a directed broadcast event notification received at a particular port is not forwarded to an external device identified as a BACnetRecipient using the BACnetObjectIdentifier choice when it is residing on the same network that the port is attached to.

Test Concept: Network number NN is the BACnet network number the IUT is connected to. Configure the notification source DS to send an event notification as a directed broadcast to Network number NN. Set Local_Forwarding_Only to FALSE, and set recipient D1 in the IUT Subscribed_Recipients property to the BACnetObjectIdentifier of a device connected to network number NN. This device may be the TD or a reference device, but it must be capable of being discovered by the IUT device. Monitoring the IUT shall be done at network number NN. Transmit the source event notification as a directed broadcast. The event notification source may be on a local or remote network. Check that the IUT did not forward the event notification. Set a recipient in the IUT Recipient_List property to the BACnetObjectIdentifier of a device connected to network number NN. Transmit the source event notification as a directed broadcast. Check that the IUT did not forward the event notification.

If the IUT Notification Forwarder object Local_Forwarding_Only property value cannot be set to FALSE, then omit this test.

Configuration Requirements:

Begin with base setup 1 for Notification Forwarder object tests, perform the following additional steps:

Reconfigure the source notification class SRC_NOTIF_CLS in DS as follows:

```
Recipient_List = {(all),          -- Valid Days
                  (all),          -- From Time, To Time
                  (BACnetAddress; network-number NN (-- 0 if local broadcast), mac-address length 0)
                  }              -- Recipient is local broadcast
                              -- One list element
```

Test Steps:

1. MAKE (Local_Forwarding_Only = FALSE)
2. TRANSMIT AddListElement-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Subscribed_Recipients,
 'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 FALSE, -- Issue Confirmed Notifications
 TR -- Time Remaining where TR > test duration
 } -- One list element
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,
 'Process Identifier' = SRC_PROCESS_ID,
 'Initiating Device Identifier' = SRC_NOTIF_DEV,
 'Event Object Identifier' = SRC_NOTIF_OBJ,
 'Time Stamp' = (any valid time stamp),
 'Notification Class' = SRC_NOTIF_CLS,
 'Priority' = (any valid priority),
 'Event Type' = (any valid event type),
 'Message Text' = (optional, any valid message text),
 'Notify Type' = SRC_NOTIF_TYP,
 'AckRequired' = (any valid value), -- absent if Notify Type is
 ACK_NOTIFICATION
 'From State' = (any valid From_State), -- absent if Notify Type is
 ACK_NOTIFICATION

'To State'	= (any valid To_State),								
'Event Values'	= (any valid event values) --	absent	if	Notify	Type	is			
ACK_NOTIFICATION									
5. WAIT Notification Fail Time									
6. CHECK (the IUT did not transmit an event notification)									
7. MAKE (Recipient_List =	{(all),	-- Valid Days							
	(all),	-- From Time, To Time							
	DEST_OBJ_ID,	-- Recipient D1							
	DEST_PROCESS_ID,	-- Process Identifier							
	FALSE,	-- Issue Confirmed Notifications							
	{T, T, T}	-- Transitions							
	})	-- One list element							
8. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,									
'Process Identifier'	= SRC_PROCESS_ID,								
'Initiating Device Identifier'	= SRC_NOTIF_DEV,								
'Event Object Identifier'	= SRC_NOTIF_OBJ,								
'Time Stamp'	= (any valid time stamp),								
'Notification Class'	= SRC_NOTIF_CLS,								
'Priority'	= (any valid priority),								
'Event Type'	= (any valid event type),								
'Message Text'	= (optional, any valid message text),								
'Notify Type'	= SRC_NOTIF_TYP,								
'AckRequired'	= (any valid value),	--	absent	if	Notify	Type	is		
ACK_NOTIFICATION									
'From State'	= (any valid From_State),	--	absent	if	Notify	Type	is		
ACK_NOTIFICATION									
'To State'	= (any valid To_State),								
'Event Values'	= (any valid event values) --	absent	if	Notify	Type	is			
ACK_NOTIFICATION									

9. WAIT Notification Fail Time

10. CHECK (the IUT did not transmit an event notification)

7.3.2.30.12.6 Port Restriction Test

Purpose: Check that event notifications received through ports that are not enabled are not forwarded.

Test Concept: Perform test 7.3.2.31.10 Port_Filter Test.

7.3.2.30.13 Persistence Tests

7.3.2.30.13.1 Recipient_List Persistence Test

Purpose: This test insures that Recipient_List property value is maintained through a device “restart”.

Test Concept: Initialize the Recipient_List property with a known value, then cycle power or restart the IUT and verify the Recipient_List property value is maintained.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

- MAKE (Recipient_List =

{(all),	-- Valid Days
(all),	-- From Time, To Time
DEST_OBJ_ID,	-- Recipient D1
DEST_PROCESS_ID,	-- Any Process Identifier
DEST_CONF_NOTIF,	-- Any Issue Confirmed Notifications
{T, T, T}	-- Transitions
})	-- One list element
- IF (IUT supports the ReinitializeDevice service) THEN {

TRANSMIT ReinitializeDevice-Request,	
'Reinitialized State of Device' = WARMSTART,	
'Password' =	(any valid password) --if required by IUT

- ```

 RECEIVE BACnet-SimpleACK-PDU
 }
3. CHECK (Did the IUT perform a WARMSTART restart)
4. WAIT for restart to complete
5. VERIFY Recipient_List = {(all), -- Valid Days
 (all), -- From Time, To Time
 DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 DEST_CONF_NOTIF, -- Issue Confirmed Notifications
 {T, T, T} -- Transitions
 } -- One list element
6. MAKE (The IUT reset by cycling power)
7. WAIT for restart to complete
8. VERIFY Recipient_List = {(all), -- Valid Days
 (all), -- From Time, To Time
 DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 DEST_CONF_NOTIF, -- Issue Confirmed Notifications
 {T, T, T} -- Transitions
 } -- One list element

```

### 7.3.2.30.13.2 Subscribed\_Recipients Persistence Test

Purpose: This test insures that Subscribed\_Recipients property values are maintained through a device “restart”.

Test Concept: Initialize the Subscribed\_Recipients property with a known value, then cycle power to restart the IUT and verify the Subscribed\_Recipients property value is maintained.

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

Test Steps:

- ```

1. IF (IUT supports the ReinitializeDevice service) THEN {
    TRANSMIT ReinitializeDevice-Request,
      'Reinitialized State of Device' = WARMSTART,
      'Password' = (any valid password)    --if required by IUT
    RECEIVE BACnet-SimpleACK-PDU
  }
2. CHECK (Did the IUT perform a WARMSTART restart)
3. WAIT for restart to complete, making a note of the time to restart as Start_Up_TimeA
4. VERIFY Subscribed_Recipients =
  {DEST_OBJ_ID,      -- Recipient D1
   DEST_PROCESS_ID,  -- Process Identifier
   DEST_CONF_NOTIF,  -- Issue Confirmed Notifications
   TR1              -- Time Remaining where (TR-Start_Up_TimeA-1) <= TR1 <= TR
  }                -- One list element
5. MAKE (The IUT reset by cycling power)
6. WAIT for restart to complete, making a note of the time to restart as Start_Up_TimeB
7. VERIFY Subscribed_Recipients =
  {DEST_OBJ_ID,      -- Recipient D1
   DEST_PROCESS_ID,  -- Process Identifier
   DEST_CONF_NOTIF,  -- Issue Confirmed Notifications
   TR1              -- Time Remaining where (TR-Start_Up_TimeA-Start_Up_TimeB-2) <= TR1 <=
TR
  }                -- One list element

```

Note To Tester: Start_Up_TimeA or Start_Up_TimeB is the time in minutes required to restart the IUT.

7.3.2.30.14 Capacity And Range Tests**7.3.2.30.14.1 Time Remaining Range Test**

Purpose: Test that the Subscribed_Recipients property can be written with the range of Time Remaining values required.

Test Concept: Add a BACnetEventNotificationSubscription to the Notification Forwarder Subscribed_Recipients property with the largest Time Remaining value required. Verify the new BACnetEventNotificationSubscription entry is added.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. TRANSMIT AddListElement-Request,
 - 'Object Identifier' = (the object being tested),
 - 'Property Identifier' = Subscribed_Recipients,
 - 'List of Elements' = {DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 DEST_CONF_NOTIF, -- Issue Confirmed Notifications
 1440 minutes -- Time Remaining
 } -- One list element
2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed_Recipients
4. CHECK (SR = {DEST_OBJ_ID, -- Recipient D1
 DEST_PROCESS_ID, -- Process Identifier
 DEST_CONF_NOTIF, -- Issue Confirmed Notifications
 TR1 -- Time Remaining where 1439 <= TR1 <= 1440
 }) -- One list element

7.3.2.30.14.2 Recipient Capacity Test

Purpose: Test that the capacity of property Recipient_List and property Subscribed_Recipients is sufficient.

Test Concept: Add entries to the Recipient_List and Subscribed_Recipients properties to verify at least 8 of each can be added at the same time. Use different Process Identifier values and recipients for each list element to ensure each one is added separately.

Configuration Requirements: Base setup 1 for Notification Forwarder object tests.

Test Steps:

1. MAKE (Recipient_List = {--list with 8 elements
 {--for each element
 (all), -- Valid Days
 (all), -- From Time, To Time
 DEST_OBJ_IDX, -- One of 8 Recipients DX where X = 1 to 8
 (DEST_PROCESS_ID + X), -- One of 8 Process Identifiers where X = 1 to 8
 DEST_CONF_NOTIF, -- Any Issue Confirmed Notifications
 {T, T, T} -- Transitions
 }
 { ... } --Seven additional list elements as above
 }
 })
2. REPEAT X = (values from 1 to 8) DO
 {TRANSMIT AddListElement-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Subscribed_Recipients,
 'List of Elements' = {DEST_OBJ_ID2X, -- One of 8 Recipients DX where X = 1 to 8
 (DEST_PROCESS_ID2 + X), -- One of 8 Process Identifiers where X = 1 to 8
 DEST_CONF_NOTIF, -- Any Issue Confirmed Notifications
 TR -- Any Time Remaining where TR > test duration
 } -- One list element each time
 }

RECEIVE BACnet-SimpleACK-PDU

}

3. VERIFY Subscribed_Recipients = { ... } -- List with 8 elements as configured above, except for Time Remaining
4. VERIFY Recipient_List = { ... } -- List with 8 elements as configured above

7.3.2.31 Alert Enrollment Tests

7.3.2.31.1 Alert Enrollment Reports The Source Object

Purpose: To verify that the Alert Enrollment object's notifications provide an object identifier as the first parameter.

Test Concept: Select an Alert Enrollment object, O1, and cause it to generate a notification. Verify that the notification uses the extended notification parameters and that the first parameter is an object identifier.

Configuration Requirements: O1 is configured to issue unconfirmed event notifications. O2, referenced in this test, is another configured object. D1 is a vendor specific delay (may be zero).

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (a condition exist which will cause O1 to transition)
3. WAIT D1
4. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid timestamp),

'Priority' = (any valid priority),

'Event Type' = EXTENDED,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = FALSE,

'From State' = NORMAL,

'To State' = NORMAL,

'Event Values' = (any valid vendor id),
 (any valid extended event type),
 (O2: an object identifier),
 (any valid list of parameters)
5. CHECK (that O2 exists in the IUT)

7.3.2.31.2 Alert Enrollment Does Not Generate Acknowledgeable Transitions

Purpose: To verify that the Alert Enrollment object's notifications are not acknowledgeable.

Test Concept: Select an Alert Enrollment object, O1, and cause it to generate a notification. Verify that the notification does not require an acknowledgment.

Configuration Requirements: O1 is configured to issue unconfirmed event notifications. O2, referenced in this test, is another configured object. D1 is a vendor specific delay (may be zero).

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (a condition exist which will cause O1 to transition)
3. WAIT D1
4. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid timestamp),

'Priority' = (any valid priority),

'Event Type' = EXTENDED,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = FALSE,

'From State' = NORMAL,

'To State' = NORMAL,

'Event Values' = (any valid vendor id),
 (any valid extended event type),
 (O2: an object identifier),
 (any valid list of parameters)

5. VERIFY Acked_Transitions = (T, T, T)

7.3.2.32 Accumulator Object Tests

7.3.2.32.1 Present_Value Remains In-Range Test

Purpose: To verify the correct wrapping operation of the Accumulator Present_Value.

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

Test Steps:

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

7.3.2.32.2 Prescale in Accumulator Test

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

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

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

Test Steps:

1. IF (Value_Set is writable) THEN
 WRITE Value_Set = (any valid value V1)
 ELSE
 MAKE (Present_Value equal any valid value V1)
2. MAKE (the Accumulator increase its Present_Value)
3. READ V2 = Present_Value)
4. READ V3 = Present_Value)
5. IF (the Accumulator is stopped) THEN
 CHECK (V3 - V2 = ((Prescale-multiplier) * pulse-count of signals generated by the measuring instrument) /
 (Prescale-moduloDivide) + V1)
 ELSE
 CHECK (V1 < V2 < V3)

7.3.2.32.3 Logging_Record in Accumulator Test

Purpose: To verify the correct values represented in Logging_Record of Accumulator.

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

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

Test Steps:

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

7.3.2.32.4 Logging_Record in Accumulator RECOVERED Test

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

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

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

Test Steps:

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

7.3.2.32.5 Logging_Record in Accumulator STARTING Test

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

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

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

Test Steps:

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

7.3.2.32.6 Out_Of_Service Accumulator Test

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

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

Test Steps:

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

7.3.2.32.7 Value_Set Writing Test

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

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

Test Steps:

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

7.3.2.32.8 Value_Before_Change Writing Test

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

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

Test Steps:

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

7.3.2.33 Pulse Converter Object Tests**7.3.2.33.1 Adjust_Value Write Test**

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

Configuration Requirements: Select a Pulse Converter object for which the pulse can be stopped so that Count remains unchanged during the test.

Test Steps:

1. READ OldC = Count
2. WRITE Adjust_Value = (NewA, any valid value)
3. VERIFY Present_Value = Count * Scale_Factor
4. VERIFY Count_Change_Time ≈ (the current local time)
5. VERIFY Count_Before_Change = OldC

7.3.2.33.2 Scale_Factor Test

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

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

Test Steps:

1. IF (Scale_Factor is writable) THEN
 WRITE Scale_Factor = (any valid value V1)
ELSE
 MAKE (Scale_Factor equal any valid value V1)
2. VERIFY (Present_Value = conversion specified by Scale_Factor V1 coefficient times the Count property)

7.3.2.33.3 Out_Of_Service Pulse Converter Test

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

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

Test Steps:

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

- ```

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

```

#### 7.3.2.33.4 Update\_Time Reflects Change to the Count and is Updated Atomically Test

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

Test Steps:

1. READ OldV = Present\_Value
2. READ OldC = Count
3. READ OldU = Update\_Time
4. READ OldT = Count\_Change\_Time
5. READ OldA = Adjust\_Value
6. READ OldS = Scale\_Factor
7. READ OldB = Count\_Before\_Change
8. WAIT (for a change in Count to any valid value, different from OldC so that it can be distinguished)
9. CHECK Present\_Value is recalculated, increasing in proportion to the change in Count multiplied by OldS (or such that Present\_Value minus OldA is still the same fixed difference)
10. VERIFY Update\_Time = (approximately the current local time, and different from OldU)
11. VERIFY Count\_Change\_Time = OldT

#### 7.3.2.33.5 Adjust\_Value Out-of-Range WriteProperty Test

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

Test Steps:

1. READ OldV = Present\_Value
2. READ OldC = Count
3. READ OldU = Update\_Time
4. READ OldT = Count\_Change\_Time
5. READ OldA = Adjust\_Value
6. READ OldS = Scale\_Factor
7. READ OldB = Count\_Before\_Change
8. TRANSMIT WriteProperty-Request  
'Property Identifier' = Adjust\_Value

- 'Property Value' = (NewA, a valid value that would cause an overflow or underflow condition in Count)
9. RECEIVE BACnet-Error-PDU  
     'Error Class' = PROPERTY  
     'Error Code' = VALUE\_OUT\_OF\_RANGE
  10. VERIFY Update\_Time = OldU
  11. VERIFY Adjust\_Value = OldA
  12. VERIFY Count\_Before\_Change = OldB

### 7.3.2.34 Channel Object Tests

#### 7.3.2.34.1 Last\_Priority Test

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

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

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

Test Steps:

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

#### 7.3.2.34.2 WriteGroup Service Support Test

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

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

Test Steps:

1. READ X = O1, Present\_Value
2. TRANSMIT WriteGroup-Request,  
     'Group Number' = (one of the Control\_Group values configured in O1),  
     'Write Priority' = (any valid value),  
     'Change List' = (O1's channel number, no overriding priority, Y: a value different than X)
3. VERIFY Present\_Value = Y

#### 7.3.2.34.3 Propagation Entirety Test

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

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

Configuration Requirements: Configure the Channel object's List\_Of\_Object\_Property\_References so that at least one of the target references (contained in entry X of List\_Of\_Object\_Property\_References) will result in an invalid datatype

coercion when the value WrittenValue is written to the Channel object. The rest of the target reference(s) shall be selected such that they will accept the written value as is without coercion. Refer to the Table 12-63 – Datatype Coercion Rules from ASHRAE 135 for the invalid datatypes.

Test Steps:

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

#### 7.3.2.34.4 Write\_Status Test

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

Test Concept: The Channel object's List\_Of\_Object\_Property\_References is read and verified to be empty. Then, the Channel object's Write\_Status is verified to be IDLE. Next, any valid value is written to the Channel object's Present\_Value and Write\_Status is verified to be IDLE still. An object property reference, R1, that the IUT cannot reach is set into the List\_Of\_Object\_Property\_References and then any valid value is written to the Present\_Value and the Write\_Status is verified to be IN\_PROGRESS. After the IUT determines that the referenced device is offline, the Write\_Status is verified to be FAILED. Finally, any valid reference, R2, that will cause successful value propagation is set to the List\_Of\_Object\_Property\_References and the test is repeated to verify that Write\_Status becomes SUCCESSFUL.

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- IDLE test
2. READ L = List\_Of\_Object\_Property\_References
3. VERIFY L = (empty)
4. VERIFY Write\_Status = IDLE
5. WRITE Present\_Value = (any valid value)
6. VERIFY Write\_Status = IDLE
- IN\_PROGRESS and FAIL test
7. WRITE List\_Of\_Object\_Property\_References = (R1) -- write the whole array
8. WRITE Present\_Value = (any valid value)
9. VERIFY Write\_Status = IN\_PROGRESS
10. WAIT (**Channel Write Fail Time** \* LEN)
11. VERIFY Write\_Status = FAILED
- SUCCESSFUL test
12. WRITE List\_Of\_Object\_Property\_References = (R2) -- write the whole array
13. WRITE Present\_Value = (any valid value)

14. WAIT (**Channel Write Fail Time** \* LEN)
15. VERIFY Write\_Status = SUCCESSFUL

#### 7.3.2.34.5 Allow\_Group\_Delay\_Inhibit Test

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

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

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

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

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Setup the Channel object
2. WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)
3. WRITE Execution\_Delay = (ED1, ED2)

-- Test that delays are inhibited

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

-- Test that delays are not inhibited

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

#### 7.3.2.34.6 Numeric to BOOLEAN Coercion Rule Test

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

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

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

Test Steps:

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

#### 7.3.2.34.7 BOOLEAN to Numeric Coercion Rule Test

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

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

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

Test Steps:

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

#### 7.3.2.34.8 Unsigned/INTEGER/REAL/Double to Numeric Coercion Rule Test

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

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

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

Test Steps:

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

**7.3.2.34.9 Invalid Datatype Coercion Test**

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

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

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

Test Steps:

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

**7.3.2.34.10 No Coercion Test**

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

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

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

Test Steps:

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

**7.3.2.34.11 Write Priority Test**

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

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

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

Test Steps:

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



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

### 7.3.2.34.12 Writing with a NULL Value Test

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

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

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

Test Steps:

1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0  
 -- Read an initial value of target properties
2. READ P1 = List\_Of\_Object\_Property\_References: ARRAY INDEX = X, which is a commandable property
3. READ P2 = List\_Of\_Object\_Property\_References: ARRAY INDEX = Y, which is a non-commandable property
- Find expected values of the target properties after a NULL value is written to them
4. READ V1 = P1.Relinquish\_Default



5. READ V2 = P2

-- Make the Channel object to propagate NULL value to targets

6. WRITE Present\_Value = NULL

7. WAIT (**Channel Write Fail Time** \* LEN)

8. IF (P2 is on external) THEN

RECEIVE WritePropertyMultiple-Error

'Error Class' = PROPERTY,

'Error Code' = INVALID\_DATATYPE |

RECEIVE BACnet-Reject-PDU

'Reject Reason' = INVALID\_PARAMETER\_DATATYPE

--Check that P1 has Relinquish\_Default value and P2 remains the same

9. VERIFY P1 = V1

10. VERIFY P2 = V2

--Check that the Channel object ignores the datatype error and Write\_Status is SUCCESSFUL

11. VERIFY Write\_Status = SUCCESSFUL

### 7.3.2.35 Elevator Group Object Tests

#### 7.3.2.35.1 Machine\_Room\_ID property references a Positive Integer Value Object

Purpose: To verify that the Machine\_Room\_ID property of an Elevator Group object can only reference a Positive Integer Value object or an object with instance number of 4194303.

Test Concept: The Machine\_Room\_ID property of an Elevator Group object, EG1, is read to verify that it contains an object reference to a Positive Integer Value object, PIV, or an object with instance number of 4194303. If the property is writable, an attempt is made to write an object reference, O1, that is not a Positive Integer Value object and has an instance number 0-4194302 (inclusive) to verify that an error is returned.

Test Steps:

1. IF (Machine\_Room\_ID contains room identification number) THEN  
VERIFY (EG1), Machine\_Room\_ID = (PIV)  
ELSE  
VERIFY (EG1), Machine\_Room\_ID = (any object type, 4194303)
2. IF (Machine\_Room\_ID is writeable) THEN  
Transmit WriteProperty-Request  
'Object Identifier' = EG1,  
'Property Identifier' = Machine\_Room\_ID,  
'Property Value' = O1  
Receive BACnet-Error-PDU  
'Error Class' = PROPERTY,  
'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 7.3.2.35.2 Linking of Lift and Escalator Objects under Group\_Members property of the Elevator Group Object

Purpose: This test verifies that objects in the Group\_Members property of Elevator Group objects contain a reference back to the Elevator Group that has it listed as a member.

Test Concept: The Group\_Members property of each Elevator Group object is read to identify member Lift and Escalator objects. The Elevator\_Group property is read from each member Lift object and Escalator object to verify it contains a reference back to the appropriate Elevator Group object. The Elevator\_Group property of the remaining Lift and Escalator objects are read to verify that it contains an object identifier instance of 4194303.

Configuration Requirements: If the IUT supports a Group\_Members property that can be made to contain a reference to one or more Lift objects, than it shall be configured as such. If the IUT supports a Group\_Members property that can be made to contain a reference to one or more Escalator objects, it shall be configured as such.

Test Steps:

1. REPEAT EGO = (each Elevator Group object in the IUT) {  
     READ L1 = (EGO, Group\_Members)  
     IF (L1 is not empty) THEN {  
         REPEAT O1 = (each Lift or Escalator object in L1) {  
             READ EGP = (O1, Elevator\_Group)  
             VERIFY EGP = EGO  
         }  
     }  
 }  
 }  
 }  
 3. REPEAT O1 = (each remaining Lift or Escalator object in the IUT) {  
     READ EGP = (O1, Elevator\_Group)  
     VERIFY EGP = (any object type, 4194303)  
 }

### 7.3.2.35.3 Landing\_Call\_Control Test

Purpose: To verify that writing to the Landing\_Call\_Control property updates the Landing\_Call\_Control and Landing\_Calls properties in the Elevator Group object and updates the Assigned\_Landing\_Calls property of a linked Lift object

Test Concept: The Landing\_Call\_Control property of an Elevator Group object (EG1) is written with a value that represents a request to travel upwards from FN1. The Landing\_Call\_Control and Landing\_Calls properties of EG1 and the Assigned\_Landing\_Calls property of the linked Lift object (L1) are checked to verify they updated correctly. The Landing\_Call\_Control property is written with a value that represents a request to travel downwards from FN2 and the aforementioned properties are checked again. The optional 'floor-text' parameter is used in one of the WRITE steps to verify the server will ignore this parameter when present. In the test steps, DF represents a valid destination floor.

Configuration Requirements: Lift object (L1) is contained in the Group\_Members property of the Elevator Group object (EG1) and has a door at array index Y on the same side of the landing call. FN1 and FN2 values should be sufficiently far away from the current position of L1 to allow for reading of the property values. No other processes shall be generating landing calls during this test.

Notes to Tester: If the Elevator Group contains more than 1 lift, the value written to Landing\_Call\_Control may get assigned to any other lift in the group based on the lift algorithm.

Test Steps:

1. WRITE EG1, Landing\_Call\_Control = (FN1, UP | DF (DF > FN1), "test string")
2. VERIFY EG1, Landing\_Call\_Control = (FN1, UP | DF, floor-text (optional))
3. VERIFY EG1, Landing\_Calls = (FN1, UP | DF, floor-text (optional))
4. IF (L1 contains the Assigned\_Landing\_Calls property)  
     VERIFY L1, Assigned\_Landing\_Calls, ARRAY INDEX (Y) = (FN1, UP)
5. WAIT (a time interval sufficient for the car to complete the call + **Internal Processing Fail Time**)
6. VERIFY EG1, Landing\_Calls = ()
7. IF (L1 contains the Assigned\_Landing\_Calls property)  
     VERIFY L1, Assigned\_Landing\_Calls, ARRAY INDEX (Y) = ()
8. WRITE EG1, Landing\_Call\_Control = (FN2, DOWN | DF (DF < FN2))
9. VERIFY EG1, Landing\_Call\_Control = (FN2, DOWN | DF, floor-text (optional))
10. VERIFY EG1, Landing\_Calls = (FN2, DOWN | DF, floor-text (optional))
11. IF (L1 contains the Assigned\_Landing\_Calls property)  
     VERIFY L1, Assigned\_Landing\_Calls = (FN1, DOWN)
12. WAIT (a time interval sufficient for the car to complete the call + **Internal Processing Fail Time**)
13. VERIFY EG1, Landing\_Calls = ()
14. IF (L1 contains the Assigned\_Landing\_Calls property)  
     VERIFY L1, Assigned\_Landing\_Calls, ARRAY INDEX (Y) = ()

### 7.3.2.36 Lift Object Tests

#### 7.3.2.36.1 Array Size of the Lift Object Properties Based on Number of Car Doors

Purpose: To verify that the size of the arrays for the Car\_Door\_Text, Assigned\_Landing\_Calls, Making\_Car\_Call, Registered\_Car\_Call, Car\_Door\_Status, Car\_Door\_Command, and Landing\_Door\_Status properties are the same.

Test Concept: The array size for each of the above properties, if present, is read and the sizes are compared to verify they are all equal.

Test Steps:

1. VERIFY (L1), Car\_Door\_Text = (Number of car doors present in the Lift), ARRAY INDEX = 0
2. VERIFY (L1), Assigned\_Landing\_Calls = (Number of car doors present in Lift), ARRAY INDEX = 0
3. VERIFY (L1), Making\_Car\_Call = (Number of car doors present in the Lift), ARRAY INDEX = 0
4. VERIFY (L1), Registered\_Car\_Call = (Number of car doors present in the Lift), ARRAY INDEX = 0
5. VERIFY (L1), Car\_Door\_Status = (Number of car doors present in the Lift), ARRAY INDEX = 0
6. VERIFY (L1), Car\_Door\_Command = (Number of car doors present in the Lift), ARRAY INDEX = 0
7. VERIFY (L1), Landing\_Door\_Status = (Number of car doors present in the Lift), ARRAY INDEX = 0
8. CHECK (Array index 0 of all these properties shall be same)

#### 7.3.2.36.2 Lift Properties Operational Test

Purpose: To verify that the property values in the Lift object update when it responds to a call.

Test Concept: The test starts with the Lift object, L1, in the lowest floor that it serves, LF, and property values are checked. A request is made to move the lift to the highest floor that it serves, HF, and property values are checked while the lift is moving and again when the lift arrives at HF. If the IUT does not contain the property specified in the test step, that step shall be skipped. In the test steps, DSR is a specific array index corresponding to the car door servicing the request.

Configuration Requirements: At the start of the test, the lift corresponding to L1 is at LF, and there are no active calls for L1. Throughout the test, L1 is in a normal operating state such that Car\_Mode = NORMAL, Out\_Of\_Service = FALSE, and no other processes shall be attempting to control L1.

Test Steps:

1. READ LF = Car\_Position
2. READ DS1 = Car\_Door\_Status
3. VERIFY Floor\_Text = (any value), ARRAY INDEX = LF
4. VERIFY Floor\_Text = (any value), ARRAY INDEX = HF
5. REPEAT N = (each array element) DO {  
    VERIFY Assigned\_Landing\_Calls = {}, ARRAY INDEX = N  
}
6. REPEAT N = (each array element) DO {  
    VERIFY Registered\_Car\_Calls = {}, ARRAY INDEX = N  
}
7. VERIFY Car\_Moving\_Direction <> UP | DOWN
8. VERIFY Car\_Mode = NORMAL
9. VERIFY Next\_Stopping\_Floor = LF
10. VERIFY Passenger\_Alarm = FALSE
11. VERIFY Reliability = NO\_FAULT\_DETECTED
12. VERIFY Out\_Of\_Service = FALSE
13. VERIFY Car\_Drive\_Status = STATIONARY | UNKNOWN
14. REPEAT N = (each array element) DO {  
    VERIFY Landing\_Door\_Status = (a list containing an entry {LF, DS1[N]}), ARRAY INDEX = N  
}
15. MAKE (A command that will cause L1 to travel to HF)

--Complete steps 16 – 19 before L1 reaches HF

16. IF (command was generated via Landing call) THEN {  
    VERIFY Assigned\_Landing\_Calls = (HF, DOWN), ARRAY INDEX = DSR

```

}
ELSE {
 --command was generated via Car call
 VERIFY Making_Car_Call = (HF), ARRAY INDEX = DSR
 VERIFY Registered_Car_Calls = (HF), ARRAY INDEX = DSR
 VERIFY Car_Assigned_Direction = (UP)
}
17. VERIFY Car_Moving_Direction = UP
18. VERIFY Next_Stopping_Floor = HF
19. VERIFY Car_Drive_Status <> STATIONARY
20. WAIT (for L1 to reach HF) + Internal Processing Fail Time
21. REPEAT N = (each array element) DO {
 VERIFY Registerd_Car_Calls = {}, ARRAY INDEX = N
}
22. VERIFY Car_Position = HF
23. VERIFY Car_Moving_Direction <> UP | DOWN
24. VERIFY Next_Stopping_Floor = HF
25. READ DS2 = Car_Door_Status
26. REPEAT (N = each array element) DO {
 VERIFY Landing_Door_Status = (a list containing an entry {LF, DS2[N]}), ARRAY INDEX = N
}

```

### 7.3.2.36.3 Out\_Of\_Service, Status\_Flags for Lift Object

Purpose: To verify the interrelationship between Out\_Of\_Service and Status\_Flags and that properties dictated by the standard to be writable when Out\_Of\_Service is TRUE are writable when Out\_Of\_Service is TRUE.

Test Concept: Out\_Of\_Service is set to TRUE and Status\_Flags is checked to verify the Out\_Of\_Service flag is set. While Out\_Of\_Service is TRUE, each of the properties (represented by LP), if present in the object, is read to obtain the current property value, X, and written with a different property value, Y. The property value is read again to verify it changed to Y.

LP = (Assigned\_Landing\_Calls, Registered\_Car\_Call, Car\_Position, Car\_Moving\_Direction, Car\_Assigned\_Direction, Car\_Door\_Status, Car\_Door\_Zone, Car\_Load, Next\_Stopping\_Floor, Passenger\_Alarm, Energy\_Meter, Car\_Drive\_Status, Fault\_Signals, Landing\_Door\_Status, Making\_Car\_Call, Car\_Door\_Command, and Car\_Mode)

Test Steps:

1. WRITE Out\_Of\_Service = TRUE
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (? , ? , ? , TRUE)
4. REPEAT P = (each property in LP present in the object) DO {
  - READ X = P
  - WRITE P = Y
  - WAIT **Internal Processing Fail Time**
  - VERIFY (P=Y)

### 7.3.2.36.4 Energy\_Meter\_Ref Property Tests

Purpose: To verify linking of Energy\_Meter property and Energy\_Meter\_Ref property.

Test Concept: If the Energy\_Meter\_Ref property of an object (O1) is present and initialized (contains an instance other than 4194303), then the Energy\_Meter property, if present, shall have a value of 0.0. If Energy\_Meter\_Ref is present and is un-initialized, then the value of Energy\_Meter property shall have any valid value.

Test Steps:

1. IF (Energy\_Meter\_Ref is present and initialized with instance other than 4194303) THEN
  - VERIFY Energy\_Meter = 0.0
- ELSE
  - VERIFY Energy\_Meter = (Any Valid Value)

### 7.3.2.37 Escalator Object Tests

#### 7.3.2.37.1 Out\_Of\_Service, Status\_Flags for Escalator Object

Purpose: To verify the interrelationship between Out\_Of\_Service and Status\_Flags and that properties dictated by the standard to be writable when Out\_Of\_Service is TRUE are writable when Out\_Of\_Service is TRUE.

Test Concept: Out\_Of\_Service is set to TRUE and Status\_Flags is checked to verify the Out\_Of\_Service flag is set. While Out\_Of\_Service is TRUE, each of the properties (represented by EP), if present in the object, is read to obtain the current property value, X, and written with a different property value, Y. The property value is read again to verify it changed to Y.

EP = (Power\_Mode, Operation\_Direction, Escalator\_Mode, Energy\_Meter, Fault\_Signals, and Passenger\_Alarm)

Test Steps:

1. WRITE Out\_Of\_Service = TRUE
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, ?, ?, TRUE)
4. REPEAT P = (each property in LP present in the object) DO {
  - READ X = P
  - WRITE (P = Y)
  - WAIT Internal Processing Fail Time
  - VERIFY (P = Y)

### 7.3.2.38 Load Control Object Tests

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

#### 7.3.2.38.1 Requested\_Shed\_Level property test with LEVEL choice

Purpose: To verify that the IUT can accept and execute a shed request with LEVEL choice.

Test Concept: The Requested\_Shed\_Level property of the Load Control object is set to a LEVEL choice and it is verified that the Load Control object behaves as per the Load Control state machine.

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

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

Test Steps:

1. VERIFY Requested\_Shed\_Level = (DRSL : one of the default Requested\_Shed\_Level values for a previous shed request, not necessarily the LEVEL default of 0)
2. VERIFY Expected\_Shed\_Level = DRSL
3. VERIFY Actual\_Shed\_Level = DRSL
4. VERIFY Present\_Value = SHED\_INACTIVE
5. VERIFY Shed\_Duration = 0
6. VERIFY Start\_Time = (the fully unspecified datetime value)
7. VERIFY Duty\_Window = (PAV, the pre-agreed upon value)
8. WRITE Enable = TRUE
9. WRITE Shed\_Duration = (SD, any value appropriate to the object)
10. WRITE Start\_Time = (ST, any valid start time including values in the past, present or future, but limited to such that current\_time is before ST + SD)

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

### 7.3.2.38.2 Shed\_Levels property test

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

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

Test Steps:

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

### 7.3.2.38.3 Load Control Status\_Flags and Reliability Test

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

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

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

Test Steps:

1. VERIFY Reliability = NO\_FAULT\_DETECTED
2. VERIFY Status\_Flags = (?, FALSE, ?, FALSE)
3. REPEAT X = (all values of the Reliability enumeration appropriate to the



```

 object type except NO_FAULT_DETECTED) DO {
 WRITE Reliability = X
 VERIFY Reliability = X
 VERIFY Status_Flags = (TRUE, TRUE, ?, FALSE)
 WRITE Reliability = NO_FAULT_DETECTED
 VERIFY Reliability = NO_FAULT_DETECTED
 VERIFY Status_Flags = (? FALSE, ?, FALSE)
}

```

#### 7.3.2.38.4 Requested\_Shed\_Level property test with PERCENT choice

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

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

Test Steps: The test steps defined in test 7.3.2.38.1 shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to a PERCENT choice, and the default value for a shed request with PERCENT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 100

#### 7.3.2.38.5 Requested\_Shed\_Level property test with AMOUNT choice

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

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

Test Steps: The test steps defined in test 7.3.2.38.1 shall be followed except that the Requested\_Shed\_Level property of the Load Control object is written to an AMOUNT choice, and the default value for a shed request with AMOUNT choice in Requested\_Shed\_Level, Expected\_Shed\_Level, and Actual\_Shed\_Level properties is 0.0

### 7.3.2.39 Lighting Output Object Tests

#### 7.3.2.39.1 Lighting Output Tracking Test

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

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

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

Test Steps:

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



**7.3.2.39.2 Lighting Output Present\_Value between 0.0 and 1.0 Test**

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

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

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

Test Steps:

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

**7.3.2.39.3 Lighting Command Operation NONE Test**

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

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

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = O1  
     'Property Identifier' = Lighting\_Command  
     'Property Value' = NONE
3. RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE
4. VERIFY (Object1), Lighting\_Command = (the value defined for this property in the EPICS)

**7.3.2.39.4 Lighting Command Operation FADE\_TO Test**

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

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

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

Test Steps:

- Start with 0% Present\_Value to test fade up
1. WRITE Present\_Value = 0, ARRAY INDEX = PTY1
  2. VERIFY Present\_Value = 0
  3. WAIT **Internal Processing Fail Time**
  4. VERIFY Tracking\_Value = 0
- Write a FADE\_TO command (operation, target-level, priority, fade-time)
5. WRITE Lighting\_Command = (FADE\_TO, V1, PTY1, FT)
  6. WAIT **Internal Processing Fail Time**

7. VERIFY Priority\_Array = V1, ARRAY INDEX = PTY1
8. VERIFY Present\_Value = V1

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

9. WAIT FT/2
10. VERIFY In\_Progress = FADE\_ACTIVE,
11. VERIFY Tracking\_Value  $\approx$  V1 / 2
12. WAIT FT/2

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

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

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

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

-- Write a FADE\_TO command (operation, target-level, priority, fade-time)

19. WRITE Lighting\_Command = (FADE\_TO, V2, PTY1, FT)
20. WAIT **Internal Processing Fail Time**
21. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1
22. VERIFY Present\_Value = V2

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

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

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

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

### 7.3.2.39.5 Lighting Command Operation RAMP\_TO Test

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

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

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

Test Steps:

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

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

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

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

8. VERIFY Present\_Value = V1

-- Check In\_Progress while ramping up

9. VERIFY In\_Progress = RAMP\_ACTIVE

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

10. WHILE (In\_Progress <> IDLE) DO {

11. VERIFY Tracking\_Value > 0 < V1

12. CHECK (Tracking\_Value is increasing with the ramp-rate)}

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

13. VERIFY In\_Progress = IDLE

14. VERIFY Tracking\_Value = V1

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

15. WRITE Present\_Value = 100, ARRAY INDEX = PTY1

16. VERIFY Present\_Value = 100

17. WAIT **Internal Processing Fail Time**

18. VERIFY Tracking\_Value = 100

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

19. WRITE Lighting\_Command = (RAMP\_TO, V2, PTY1, any valid rate)

20. WAIT **Internal Processing Fail Time**

21. VERIFY Priority\_Array = V2, ARRAY INDEX = PTY1

22. VERIFY Present\_Value = V2

-- Check In\_Progress while ramping up

23. VERIFY In\_Progress = RAMP\_ACTIVE,

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

24. WHILE (In\_Progress <> RAMP\_ACTIVE) DO {

25. VERIFY Tracking\_Value < 0

26. VERIFY Tracking\_Value > V2

27. CHECK (Tracking\_Value is decreasing with the ramp-rate)}

-- Check In\_Progress and Tracking\_Value

28. VERIFY In\_Progress = IDLE

29. VERIFY Tracking\_Value = V2

### 7.3.2.39.6 Lighting Command Operation STEP\_UP Test

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

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

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

Test Steps:

-- Start with 0% Present\_Value

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

2. VERIFY Present\_Value = 0

3. WAIT **Internal Processing Fail Time**

4. VERIFY Tracking\_Value = 0

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

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

6. WAIT **Internal Processing Fail Time**

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

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

8. VERIFY Present\_Value = 0

9. VERIFY Tracking\_Value = 0

-- Now test with Tracking\_Value > 0

10. WRITE Present\_Value = 1, ARRAY INDEX = PTY1

11. VERIFY Present\_Value = 1

12. WAIT **Internal Processing Fail Time**

13. VERIFY Tracking\_Value = 1

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

14. REPEAT X = (1 through (100 - step-increment) by step-increment) DO{

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

    WAIT **Internal Processing Fail Time**

    VERIFY Priority\_Array = X + step-increment, ARRAY INDEX = PTY1

    VERIFY Present\_Value = X + step-increment

    VERIFY Tracking\_Value = X + step-increment

}

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

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

16. WAIT **Internal Processing Fail Time**

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

18. VERIFY Present\_Value = 100

19. VERIFY Tracking\_Value = 100

### 7.3.2.39.7 Lighting Command Operation STEP\_DOWN Test

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

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

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

Test Steps:

-- Start with 0% Present\_Value

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

2. VERIFY Present\_Value = 0

3. WAIT **Internal Processing Fail Time**

4. VERIFY Tracking\_Value = 0

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

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

6. WAIT **Internal Processing Fail Time**

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

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

-- Now test with Tracking\_Value = 100

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

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

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

-- Now step down one more time to confirm that the values will not go down below 1

15. WRITE Lighting\_Command = (STEP\_DOWN, PTY1, any valid value)
16. WAIT **Internal Processing Fail Time**
17. VERIFY Priority\_Array = 1, ARRAY INDEX = PTY1
18. VERIFY Present\_Value = 1
19. VERIFY Tracking\_Value = 1

### 7.3.2.39.8 Lighting Command Operation STEP\_ON Test

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

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

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

Test Steps:

-- Start with 0% Present\_Value

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

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

5. WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
6. WAIT **Internal Processing Fail Time**

-- Confirm that the Present\_Value and Tracking\_Value became 1

7. VERIFY Priority\_Array = 1, ARRAY INDEX = PTY1
8. VERIFY Present\_Value = 1
9. VERIFY Tracking\_Value = 1

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

10. REPEAT X = (1 through (100 - step-increment)) DO {
  - WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)
  - WAIT **Internal Processing Fail Time**

```

VERIFY Priority_Array = X + step-increment, ARRAY INDEX = PTY1
VERIFY Present_Value = X + step-increment
VERIFY Tracking_Value = X + step-increment

```

```

}

```

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

11. WRITE Lighting\_Command = (STEP\_ON, PTY1, any valid values)

12. WAIT **Internal Processing Fail Time**

13. VERIFY Priority\_Array = 100, ARRAY INDEX = PTY1

14. VERIFY Present\_Value = 100

15. VERIFY Tracking\_Value = 100

### 7.3.2.39.9 Lighting Command Operation STEP\_OFF Test

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

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

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

Test Steps:

-- Start with 0% Present\_Value

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

2. VERIFY Present\_Value = 0

3. WAIT **Internal Processing Fail Time**

4. VERIFY Tracking\_Value = 0

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

5. WRITE Lighting\_Command = (STEP\_OFF, PTY1, step-increment)

6. WAIT **Internal Processing Fail Time**

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

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

8. VERIFY Present\_Value = 0

9. VERIFY Tracking\_Value = 0

-- Now test with Tracking\_Value = 100

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

11. VERIFY Present\_Value = 100

12. WAIT **Internal Processing Fail Time**

13. VERIFY Tracking\_Value = 100

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

14. REPEAT X = (100 through (1 + step-increment)) DO {

```

 WRITE Lighting_Command = (STEP_OFF, PTY1, step-increment)

```

```

 WAIT Internal Processing Fail Time

```

```

 VERIFY Priority_Array = X - step-increment, ARRAY INDEX = PTY1

```

```

 VERIFY Present_Value = X - step-increment

```

```

 VERIFY Tracking_Value = X - step-increment

```

```

}

```

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



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

### 7.3.2.39.10 Transition None Test

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

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

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

Test Steps:

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

### 7.3.2.39.11 Transition Test

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

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

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

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

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

Test Steps:

1. VERIFY Transition = T
2. VERIFY In\_Progress = IDLE
3. V0 = READ Present\_Value
4. WRITE Present\_Value = V1, ARRAY INDEX = PTY1
5. VERIFY Present\_Value = V1
6. WAIT W1
7. VERIFY Tracking\_Value  $\approx (V1 + V0) / 2$
8. VERIFY In\_Progress = A



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

### 7.3.2.39.12 Feedback\_Value Clamping Test

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

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

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

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

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

Test Steps:

- ```
-- Verify Feedback_Value when output is above Max_Actual_Value
1. WRITE Max_Actual_Value = 100
2. WRITE Min_Actual_Value = 1
3. WRITE Present_Value = 100, PRIORITY = PTY1
4. WHILE In_Progress <> IDLE {}
5. WRITE Max_Actual_Value = (Lowest supported Max_Actual_Value)
6. VERIFY Feedback_Value = 100

-- Verify Feedback_Value when output is below Min_Actual_Value
7. WRITE Max_Actual_Value = 100
8. WRITE Min_Actual_Value = 1
9. WRITE Present_Value = 1, PRIORITY = PTY1
10. WHILE In_Progress <> IDLE {}
11. WRITE Min_Actual_Value = (Highest supported Min_Actual_Value)
12. VERIFY Feedback_Value = 1
```

7.3.2.39.13 Min_Actual_Value and Max_Actual_Value Test

Purpose: To verify that Min_Actual_Value remains less than Max_Actual_Value and within the allowable range when either is written to a value that would violate these conditions.

Test Concept: Write a value to Min_Actual_Value which is larger than Max_Actual_Value. Verify that Max_Actual_Value became equal to Min_Actual_Value. Next, write a value to Max_Actual_Value which is less than Min_Actual_Value. Verify that Min_Actual_Value became equal to Max_Actual_Value.

Verify that neither Min_Actual_Value nor Max_Actual_Value will accept a value outside the range 1.0 to 100.0.

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

Test Steps:

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

7.3.2.39.14 Min_Actual_Value and Max_Actual_Value Scaling Test

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

Test Concept: Set Min_Actual_Value to a value other than the lowest supported minimum value, and set Max_Actual_Value to a value other than the highest support value but larger than Min_Actual_Value.

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

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

Test Steps:

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

7.3.2.40 Access Door Object Tests

7.3.2.40.1 Commandable Present_Value Test

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

Test Concept: The IUT shall be configured with a door control output that can be observed during the test. The Present_Value property is written with each of the following values: UNLOCK, LOCK, PULSE_UNLOCK, EXTENDED_PULSE_UNLOCK, and the Access_Door object is monitored to ensure that the door locks and unlocks appropriately.

Configuration Requirements: The Relinquish_Default shall have the value LOCK. All writes are at a priority higher than any internal algorithms writing to this property. Out_Of_Service shall be set to FALSE. Prior to the test the Present_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

-- Test UNLOCK value

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

-- Test LOCK value

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

-- Test PULSE_UNLOCK value

9. WRITE Present_Value = PULSE_UNLOCK
10. WAIT (**Internal Processing Fail Time** + Door_Unlock_Delay_Time if present)
11. IF (Lock_Status is present) THEN
 VERIFY Lock_Status = UNLOCKED
12. CHECK (that the IUT is in a state that would cause the door to be unlocked)
13. WAIT (Door_Pulse_Time)
14. VERIFY Present_Value = LOCK
15. IF (Lock_Status is present) THEN
 VERIFY Lock_Status = LOCKED
16. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED_PULSE_UNLOCK value

17. WRITE Present_Value = EXTENDED_PULSE_UNLOCK
18. WAIT (**Internal Processing Fail Time** + Door_Unlock_Delay_Time if present)

7.3.2.40.2 Door_Status, Lock_Status and Door_Alarm_State Tests

Purpose: This test case verifies that Door_Status, Lock_Status, and Door_Alarm_State properties are writable when Out_Of_Service is TRUE.

Test Concept: Set Out_Of_Service to TRUE and then make sure one at a time that Door_Status, Lock_Status, and Door_Alarm_State, if present, are writable.

Configuration Requirements: If the Out_Of_Service property of this object is not writable, and if the Out_Of_Service property cannot be changed by other means, then this test shall be omitted. All writes to the Present_Value shall be performed at a priority higher (numerically smaller) than any internal algorithms writing to this property. For testing Door_Alarm_State, test only values listed in either the Alarm_Values or Fault_Values.

Test Steps:

1. MAKE (Out_Of_Service TRUE)
2. VERIFY Status_Flags = (?, ?, ?, TRUE)
3. IF (Door_Status is present) THEN
 REPEAT X = (all values of the Door_Status enumeration values supported by the property) DO {
 WRITE Door_Status = X

```

        VERIFY Door_Status = X
    }
4. IF (Lock_Status is present) THEN
    REPEAT X = (all values of the Lock_Status enumeration values supported by the property) DO {
        WRITE Lock_Status = X
        VERIFY Lock_Status = X
    }
5. IF (Door_Alarm_State is present) THEN
    REPEAT X = (all values of the Door_Alarm_State enumeration values supported by the property) DO {
        WRITE Door_Alarm_State = X
        VERIFY Door_Alarm_State = X
    }

```

7.3.2.40.3 Door_Status with Physical Door Status Tests

Purpose: To verify that the Door_Status property reflects the state of the physical door (CLOSED, OPENED, UNUSED, and DOOR_FAULT if the object supports detecting door faults).

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

Configuration Requirements: The IUT shall be configured such that it can determine the state of a door. The Access_Door object associated with this physical door shall be configured with Out_Of_Service = FALSE.

Test Steps:

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

7.3.2.40.4 Lock_Status Tests

Purpose: To verify that the Lock_Status property reflects the state of the physical lock. (LOCKED, UNLOCKED, and LOCK_FAULT if the object supports detecting lock faults).

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

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

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

Test Steps:

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

7.3.2.40.5 Secured_Status Tests

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

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

Configuration Requirements: All writes to the Present_Value shall be performed at a priority higher than any internal algorithms writing to this property. If this object supports intrinsic reporting, then the Alarm_Values property shall be empty. If this object supports the Masked_Alarm_Values property, then it shall be empty. Out_Of_Service is FALSE.

Test Steps:

- Create a condition where the Secured_Status becomes SECURED
 1. WRITE Present_Value = LOCK
 2. WAIT (**Internal Processing Fail Time**)
 3. VERIFY Status_Flags = (FALSE ?, ?, ?)
 4. IF (Lock_Status property is present) THEN
 - MAKE (Lock_Status = LOCKED or UNUSED)
 5. MAKE (Door_Status = CLOSED or UNUSED)
- Verify that the Secured_Status is SECURED when it should
 6. VERIFY Secured_Status = SECURED
- Verify that Secured_Status is UNSECURED when Present_Value is anything other than LOCKED
 7. REPEAT X = (UNLOCK, PULSE_UNLOCK, EXTENDED_PULSE_UNLOCK) DO {
 - WRITE Present_Value = X
 - WAIT (**Internal Processing Fail Time**)
 - VERIFY Secured_Status = UNSECURED
- Recreate a condition where the Secured_Status becomes SECURED again
 8. WRITE Present_Value = LOCK
 9. WAIT (**Internal Processing Fail Time**)
 10. VERIFY Secured_Status = SECURED
- Verify that Secured_Status is UNSECURED when Masked_Alarm_Value, if exist, is NOT empty
 11. IF (Masked_Alarm_Values is present) THEN
 - MAKE (Masked_Alarm_Values = (any valid BACnetDoorAlarmState enumeration))
 - WAIT(**Internal Processing Fail Time**)
 - VERIFY Secured_Status = UNSECURED
- Recreate a condition where the Secured_Status becomes SECURED again
 - MAKE (Masked_Alarm_Values = {})
 - WAIT (**Internal Processing Fail Time**)
 - VERIFY Secured_Status = SECURED
- Verify that Secured_Status is UNSECURED when Lock_Status, if present, is anything other than LOCKED or UNUSED
 12. IF (Lock_Status property is present) THEN
 - REPEAT X = (UNLOCKED, UNKNOWN, LOCK_FAULT) DO {
 - MAKE (Lock_Status = X)
 - WAIT (**Internal Processing Fail Time**)
 - VERIFY Secured_Status = UNSECURED
 - REPEAT X = (LOCKED, UNUSED) DO {
 - MAKE (Lock_Status = X)
 - VERIFY Secured_Status = SECURED
- Verify that Secured_Status is UNSECURED when Door_Status, is anything other than CLOSED or UNUSED

```

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

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

```

7.3.2.40.6 Door_Unlock_Delay_Time Test

Purpose: To verify that when the Door_Unlock_Delay_Time property has a non-zero value, the output is delayed in unlocking when a PULSE_UNLOCK or EXTENDED_PULSE_UNLOCK is written to the Present_Value and not when UNLOCK is written.

Test Concept: When unlocking the door by writing PULSE_UNLOCK to the Present_Value of the Access Door object, it is verified that the door is still locked for the specified Door_Pulse_Time, then the door is unlocked. The same test is done for EXTENDED_PULSE_UNLOCK, but this time, it is verified that the door is still locked for the specified Door_Extended_Pulse_Time then the door is unlocked.

Configuration Requirements: The IUT shall be configured with a door control output that can be observed during the test. The Relinquish_Default shall have the value LOCK. All writes to the Present_Value shall be performed at a priority higher than any internal algorithms writing to this property. Door_Unlock_Delay_Time shall be set to a non-zero value which is sufficient to observe the delay and check the status of the lock. Out_Of_Service shall be set to FALSE. Prior to the test the Present_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

```

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

4. IF (Lock_Status is present) THEN
    VERIFY Lock_Status = UNLOCKED
5. CHECK (that the door control output is in a state that would cause the door to be unlocked)
6. WAIT (Door_Pulse_Time)
7. VERIFY Present_Value = LOCK
8. IF (Lock_Status is present) THEN
    VERIFY Lock_Status = LOCKED
9. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED_PULSE_UNLOCK
10. WRITE Present_Value = EXTENDED_PULSE_UNLOCK
11. WAIT (Internal Processing Fail Time)
12. BEFORE Door_Unlock_Delay_Time
    IF (Lock_Status is present) THEN

```


VERIFY Lock_Status = LOCKED
CHECK (that the door control output is in a state that would cause the door to be locked)

13. IF (Lock_Status is present) THEN
 VERIFY Lock_Status = UNLOCKED
14. CHECK (that the door control output is in a state that would cause the door to be unlocked)
15. WAIT (Door_Extended_Pulse_Time)
16. VERIFY Present_Value = LOCK
17. IF (Lock_Status is present) THEN
 VERIFY Lock_Status = LOCKED
18. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test UNLOCK

19. WRITE Present_Value = UNLOCK
20. WAIT (**Internal Processing Fail Time**)
21. IF (Lock_Status is present) THEN
 VERIFY Lock_Status = UNLOCKED
22. CHECK (that the door control output is in a state that would cause the door to be locked)

7.3.2.40.7 Masked_Alarm_Values Test

Purpose: To verify that the Masked_Alarm_Values prevents an intrinsic alarm from occurring.

Test Concept: The Access Door is verified to be in an Out_Of_Service state and is not in an alarm state. Then a non-NORMAL enumeration value of BACnetDoorAlarmState X is written to the Door_Alarm_State, and the Access Door object transitions to an alarm state. X is written to the Masked_Alarm_Value, and Door_Alarm_State is checked to verify it returned to NORMAL. The sequence is repeated for all non-NORMAL enumeration values of BACnetDoorAlarmState.

Configuration Requirements: The Masked_Alarm_Values list shall be empty at the start of this test. Out_Of_Service shall be set to TRUE to allow writing to the Door_Alarm_State property. If Out_Of_Service is not writeable and cannot be set to TRUE by any other means, this test shall be skipped. The enumeration BACnetDoorAlarmState value X to be used in the test has to be present in either the Alarm_Values or Fault_Values property.

Test Steps:

1. VERIFY Status_Flags = (FALSE, ?, ?, TRUE)
2. VERIFY Door_Alarm_State = NORMAL
3. REPEAT X = (all valid values of the enumeration BACnetDoorAlarmState except NORMAL) DO {
 WRITE Door_Alarm_State = X
 WAIT (**Internal Processing Fail Time**)
 VERIFY Status_Flags = (TRUE, ?, ?, TRUE)
 WRITE Masked_Alarm_Values = { X }
 WAIT (**Internal Processing Fail Time**)
 VERIFY Door_Alarm_State = NORMAL
 VERIFY Status_Flags = (FALSE, ?, ?, TRUE)
 WRITE Masked_Alarm_Values = { }
 WAIT (**Internal Processing Fail Time**)
}

7.3.2.40.8 Door_Open_Too_Long Test

Purpose: To verify that the DOOR_OPEN_TOO_LONG condition is generated when the Access Door object is commanded to the LOCK state, but the physical door remains open beyond Door_Open_Too_Long_Time.

Test Concept: Setup the Access Door object to trigger alarm on DOOR_OPEN_TOO_LONG state using Alarm_Values and Masked_Alarm_Values. Next, set the physical door to the closed state to confirm that the Access Door object is in NORMAL state. Then, unlock the physical door and set the physical door to the open state. Finally, command the Access Door object to LOCK and verify that the Door_Alarm_State changes to DOOR_OPEN_TOO_LONG after the specified Time_Delay.

Configuration Requirements: This test shall be skipped if the IUT does not support intrinsic alarming. The IUT shall be configured such that it can determine and change the open/closed state of a door. All writes to the Present_Value are at

a priority higher than any internal algorithms writing to this property. The `Door_Alarm_State` shall have the value `NORMAL` at the start of the test. The `Access Door` object is configured with `DOOR_OPEN_TOO_LONG` in the `Alarm_Values` property and excluded from `Masked_Alarm_Values` property if present.

Test Steps:

1. MAKE (set the physical door to the closed state)
2. VERIFY `Door_Alarm_State` = `NORMAL`
3. WRITE `Present_Value` = `UNLOCK`
4. MAKE (set the physical door to the open state)
5. WRITE `Present_Value` = `LOCK`
6. WAIT (**Internal Processing Fail Time**)
7. WHILE (`Door_Open_Too_Long_Time` has not expired) DO {
 VERIFY `Door_Alarm_State` = `NORMAL`
}
- WAIT (`Time_Delay`)
8. VERIFY `Door_Alarm_State` = `DOOR_OPEN_TOO_LONG`

7.3.2.41 Access Point Object Tests

The Access Point object type represents the external interface of the access control decision engine for a specific door. A credential is entered, the access rights of the credential are determined, and the access decision is determined based on the access rights. Testing this authentication and authorization functionality requires the support of other standard BACnet access control object types. The required and optional object types are shown in Figure 7-3-2-41.

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

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

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

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

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

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

- a) `Authorization_Mode` shall have the value `AUTHORIZE`.
- b) `Out_Of_Service` shall be `FALSE`.
- c) `Lockout` shall be `FALSE`.
- d) `Muster_Point` shall be `FALSE`.
- e) `Authentication_Status` shall be `READY`.

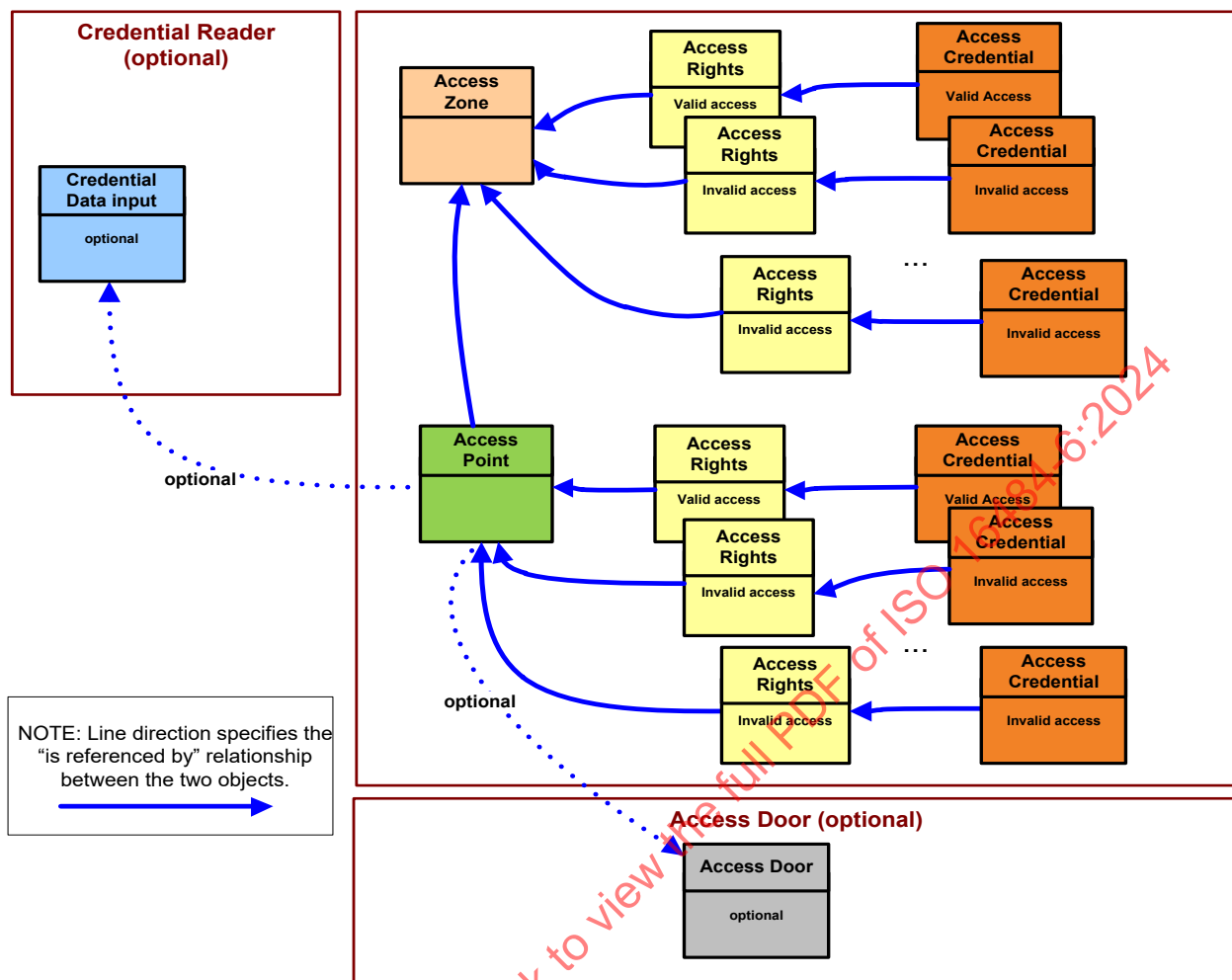


Figure 7-3-2-41: Objects and relationships used for testing Access Point objects

7.3.2.41.1 Authentication_Status and Access_Event Test

Purpose: This test case verifies the authentication and authorization process are disabled when Out_Of_Service of the Access Point object is TRUE. It also verifies the interrelationship between the Out_Of_Service, Authentication_Status, Access_Event, and Access_Event_Time properties.

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

Configuration Requirements: See 7.3.2.41.

Test Steps:

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

7. VERIFY Authentication_Status = READY
8. VERIFY Access_Event = OUT_OF_SERVICE_RELINQUISHED
9. VERIFY Access_Event_Time = (the time Out_Of_Service was set to FALSE)

7.3.2.41.2 Allowed Access Test

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

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration: An active credential with valid access rights for the access point shall be represented by Access Credential object C1.

Test Steps:

1. READ EventTag = Access_Event_Tag
2. MAKE (present a valid credential at credential reader for this access point)
3. VERIFY Access_Event = GRANTED
4. VERIFY Access_Event_Time = (the time that the credential was presented)
5. VERIFY Access_Event_Credential = C1
6. VERIFY Access_Event_Tag = EventTag + 1

7.3.2.41.3 Denied Access Test

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

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration: The vendor shall provide a set of credentials which correspond to Access Credential objects configured such that access to the access point shall be denied for the following reasons:

- a. DENIED_POINT_NO_ACCESS_RIGHTS = C1
- b. DENIED_NO_ACCESS_RIGHTS = C2
- c. DENIED_ZONE_NO_ACCESS_RIGHTS = C3
- d. DENIED_CREDENTIAL_NOT_YET_ACTIVE = C4
- e. DENIED_CREDENTIAL_EXPIRED = C5
- f. DENIED_CREDENTIAL_MANUAL_DISABLE = C6
- g. DENIED_INCORRECT_AUTHENTICATION_FACTOR = C7
- h. DENIED_OUT_OF_TIME_RANGE = C8
- i. DENIED_THREAT_LEVEL = C9
- j. DENIED_PASSBACK = C10
- k. DENIED_UNEXPECTED_LOCATION_USAGE = C11
- l. DENIED_MAX_ATTEMPTS = C12
- m. DENIED_AUTHENTICATION_FACTOR_LOST = C13
- n. DENIED_AUTHENTICATION_FACTOR_STOLEN = C14
- o. DENIED_AUTHENTICATION_FACTOR_DAMAGED = C15
- p. DENIED_AUTHENTICATION_FACTOR_DESTROYED = C16
- q. DENIED_AUTHENTICATION_FACTOR_DISABLED = C17
- r. DENIED_AUTHENTICATION_FACTOR_ERROR = C18
- s. DENIED_CREDENTIAL_UNASSIGNED = C19
- t. DENIED_CREDENTIAL_NOT_PROVISIONED = C20
- u. DENIED_CREDENTIAL_LOCKOUT = C21
- v. DENIED_CREDENTIAL_MAX_DAYS = C22
- w. DENIED_CREDENTIAL_MAX_USES = C23
- x. DENIED_CREDENTIAL_DISABLED = C24
- y. DENIED_LOCKOUT = C25

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

Test Steps:

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

7.3.2.41.4 Authorization Mode Test

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

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Note: If the VERIFICATION_REQUIRED or AUTHORIZATION_DELAYED mode is supported, the vendor must provide a mechanism for external verification to be performed.

Test Steps:

- verify GRANT_ACTIVE mode
- 1. IF (GRANT_ACTIVE is supported) THEN
 - READ EventTag = Access_Event_Tag
 - WRITE Authorization_Mode = GRANT_ACTIVE
 - MAKE (present credential C2 at credential reader for this access point)
 - VERIFY Access_Event = GRANTED
 - VERIFY Access_Event_Time = (the time that credential C2 was presented)
 - VERIFY Access_Event_Credential = C2
 - VERIFY Access_Event_Tag = EventTag + 1
- verify DENY_ALL mode
- 2. IF (DENY_ALL is supported) THEN
 - READ EventTag = Access_Event_Tag
 - WRITE Authorization_Mode = DENY_ALL
 - MAKE (present credential C1 at credential reader for this access point)
 - VERIFY Access_Event = DENIED_DENY_ALL
 - VERIFY Access_Event_Time = (the time that credential C1 was presented)
 - VERIFY Access_Event_Credential = C1
 - VERIFY Access_Event_Tag = EventTag + 1
- verify VERIFICATION_REQUIRED mode (verification authorized)

```

3. IF (VERIFICATION_REQUIRED is supported) THEN
    READ EventTag = Access_Event_Tag
    WRITE Authorization_Mode = VERIFICATION_REQUIRED
    MAKE (present credential C1 at credential reader for this access point)
    VERIFY Access_Event = VERIFICATION_REQUIRED
    VERIFY Access_Event_Time = (the time that credential C1 was presented most recently)
    VERIFY Access_Event_Credential = C1
    VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
    MAKE (external verification process grants access)
    VERIFY Access_Event = GRANTED
    VERIFY Access_Event_Time = (the time that verification process granted access)
    VERIFY Access_Event_Credential = C1
    VERIFY Access_Event_Tag = EventTag + 1

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

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

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

-- verify AUTHORIZATION_DELAYED mode (access denied)
    WRITE Authorization_Mode = AUTHORIZATION_DELAYED
    READ EventTag = Access_Event_Tag
    MAKE (present credential C1 at credential reader for this access point)

```

```

VERIFY Access_Event = AUTHORIZATION_DELAYED
VERIFY Access_Event_Time = (the time that credential C1 was presented)
VERIFY Access_Event_Credential = C1
MAKE (external verification process denies access)
VERIFY Access_Event = DENIED_VERIFICATION_FAILED
VERIFY Access_Event_Time = (the time that verification process denied access)
VERIFY Access_Event_Credential = C1
VERIFY Access_Event_Tag = EventTag + 1

```

-- verify NONE mode

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

7.3.2.41.5 Access Rights Exemptions Test

Purpose: To verify the access rights exemption functionality.

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Test Steps:

-- verify access is denied for the credential

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY Access_Event = DENIED_NO_ACCESS_RIGHTS
3. VERIFY Access_Event_Time = (the time that the credential was presented)
4. VERIFY Access_Event_Credential = C1

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

5. MAKE C1, Authorization_Exemption = (ACCESS_RIGHTS)
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY Access_Event = GRANTED
8. VERIFY Access_Event_Time = (the time that the credential was presented)
9. VERIFY Access_Event_Credential = C1

7.3.2.41.6 Change Authentication Policy Test

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

Test Concept: The Active_Authentication_Policy is written with values of 1 to X, where X is the number of valid authentication policies to verify that the value is accepted. Then, it is written with a value larger than X to verify that the value is rejected with a VALUE_OUT_OF_RANGE error. Finally, it is written with a value of 0 to verify that the value is also rejected with the same error.

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Test Steps:

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

1. READ Count = Number_Of_Authentication_Policies
2. REPEAT X = (1 to Count) DO {
 - WRITE Active_Authentication_Policy = X
 - VERIFY Active_Authentication_Policy = X

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

3. TRANSMIT WriteProperty-Request,
 'Object Identifier' = Access_Point object,
 'Property Identifier' = Active_Authentication_Policy,
 'Property Value' = (any value larger than Count)
4. RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE

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

5. TRANSMIT WriteProperty-Request,
 'Object Identifier' = Access_Point object,
 'Property Identifier' = Active_Authentication_Policy,
 'Property Value' = 0
6. RECEIVE BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE

7.3.2.41.7 Lockout State Test

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

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Test Steps:

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

1. WRITE Lockout = TRUE
2. WAIT **Internal Processing Fail Time**
3. VERIFY Access_Event = LOCKOUT_OTHER
4. VERIFY Access_Event_Time = (the time that TRUE was written to the Lockout property)
5. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
6. MAKE (present credential C1 at credential reader for this access point)
7. VERIFY Access_Event = DENIED_LOCKOUT
8. VERIFY Access_Event_Time = (the time that credential C1 was presented)
9. VERIFY Access_Event_Credential = C1

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

10. WRITE Lockout = FALSE
11. WAIT **Internal Processing Fail Time**
12. VERIFY Access_Event = LOCKOUT_RELINQUISHED
13. VERIFY Access_Event_Time = (the time that FALSE was written to the Lockout property)
14. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
15. IF (Failed_Attempts and Max_Failed_Attempts are supported) THEN
 REPEAT X= (1 to Max_Failed_Attempts + 1) DO {
 READ FailedAttempts = Failed_Attempts

MAKE (present credential C2 at credential reader for this access point)
 VERIFY (Failed_Attempts = FailedAttempts + 1)

}

16. VERIFY (Lockout = TRUE)
17. VERIFY (Access_Event = LOCKOUT_MAX_ATTEMPTS)
18. VERIFY (Access_Event_Time = the time that Lockout was set to TRUE)
19. VERIFY (Access_Event_Credential = C2)
20. MAKE (present credential C1 at credential reader for this access point)
21. VERIFY (Access_Event = DENIED_LOCKOUT)
22. VERIFY (Access_Event_Time = the time that credential C1 was presented)
23. VERIFY (Access_Event_Credential = C1)

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

24. WAIT Lockout_Relinquish_Time
25. VERIFY (Lockout = FALSE)
26. VERIFY (Access_Event = LOCKOUT_RELINQUISHED)
27. VERIFY (Access_Event_Time = the time that Lockout was set to FALSE)
28. VERIFY Access_Event_Credential = (4194303, ?, 4194303)
29. MAKE (present credential C1 at credential reader for this access point)
30. VERIFY (Access_Event = GRANTED)
31. VERIFY (Access_Event_Time = the time that credential C1 was presented)
32. VERIFY (Access_Event_Credential = C1)

7.3.2.41.8 Threat Level Test

Purpose: To verify Threat_Level is used to Grant or Deny access based on the Threat_Authority of the Access Credential.

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Test Steps:

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

1. WRITE Threat_Level = (any value less than X)
2. MAKE (present credential C1 at credential reader for this access point)
3. VERIFY Access_Event = GRANTED
4. VERIFY Access_Event_Time = (the time that credential C1 was presented)
5. VERIFY Access_Event_Credential = C1

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

6. WRITE Threat_Level = X
7. MAKE (present credential C1 at credential reader for this access point)
8. VERIFY Access_Event = GRANTED
9. VERIFY Access_Event_Time = (the time that credential C1 was presented)
10. VERIFY Access_Event_Credential = C1

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

11. WRITE Threat_Level = (any value greater than X)
12. MAKE (present credential C1 at credential reader for this access point)
13. VERIFY Access_Event = DENIED_THREAT_LEVEL
14. VERIFY Access_Event_Time = (the time that credential C1 was presented)
15. VERIFY Access_Event_Credential = C1

7.3.2.41.9 Denied Access Occupancy Upper Limit Test

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

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

Configuration Requirements: See Clause 7.3.2.41. This test requires the following additional configuration:

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

Test Steps:

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

```
1. REPEAT C = (C1...CX) DO {
    READ Count = Z1, Occupancy_Count
    MAKE (present credential C at credential reader for this access point)
    VERIFY Z1,Occupancy_Count = (Count + 1)
    VERIFY Access_Event = GRANTED
    VERIFY Access_Event_Time = (the time that credential C was presented)
    VERIFY Access_Event_Credential = C
}
```

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

```
2. READ Count = Z1, Occupancy_Count
3. VERIFY Z1,Occupancy_Upper_Limit = Count
4. MAKE (present credential C(X+1) at credential reader for this access point)
5. VERIFY Access_Event = DENIED_UPPER_OCCUPANCY_LIMIT
6. VERIFY Access_Event_Time = (the time that credential C(X+1) was presented)
7. VERIFY Access_Event_Credential = C(X+1)
8. VERIFY Z1,Occupancy_Count = Count
```

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

```
9. IF (the Authorization_Exemption property is not supported or the OCCUPANCY_CHECK exemption is not supported) THEN {
    READ Count = Z1, Occupancy_Count
    VERIFY Z1,Occupancy_Upper_Limit = Count
    WRITE C(X+1), Authorization_Exemptions = (OCCUPANCY_CHECK)
    MAKE (present credential C(X+1) at credential reader for this access point)
    VERIFY Access_Event = GRANTED
    VERIFY Access_Event_Time = (the time that credential C(X+1) was presented)
    VERIFY Access_Event_Credential = C(X+1)
    VERIFY Z1,Occupancy_Count = Count + 1
}
```

7.3.2.41.10 Denied Access Disabled Credential Test

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

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

Configuration Requirements: See Clause 7.3.2.41.1. This test requires the following additional configuration:

- a) An access credential with valid access to AP1 shall be represented by Access Credential C1.
- b) The Credential_Status property shall have the value ACTIVE.
- c) The Reason_For_Disable property shall be empty.

Test Steps:

– test granted access when credential is active

1. VERIFY Reason_For_Disable = ()
2. VERIFY Credential_Status = ACTIVE
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1,Access_Event = GRANTED
5. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access_Event_Credential = C1

– test denied access when credential is inactive

7. MAKE (Expiration_Time = time < current time)
8. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
9. VERIFY Credential_Status = INACTIVE
10. MAKE (present credential C1 at credential reader for access point AP1)
11. VERIFY AP1,Access_Event = DENIED_CREDENTIAL_EXPIRED
12. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
13. VERIFY AP1,Access_Event_Credential = C1

7.3.2.42 Access Zone Object Tests

Many of the tests for the Access Zone object require interactions from other BACnet access control objects as a result of a valid credential being processed. The required and optional BACnet object types are shown in Figure 7-3-2-42.

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

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

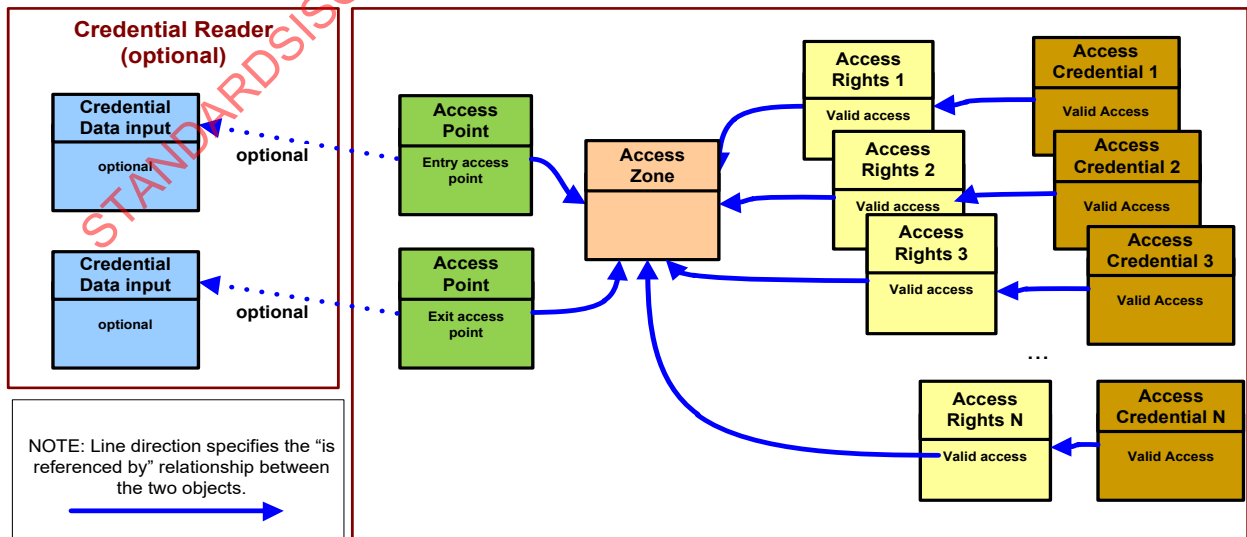


Figure 7-3-2-42: Objects used for testing the Access Zone object

7.3.2.42.1 Occupancy State Test

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

Test Concept: Adjust the occupancy count of the zone to levels above, at, and below the Occupancy_Upper_Limit and Occupancy_Lower_Limit and verify that the Occupancy_State has the appropriate value.

Configuration Requirements: See Clause 7.3.2.42. This test requires the following additional configuration:

- a) The Access Zone shall not be out of service.
- b) Occupancy_Count_Enable shall have a value of TRUE
- c) The Occupancy_Upper_Limit shall have a value X where $X > 0$.
- d) If the property is supported, the Occupancy_Lower_Limit shall have a value Y where $0 < Y < X$.

Test Steps:

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

1. WRITE Adjust_Value = 0
2. VERIFY Occupancy_Count = 0
3. WRITE Adjust_Value = X-1
4. VERIFY Occupancy_Count = (X-1)
5. VERIFY Occupancy_State = NORMAL

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

6. WRITE Adjust_Value = 1
7. VERIFY Occupancy_Count = X
8. VERIFY Occupancy_State = AT_UPPER_LIMIT
9. WRITE Adjust_Value = 1
10. VERIFY Occupancy_Count = X+1
11. VERIFY Occupancy_State = ABOVE_UPPER_LIMIT

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

12. WRITE Adjust_Value = 0
13. VERIFY Occupancy_Count = 0
14. VERIFY Occupancy_State = BELOW_LOWER_LIMIT
15. WRITE Adjust_Value = Y
16. VERIFY Occupancy_Count = Y
17. VERIFY Occupancy_State = AT_LOWER_LIMIT
18. WRITE Adjust_Value = 1
19. VERIFY Occupancy_Count = (Y+1)
20. VERIFY Occupancy_State = NORMAL

-- verify occupancy state when occupancy counting is disabled

21. WRITE Occupancy_Count_Enable = FALSE
22. VERIFY Occupancy_State = DISABLED

7.3.2.42.2 Occupancy Counting Test

Purpose: To verify the occupancy counting functionality.

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

Configuration Requirements: See Clause 7.3.2.42. This test requires the following additional configuration:

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

Test Steps:

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

1. WRITE Adjust_Value = 0
2. VERIFY Occupancy_Count = 0
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY Occupancy_Count = 1
5. MAKE (present credential C2 at credential reader for access point AP1)
6. VERIFY Occupancy_Count = 2

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

7. MAKE (present credential C1 at credential reader for access point AP2)
8. VERIFY Occupancy_Count = 1
9. MAKE (present credential C2 at credential reader for access point AP2)
10. VERIFY Occupancy_Count = 0

7.3.2.42.3 Keeping Track of Credentials Test

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

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

Configuration Requirements: See Clause 7.3.2.42. This test requires the following additional configuration:

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

Test Steps:

-- verify that presenting a credential at an entry access point puts it in the Credentials_In_Zone list

1. VERIFY Credentials_In_Zone = ()
2. MAKE (present credential C1 at credential reader for access point AP1)
3. VERIFY Credentials_In_Zone = (C1)
4. VERIFY Last_Credential_Added = C1
5. VERIFY Last_Credential_Added_Time = (the time that credential C1 was presented at AP1)
6. MAKE (present credential C2 at credential reader for access point AP1)
7. VERIFY Credentials_In_Zone = (C1,C2)
8. VERIFY Last_Credential_Added = C2
9. VERIFY Last_Credential_Added_Time = (the time that credential C2 was presented at AP1)

-- verify that presenting a credential at an exit access point removes it from the Credentials_In_Zone list

10. MAKE (present credential C1 at credential reader for access point AP2)
11. VERIFY Credentials_In_Zone = (C2)
12. VERIFY Last_Credential_Removed = C1
13. VERIFY Last_Credential_Removed_Time = the time that credential C1 was presented at AP2
14. MAKE (present credential C2 at credential reader for access point AP2)
15. VERIFY Credentials_In_Zone = ()
16. VERIFY Last_Credential_Removed = C2
17. VERIFY Last_Credential_Removed_Time = the time that credential C2 was presented at AP2

7.3.2.42.4 Passback Mode Test

Purpose: To verify the passback functionality and passback exemption.

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

Configuration Requirements: See Clause 7.3.2.42. This test requires the following additional configuration:

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

Test Steps:

-- verify soft passback mode

1. MAKE (Passback_Mode = SOFT_PASSBACK)
2. READ EventTag = AP1.Access_Event_Tag
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1.Access_Event = GRANTED
5. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented)
6. VERIFY AP1.Access_Event_Credential = C1
7. VERIFY AP1.Access_Event_Tag = (EventTag + 1)
8. MAKE (present credential C1 at credential reader for access point AP1)
9. VERIFY AP1.Access_Event = GRANTED
10. VERIFY (AP1.Access_Event_Time = (the time that credential C1 was presented most recently))
11. VERIFY AP1.Access_Event_Credential = C1
12. VERIFY AP1.Access_Event_Tag = (EventTag + 2)
13. MAKE (present credential C1 at credential reader for access point AP2)

-- verify hard passback mode

14. MAKE (Passback_Mode = HARD_PASSBACK)
15. READ EventTag = AP1.Access_Event_Tag
16. MAKE (present credential C1 at credential reader for access point AP1)
17. VERIFY AP1.Access_Event = GRANTED
18. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
19. VERIFY AP1.Access_Event_Credential = C1
20. VERIFY AP1.Access_Event_Tag = EventTag + 1
21. MAKE (present credential C1 at credential reader for access point AP1)
22. VERIFY AP1.Access_Event = DENIED_PASSBACK
23. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
24. VERIFY AP1.Access_Event_Credential = C1
25. VERIFY AP1.Access_Event_Tag = (EventTag + 2)

-- verify passback timeout

26. WAIT (Passback_Timeout)
27. MAKE (present credential C1 at credential reader for access point AP1)
28. VERIFY AP1.Access_Event = GRANTED
29. VERIFY (AP1.Access_Event_Time = the time that credential C1 was presented)
30. VERIFY (AP1.Access_Event_Credential = C1)

-- verify hard passback off

31. MAKE (Passback_Mode = PASSBACK_OFF)
32. READ EventTag = AP1.Access_Event_Tag
33. MAKE (present credential C1 at credential reader for access point AP1)
34. VERIFY AP1.Access_Event = GRANTED
35. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
36. VERIFY AP1.Access_Event_Credential = C1

37. VERIFY AP1.Access_Event_Tag = EventTag + 1
38. MAKE (present credential C1 at credential reader for access point AP1)
39. VERIFY AP1.Access_Event = GRANTED
40. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
41. VERIFY (AP1.Access_Event_Credential = C1
42. VERIFY AP1.Access_Event_Tag = (EventTag + 2)

-- verify passback exemption

43. MAKE (Passback_Mode = HARD_PASSBACK)
44. MAKE (C1, Authorization_Exemption = (PASSBACK))
45. READ EventTag = AP1.Access_Event_Tag
46. MAKE (present credential C1 at credential reader for access point AP1)
47. VERIFY AP1.Access_Event = GRANTED
48. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
49. VERIFY AP1.Access_Event_Credential = C1
50. VERIFY AP1.Access_Event_Tag = EventTag + 1
51. MAKE (present credential C1 at credential reader for access point AP1)
52. VERIFY AP1.Access_Event = GRANTED
53. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
54. VERIFY AP1.Access_Event_Credential = C1
55. VERIFY AP1.Access_Event_Tag = (EventTag + 2)

7.3.2.43 Access Rights Object Tests

Many of the tests for the Access Rights object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in Figure 7-3-2-43.

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

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

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

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

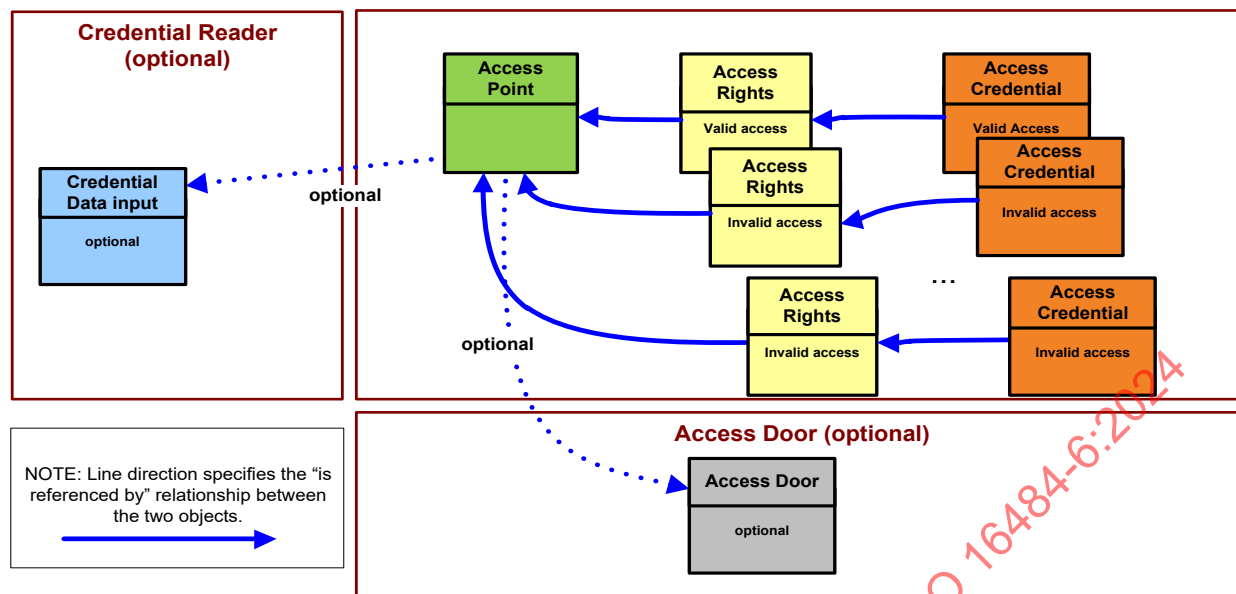


Figure 7-3-2-43: Objects used for testing the Access Rights object

7.3.2.43.1 Enable Test

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

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

Configuration Requirements: See Clause 7.3.2.43. This test requires the following additional configuration:

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

Test Steps:

- verify access granted with this access rights object when enable property is TRUE
- 1. MAKE (present credential C1 at credential reader for access point AP1)
- 2. VERIFY AP1,Access_Event = GRANTED
- 3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
- 4. VERIFY AP1,Access_Event_Credential = C1
- verify access denied with this access rights object when Enable property is FALSE
- 5. WRITE AR1, Enable = FALSE
- 6. MAKE (present credential C1 at credential reader for access point AP1)
- 7. VERIFY AP1,Access_Event = DENIED_NO_ACCESS_RIGHTS
- 8. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
- 9. VERIFY AP1,Access_Event_Credential = C1

7.3.2.43.2 Negative Rules Test

Purpose: To verify that negative access rules explicitly disable access to an access point.

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

Configuration Requirements: See Clause 7.3.2.43. This test requires the following additional configuration:

- An access point shall be represented by Access Point object AP1.

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

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = DENIED_POINT_NO_ACCESS_RIGHTS
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1

7.3.2.43.3 Positive Access Rules Test

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

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

Configuration Requirements: See Clause 7.3.2.43. This test requires the following additional configuration:

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

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1

7.3.2.43.4 Accompaniment Test

Purpose: To verify that the accompaniment functionality works properly.

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

Configuration Requirements: See Clause 7.3.2.43. This test requires the following additional configuration:

- a) An access point shall be represented by Access Point object AP1. If the Accompaniment_Time property is supported, it shall be set to a value > 0.
- b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
- c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1 that requires accompaniment.
- d) An active credential which has access rights which allow it to accompany a credential with access rights specified by AR1 through AP1, shall be represented by Access Credential object C2.
- e) An active credential which has valid access rights to AP1 but which does not meet the accompaniment requires of AR1, shall be represented by Access Credential object C3.

Test Steps:

-- valid access through the access point

1. READ Tag = Access_Event_Tag
2. MAKE (present credential C1 at credential reader for access point AP1)
3. MAKE (present credential C2 at credential reader for access point AP1)
4. VERIFY AP1,Access_Event = GRANTED
5. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access_Event_Credential = C1
7. VERIFY AP1,Access_Event_Tag = (Tag + 1)

- no accompaniment presented
8. MAKE (present credential C1 at credential reader for access point AP1)
 9. WAIT AP1,Accompaniment_Time
 10. VERIFY AP1,Access_Event = DENIED_NO_ACCOMPANIMENT
 11. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
 12. VERIFY AP1,Access_Event_Credential = C1
 13. VERIFY Access_Event_Tag = (Tag + 2)

-- Invalid accompaniment

14. MAKE (present credential C1 at credential reader for access point AP1)
15. MAKE (present credential C3 at credential reader for access point AP1)
16. VERIFY AP1,Access_Event = DENIED_INCORRECT_ACCOMPANIMENT
17. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
18. VERIFY AP1,Access_Event_Credential = C1
19. VERIFY AP1,Access_Event_Tag = (Tag + 3)

7.3.2.44 Access Credential Object Tests

Many of the tests for the Access Credential object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in Figure 7-3-2-44.

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

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

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

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

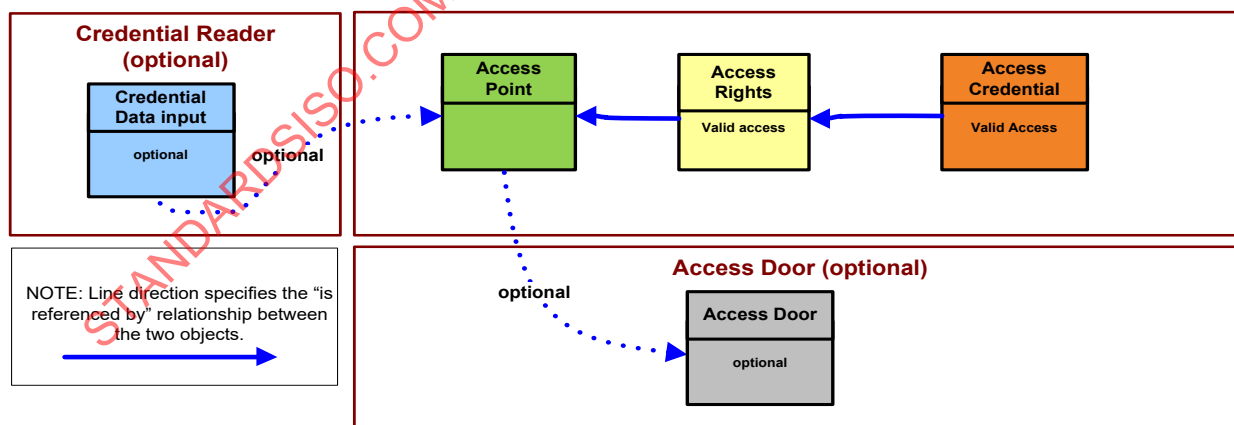


Figure 7-3-2-44: BACnet Objects used for testing the Access Credential object

7.3.2.44.1 Credential Status, Credential Disable and Reason for Disable Test

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

Test Concept: The credential status is set to INACTIVE, and the corresponding reason or reasons are written to the Reason_For_Disable list.

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

- a) An access credential shall be represented by Access Credential C1.
- b) The Credential_Status property shall have the value ACTIVE.
- c) The Reason_For_Disable property shall be empty.

Notes to Tester: This test only verifies the most common disable reasons (DISABLED_MANUAL, DISABLED_NOT_YET_ACTIVE, DISABLED_EXPIRED).

Test Steps:

-- test DISABLED_MANUAL

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

-- test DISABLED_NOT_YET_ACTIVE

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

-- test DISABLED_EXPIRED

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

7.3.2.44.2 Activation Time and Expiration Time Test

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

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

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

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

Test Steps:

-- test activation time

1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. MAKE (set Activation_Time = time > current time)
4. VERIFY Credential_Status = INACTIVE

5. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
6. MAKE (set Activation_Time = time < current time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()

-- test expiration time

9. VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ()
11. MAKE (Expiration_Time = time < current time)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
14. MAKE (Expiration_Time = time > current time)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ()

7.3.2.44.3 Disabled Access Rights Test

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

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

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

Test Steps:

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

7.3.2.44.4 Days Remaining and Uses Remaining Test

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

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

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

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

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

Test Steps:

-- test day remaining

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

7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()

-- test uses remaining

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

7.3.2.44.5 Absentee Limit Test

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

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

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

- a) Absentee_Limit ≥ 0
- b) The Credential_Status property shall have the value ACTIVE.
- c) The Reason_For_Disable property shall be empty.
- d) Days_Remaining shall have a value > 0 .
- e) Last_Use_Time shall be set to a valid date and time.

Test Steps:

1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ()
3. TRANSMIT UTCTimeSynchronization-Request, 'Time' = (any day and time greater than Absentee_Limit days)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_INACTIVITY)

-- clear the condition

6. MAKE (set Absentee_Limit > number of days since Last_Use_Time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ()

7.3.2.44.6 Last Access Point, Last Use Time and Last Access Event Test

Purpose: To verify that the Last Access Point, Last Access Event, and Last Use Time properties are updated when this credential is used at an access point.

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

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

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. VERIFY Last_Access_Point = AP1
6. VERIFY Last_Use_Time = AP1,Access_Event_Time
7. VERIFY Last_Access_Event = GRANTED

7.3.2.44.7 Extended Time Enable Test

Purpose: To verify that a credential used at an access point with the extended flag set to TRUE results in the corresponding door to pulse open for a Pulse_Extended period of time as defined in the door object.

Configuration Requirements: See Clause 7.3.2.44. This test requires the following additional configuration:

- An access door shall be represented by Access Door object AD1 and Door_Delay_Time shall be 0.
- An access point, which controls AD1, shall be represented by Access Point object AP1.
- An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
- An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.
- The Extended_Time_Enable property shall be FALSE.

Test Steps:

-- verify that PULSE_UNLOCK is written when the extended time enable is FALSE

- MAKE (present credential C1 at credential reader for access point AP1)
- VERIFY AP1,Access_Event = GRANTED
- VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
- VERIFY AP1,Access_Event_Credential = C1
- BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = PULSE_UNLOCK

-- verify that EXTENDED_PULSE_UNLOCK is written when the extended time enable is TRUE

- WRITE Extended_Time_Enable = TRUE
- MAKE (present credential C1 at credential reader for access point AP1)
- VERIFY AP1,Access_Event = GRANTED
- VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
- VERIFY AP1,Access_Event_Credential = C1
- BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = EXTENDED_PULSE_UNLOCK

7.3.2.45 Credential Data Input Object Tests

The Credential Data Input object type represents a device or process that reads an authentication factor from a physical device such as a card reader, keypad, or biometric device. There are countless variations and authentication formats supported for these devices. As such, there is not a standard format or device configuration that can be mandated for these tests.

The vendor must configure the IUT such that the Credential Data Input device can read an authentication factor from the corresponding physical device, including setting the Supported_Formats[1] property to the correct authentication factor format (Figure 7-3-2-45). This configuration is considered to be a local matter and will not be tested.

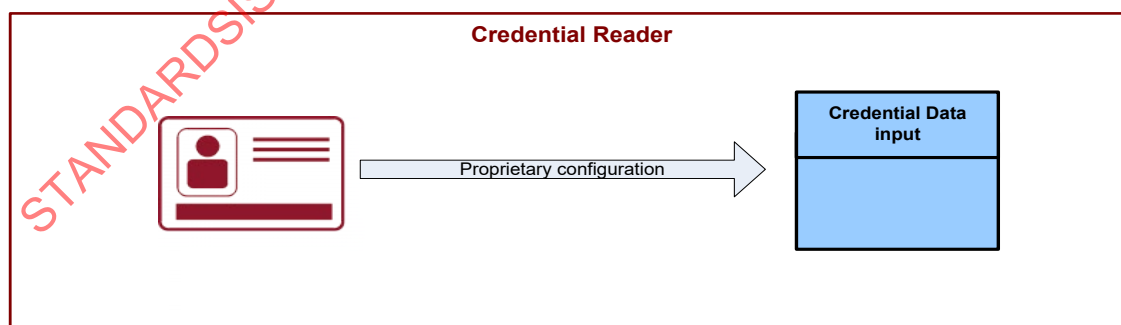


Figure 7-3-2-45: Credential Data Input configuration

7.3.2.45.1 Return From Out Of Service Undefined Test

Purpose: To verify that the Present_Value Format-Type becomes undefined when out of service is set to false.

Configuration Requirements: See Clause 7.3.2.45. This test requires the following additional configuration:

- The Out_Of_Service property shall be TRUE.

Test Steps:

1. WRITE Out_Of_Service = FALSE
2. VERIFY Present_Value. Format-Type = UNDEFINED
3. VERIFY Present_Value. Format-Class = 0

7.3.2.45.2 Read Valid Authentication Factor Test

Purpose: To verify that Present_Value is set to the proper value when an authentication factor with a recognized format is read at the corresponding physical device.

Configuration Requirements: See Clause 7.3.2.45. This test requires the following additional configuration:

- a) The Out_Of_Service property shall be FALSE.
- b) Two authentication factors, AF1 and AF2, shall be provided which can be read by the physical device which have the format specified in Supported_Formats[1].

Test Steps:

-- test AF1

1. MAKE (present AF1 at the credential reader)
2. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type
3. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
4. VERIFY Present_Value. Value = the authentication format value of AF1

-- test AF2

5. MAKE (present AF2 at the credential reader)
6. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type
7. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
8. VERIFY Present_Value. Value = the authentication format value of AF2

7.3.2.46 Network Port Object Tests

Configuration Requirements: In addition to the requirements listed for each test, the Network Port object which is being tested shall be configured and operating and have no changes pending, unless required by the specific test's specification.

7.3.2.46.1 Network Port Configuration Tests

These tests verify that Network Port objects reflect the configuration in use, and for writable ones, change the network configuration.

7.3.2.46.1.1 Configure Network Through Network Port Object Test

Purpose: This test verifies that Network Port properties control aspects of the network configuration as expected.

Test Concept: Given the complexity of the Network Port object, and the impact changes to the Network Port has on the test network, this test is provided to allow testing of the Network Port functionality as the network is reconfigured for other tests. The Network Port object is modified to meet the conditions of the new test network setup. The changes are activated, the TD is reconfigured to match, and communication with the IUT is re-verified. The configuration of the network is expected to be tested in more detail as the other datalink tests are applied.

Configuration Requirements: The test network is configured such that the TD and IUT can communicate, but the configuration does not match the target network configuration. P1 through PN are Network Port properties that need to be written in order to transition the network from the current setup to the target network setup. This set of properties shall be selected from the set of the properties that are writable in the IUT.

Test Steps:

1. REPEAT P = P1 ... PN DO {
 WRITE P = (NV: the value required for the target network setup)
 VERIFY P = NV
}
2. VERIFY Changes_Pending = TRUE
3. REPEAT P = P1 ... PN DO {
 CHECK (the new value for P is not in use by the network port, unless the new value is the same as the

- old value)
- }
- 4. TRANSMIT ReinitializeDevice-Request
 - 'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
 - 'Password' = (any valid password)
- 5. RECEIVE BACnet-SimpleACK-PDU
- 6. MAKE(change the TD network setup and the network setup of all other devices on the network to match the target network setup)
- 7. WAIT **Activate Changes Fail Time**
- 8. VERIFY Changes_Pending = FALSE

7.3.2.46.1.2 Verify Network Configuration Through Network Port Object Test

Purpose: This test verifies that Network Port properties correctly reflect aspects of the network configuration as expected.

Test Concept: Given the complexity of the Network Port object, and the impact changes to the Network Port has on the test network, this test is provided to allow testing of the Network Port functionality as the network is reconfigured for other tests. The IUT's network configuration is modified to meet the conditions of the new test network setup. The TD is reconfigured to match, and communication with the IUT is re-verified. The Network Port object is then checked to ensure it reflects the new network setup.

Test Steps:

1. MAKE(configure the network, including reconfiguring the TD, IUT, and other devices on the network)
2. CHECK(that the value of each of the present Network Port properties which applies to the associated data link reflects the current network setup)

7.3.2.46.1.3 Network Port Non-Volatility Properties Test

Purpose: This test verifies that after Network Port properties are changed and activated, the revised value is maintained through a power failure and device restart.

Test Concept: Write one or more properties, P1 ... PN, of a Network Port object which are required for proper operation of the network port. If any of the properties utilize the pending changes functionality, activate the changes. Restart the IUT device by temporarily removing power. When the device has resumed operation after that restart, verify that the new values for the properties were maintained across the reset and are in use by the port.

Test Steps:

1. REPEAT P = P1 ... PN DO {
 - WRITE P = (a new value different from the property's current value)
- }
2. IF any of the properties utilize the pending change functionality THEN
 - VERIFY Changes_Pending = TRUE
 - TRANSMIT ReinitializeDevice-Request
 - 'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
 - 'Password' = (any valid password)
 - RECEIVE BACnet-SimpleACK-PDU
 - MAKE(reconfigure the TD and other devices on the network to the new network settings)
 - WAIT **Activate Changes Fail Time**
- ELSE
 - VERIFY Changes_Pending = FALSE
3. REPEAT P = P1 ... PN DO {
 - VERIFY P = (the new value for the property)
- }
4. MAKE (the IUT power cycle to reinitialize)
5. REPEAT P = P1 ... PN DO {
 - VERIFY P = (the new value for the property)
 - CHECK (that the value for P is in use by the network port)
- }

7.3.2.46.1.4 Network Port Configuration Conflict Test

Purpose: To verify that either multiple clients can write to a Network Port object at the same time, or the CONFIGURATION_IN_PROGRESS error is reported.

Test Concept: The TD simulates 2 devices (TD and TD2), attempting to write to properties in a Network Port object, O1. The IUT must either accept the second write, or the IUT returns an error with an Error Class of DEVICE and an error code of CONFIGURATION_IN_PROGRESS. Finally, the Network Port's changes are activated and then verified.

Test Steps:

1. TRANSMIT WriteProperty-Request,
SOURCE = TD,
'Object Identifier' = O1,
'Property Identifier' = (P1: a writable property which utilizes the pending changes functionality),
'Property Value' = (any valid value),
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT WriteProperty-Request,
SOURCE = TD2,
'Object Identifier' = O1,
'Property Identifier' = (P2: a writable property which utilizes the pending changes functionality, and is different than P1, if possible),
'Property Value' = (any valid value),
4. RECEIVE BACnet-SimpleACK-PDU
DESTINATION = IUT
| BACnet-Error-PDU
'Error Class' = DEVICE,
'Error Code' = CONFIGURATION_IN_PROGRESS
5. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = ACTIVATE_CHANGES,
'Password' = (any valid password)
6. RECEIVE BACnet-SimpleACK-PDU
7. MAKE(reconfigure the TD and other devices on the network to the new network settings)
8. WAIT **Activate Changes Fail Time**
9. IF P1 is a different property than P2 THEN
CHECK(P1's value is in use by the network port)
10. CHECK(P2's value is in use by the network port)

7.3.2.46.2 Network-Number-Is Updates Network_Number_Quality Test

Purpose: To verify that Network_Number_Quality is updated when the IUT learns its Network_Number from Network-Number-Is.

Test Concept: Write 0 to Network_Number to set Network_Number_Quality to UNKNOWN. Send a Network-Number-Is message to the IUT indicating that the Network_Number is learned and verify that Network_Number_Quality changes to LEARNED. Send a Network-Number-Is message to the IUT indicating that the Network_Number is configured and verify that Network_Number_Quality changes to LEARNED_CONFIGURED. Write 0 to Network_Number and verify that Network_Number_Quality changes to UNKNOWN.

Configuration Requirements: Select a Network Port object, O1, which is enabled and has a writable Network_Number. Connect the TD to the network associated with Network Port O1. This test shall be skipped if the TD cannot be directly connected to the IUT's network.

Test Steps:

- set network number quality to UNKNOWN
1. WRITE Network_Number = 0
 2. TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = ACTIVATE_CHANGES,
'Password' = (any valid password)
 3. RECEIVE BACnet-SimpleACK-PDU

4. **WAIT Activate Changes Fail Time**
5. **VERIFY** Network_Number_Quality = UNKNOWN
- make IUT learn the network number
6. **TRANSMIT** Network-Number-Is
 DESTINATION = LOCAL_BROADCAST | IUT,
 'Network Number' = (N1: any valid value)
 'Flag' = 0 -- learned
7. **VERIFY** Network_Number_Quality = LEARNED
8. **VERIFY** Network_Number = N1
- make IUT learn the network number from a configure device
9. **TRANSMIT** Network-Number-Is
 DESTINATION = LOCAL_BROADCAST | IUT,
 'Network Number' = (N2: any valid value)
 'Flag' = 1 -- configured
10. **VERIFY** Network_Number_Quality = LEARNED_CONFIGURED
11. **VERIFY** Network_Number = N2
- configure the IUT's network number
12. **WRITE** Network_Number = (N3: any valid value other than 0)
13. **TRANSMIT** ReinitializeDevice-Request,
 'Reinitialized State of Device' = ACTIVATE_CHANGES,
 'Password' = (any valid password)
14. **RECEIVE** BACnet-SimpleACK-PDU
15. **WAIT Activate Changes Fail Time**
16. **VERIFY** Network_Number_Quality = CONFIGURED
17. **TRANSMIT** Network-Number-Is
 DESTINATION = LOCAL_BROADCAST | IUT,
 'Network Number' = (N4: any valid value)
 'Flag' = 1 -- configured
18. **VERIFY** Network_Number_Quality = CONFIGURED
19. **VERIFY** Network_Number = N3
- revert network number quality to UNKNOWN
20. **WRITE** Network_Number = 0
21. **TRANSMIT** ReinitializeDevice-Request,
 'Reinitialized State of Device' = ACTIVATE_CHANGES,
 'Password' = (any valid password)
22. **RECEIVE** BACnet-SimpleACK-PDU
23. **WAIT Activate Changes Fail Time**
24. **VERIFY** Network_Number_Quality = UNKNOWN

7.3.2.46.3 Network Port Command Tests

7.3.2.46.3.1 IDLE Command Rejected

Purpose: To verify that the Command property does not accept write of IDLE.

Test Concept: Write IDLE to the command property and verify that an error-class of PROPERTY with an error-code of VALUE_OUT_OF_RANGE is returned.

Configuration Requirements: Execute the test against a Network Port object with a writable Command property. This test shall be skipped if the IUT does not support the Command property. The Network Port object shall have no pending changes.

Test Steps:

1. **TRANSMIT** WriteProperty-Request,
 'Object Identifier' = (a Network Port object),

- 'Property Identifier' = Command,
 'Property Value' = IDLE
2. RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE

7.3.2.46.3.2 DISCARD_CHANGES Command Test

Purpose: To verify that the Network Port discards pending changes when the Command DISCARD_CHANGES is received.

Test Concept: Write values to one or more properties, P1 .. Px, which utilize the pending changes functionality. Write DISCARD_CHANGES to the Command property and verify that the properties have reverted to their previous values.

Configuration Requirements: Execute the test on a Network Port object which supports the DISCARD_CHANGES command. This test shall be skipped if the IUT does not support the DISCARD_CHANGES command.

Test Steps:

- save initial values of the properties and change each one to a new value
1. REPEAT I = (in the range 1 through the number of properties being written) DO {
 V[I] = READ P[I]
 WRITE P[I] = (a value different than V[I], if possible)
 }
- discard the changes
2. WRITE Command = DISCARD_CHANGES
 3. WAIT **Activate Changes Fail Time**
- verify that no changes are pending any more
4. VERIFY Changes_Pending = FALSE
 5. VERIFY Command = IDLE
- verify that the properties have reverted in value, and that the old value remains in use by the port
6. REPEAT I = (in the range 1 through the number of properties being written) DO {
 VERIFY P[I] = V[I]
 CHECK(the value V[I] is in use by the network port)
 }
- command the device to activate any changes which should have no effect
7. TRANSMIT ReinitializeDevice-Request
 'Reinitialized State of Device' = WARMSTART | ACTIVATE_CHANGES
 'Password' = (any valid password)
 8. RECEIVE BACnet-SimpleACK-PDU
 9. MAKE(reconfigure the TD and other devices on the network to the new network settings)
 10. WAIT **Activate Changes Fail Time**
 11. VERIFY Command = IDLE
- verify that the properties retain their original values, and that that value remains in use by the port
12. REPEAT I = (in the range 1 through the number of properties being written) DO {
 VERIFY P[I] = V[I]
 CHECK(the value V[I] is in use by the network port)
 }

7.3.2.46.3.3 RENEW_FD_REGISTRATION Command Tests

7.3.2.46.3.3.1 RENEW_FD_REGISTRATION Command Test

Purpose: To verify that the Network Port attempts to renew its Foreign Device registration when commanded to do so.

Test Concept: Starting with a Network Port object which is already registered with a BBMD as a Foreign Device, command the Network Port to RENEW_FD_REGISTRATION but do have the TD respond. Verify that the Network Port

attempts to renew its Foreign Device registration. Wait until the Network Port has completed its attempt and verify that the Reliability has been set to RENEW_FD_REGISTRATION_FAILURE. Command the Network Port to RENEW_FD_REGISTRATION and have the TD respond. Verify that the attempt succeeds and that Reliability is reset to NO_FAULT_DETECTED. Command the Network Port to RENEW_FD_REGISTRATION and have the TD respond. Verify that the attempt succeeds.

Configuration Requirements: Configure a Network Port for BACnet/IP or BACnet/IPv6 in FOREIGN mode. Allow the IUT to complete its registration with the TD acting as the BBMD before continuing. If the IUT does not support registering as a Foreign Device, or the IUT does not support the RENEW_FD_REGISTRATION command, then this test shall be skipped. The Network Port object shall have no pending changes.

Test Steps:

- make sure our initial conditions are good
 1. VERIFY Changes_Pending = FALSE
 2. VERIFY Reliability = NO_FAULT_DETECTED
 3. VERIFY BACnet_IP_Mode = FOREIGN
- request the renewal, and wait for it to timeout
 4. WRITE Command = RENEW_FD_REGISTRATION
 5. BEFORE **Internal Processing Fail Time**
 RECEIVE Register-Foreign-Device
 'Time-to-Live' = FD_Subscription_Lifetime
 6. WAIT **Foreign Device Registration Fail Time**
 7. VERIFY Reliability = RENEW_FD_REGISTRATION_FAILURE
 8. VERIFY Command = IDLE
- re-request the renewal, and allow it to succeed
 9. WRITE Command = RENEW_FD_REGISTRATION
 10. BEFORE **Internal Processing Fail Time**
 RECEIVE Register-Foreign-Device
 'Time-to-Live' = FD_Subscription_Lifetime
 11. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion
 12. VERIFY Reliability = NO_FAULT_DETECTED
 13. VERIFY Command = IDLE
- 14. WAIT (a random amount of time significantly less than FD_Subscription_Lifetime)
- re-request the renewal, and allow it to succeed
 15. WRITE Command = RENEW_FD_REGISTRATION
 16. BEFORE **Internal Processing Fail Time**
 RECEIVE Register-Foreign-Device
 'Time-to-Live' = FD_Subscription_Lifetime
 17. TRANSMIT BVLC-Result,
 'Result Code' = Successful completion
 18. VERIFY Reliability = NO_FAULT_DETECTED
 19. VERIFY Command = IDLE

7.3.2.46.3.3.2 RENEW_FD_REGISTRATION Command Failure Test

Purpose: To verify that Network Port object respond to RENEW_FD_REGISTRATION commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is either not an BACnet/IP nor BACnet/IPv6 port or which is not in FOREIGN mode, to renew its FD subscription. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE_OUT_OF_RANGE.

Configuration Requirements: Select a Network Port which is not in FOREIGN mode. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

- ```
-- make sure our initial conditions are good
1. IF Network_Type is IPV4 or IPV6 THEN
2. VERIFY BACnet_IP_Mode <> FOREIGN

3. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Network Port object),
 'Property Identifier' = Command,
 'Property Value' = RENEW_FD_SUBSCRIPTION,
4. RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE
5. VERIFY Command = IDLE
```

### 7.3.2.46.3.4 RESTART\_SLAVE\_DISCOVERY Command Tests

#### 7.3.2.46.3.4.1 RESTART\_SLAVE\_DISCOVERY Command Test

Purpose: To verify that the Network Port restarts the slave discovery process when commanded to.

Test Concept: Starting with a Network Port object which is configured as an MS/TP Slave Proxy, command the Network Port to RESTART\_SLAVE\_DISCOVERY. Verify that the IUT restarts slave discovery.

Test Configuration Configure an MSTP Network Port object to act as an MS/TP Slave Proxy with Auto\_Slave\_Discovery set to TRUE. Configure the TD to act as an MS/TP slave. Delay the start of the test until after the IUT has completed its initial slave confirmation. If the IUT does not support automatic slave discovery, or the IUT does not support the RESTART\_SLAVE\_DISCOVERY command, then this test shall be skipped.

Notes to Tester: The IUT may interrogate the slave addresses in any order. The IUT is allowed to generate any other traffic during the test, including, and is not limited to reading property values from the devices it finds.

Test Steps:

- ```
-- make sure our initial conditions are good
1. VERIFY Network_Type = MSTP
2. VERIFY Slave_Proxy_Enable = TRUE

-- request the renewal, and wait for it to timeout
3. WRITE Command = RESTART_SLAVE_DISCOVERY
4. BEFORE Slave Proxy Confirm Interval
   REPEAT addr=(all MS/TP addresses excluding the IUT's MAC address) DO {
       RECEIVE DESTINATION=addr, SRC=IUT
       ReadProperty-Request,
       'Object Identifier' = (DEVICE,4194303),
       'Property Identifier' = Protocol_Services_Supported
   }
5. VERIFY Command = IDLE
```

7.3.2.46.3.4.2 RESTART_SLAVE_DISCOVERY Command Failure Test

Purpose: To verify that Network Port object respond to RESTART_SLAVE_DISCOVERY commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is not acting as a slave proxy to RESTART_SLAVE_DISCOVERY. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE_OUT_OF_RANGE.

Configuration Requirements: Select a Network Port which is not configured to be a slave proxy. If the IUT supports slave proxy functionality, this test shall be skipped as the standard does not specify how the IUT should respond when slave proxy is supported but not enabled. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Network Port object),
 'Property Identifier' = Command,
 'Property Value' = RESTART_SLAVE_DISCOVERY
2. IF Network_Type is MSTP THEN
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = OPTIONAL_FUNCTIONALITY_SUPPORTED
 ELSE
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE
3. VERIFY Command = IDLE

7.3.2.46.3.5 RENEW_DHCP Command Tests

7.3.2.46.3.5.1 RENEW_DHCP Command Test

Purpose: To verify that the Network Port attempts to renew its addressing information when commanded to.

Test Concept: Starting with a Network Port object which is configured to use DHCP and which supports the RENEW_DHCP command, command the port to RENEW_DHCP. Verify that the IUT requests a renewal of addressing information.

Configuration Requirements: Select a Network Port which is an IP or IPv6 port setup for auto-addressing. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

- make sure our initial conditions are good
1. IF Network_Type = IPV4 THEN
 VERIFY IP_DHCP_Enable = TRUE
 2. IF Network_Type = IPV6 THEN
 VERIFY IPv6_Auto_Addressing_Enable = TRUE
- request the renewal, and wait for it to timeout
3. WRITE Command = RENEW_DHCP
 4. CHECK(that the IUT requested a renewal of its addressing information)
 5. VERIFY Command = IDLE

7.3.2.46.3.5.2 RENEW_DHCP Command Failure Test

Purpose: To verify that Network Port object responds to RENEW_DHCP commands when the command is not supported / enabled.

Test Concept: Attempt to command a Network Port which is either not an BACnet/IP nor BACnet/IPv6 port or which is not in configured for auto-addressing. Verify that the attempt fails with an Error Class of PROPERTY and an error code of VALUE_OUT_OF_RANGE.

Configuration Requirements: Select a Network Port which is not an IP or IPv6 port setup for auto-addressing. If the IUT does not support the Command property, then this test shall be skipped.

Test Steps:

- make sure our initial conditions are good
1. IF Network_Type is IPV4 and IP_DHCP_Enable is present THEN
 VERIFY IP_DHCP_ENABLE = FALSE
 2. IF Network_Type is IPV6 and IPV6_Auto_Addressing_Enabled is present THEN
 VERIFY IPV6_Auto_Addressing_Enabled = FALSE

3. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Network Port object),
 'Property Identifier' = Command,
 'Property Value' = RENEW_DHCP
4. IF Network_Type is IPV4 or IPV6 THEN
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
 ELSE
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE
5. VERIFY Command = IDLE

7.3.2.46.3.6 RESTART_AUTO-NEGOTIATION Command Tests

7.3.2.46.3.6.1 RESTART_AUTO-NEGOTIATION Command Test

Purpose: To verify that the Network Port attempts to renegotiate its link speed when commanded to.

Test Concept: Starting with a Network Port object which is configured to auto-negotiate its link speed and which supports the RESTART_AUTONEGOTIATION command, is commanded to restart auto-negotiation. The link speed is changed, and it is verified that the IUT performs link speed negotiation and is able to communicate with the new speed.

Configuration Requirements: The TD and IUT are connected on a network for which the IUT performs link speed auto-negotiation. The Network Port object for the port is configured to perform auto-negotiation, If the IUT does not support the RESTART_AUTONEGOTIATION command, then this test shall be skipped.

Test Steps:

-- make sure our initial conditions are good

1. VERIFY Link_Speed_Autonegotiate = TRUE

-- request the renewal, and wait for it to timeout

3. WRITE Command = RESTART_AUTONEGOTIATION
4. MAKE(change the link speed for the network or link)
5. WAIT Auto Negotiation Fail Time
6. CHECK(check any external indications that the new link speed was detected)
7. VERIFY Command = IDLE -- the act of validating the command property is sufficient to validate that the command worked

7.3.2.46.3.6.2 RESTART_AUTONEGOTIATION Command Failure Test

Purpose: To verify that Network Port objects respond to the RESTART_AUTO-NEGOTIATION command with the correct error codes when the command is not supported / enabled.

Test Concept: Starting with a Network Port object which is not configured to auto-negotiate its link speed or which does not support the RESTART_AUTO-NEGOTIATION, command it to restart auto-negotiation. Verify that the correct error code is returned.

Configuration Requirements: If the network port support auto-negotiation, disable it. If the IUT does not support the Command property, or all Network Port object support auto-negotiation and it cannot be disabled, then this test shall be skipped.

Test Steps:

-- make sure our initial conditions are good

1. VERIFY Link_Speed_Auto-negotiate = TRUE

-- request the renewal, and wait for it to timeout

2. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Network Port object),
 'Property Identifier' = Command,
 'Property Value' = RESTART_AUTO-NEGOTIATION
3. IF the port does not support auto-negotiation THEN
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
 ELSE
 RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE

7.3.2.46.3.7 DISCONNECT Command Tests

7.3.2.46.3.7.1 DISCONNECT Command Test

Purpose: To verify that the Network Port attempts to disconnect its link speed when commanded to.

Test Concept: Starting with a Network Port object which supports the DISCONNECT command. The port is commanded to disconnect. The disconnection of the link is verified.

Configuration Requirements: The TD and IUT are connected on a network which supports disconnection. If the IUT does not support the DISCONNECT command, then this test shall be skipped.

Test Steps:

- make sure our initial conditions are good
1. VERIFY Network_Type = (a network type that supports disconnection, such as PTP)
2. WRITE Command = DISCONNECT
3. CHECK(that the link was disconnected)

7.3.2.46.3.7.2 DISCONNECT Command Failure Test

Purpose: To verify that Network Port objects respond to the DISCONNECT command with the correct error codes when the command is not supported / enabled.

Test Concept: With a Network Port object for a network which does not support disconnection, command it to disconnect. Verify that the correct error code is returned.

Configuration Requirements: If the IUT does not support the Command property, or all Network Port object support disconnection, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Network Port object),
 'Property Identifier' = Command,
 'Property Value' = DISCONNECT
2. RECEIVE BACnet-Error-PDU
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE

7.3.2.46.3.8 RESTART_PORT Command Tests

7.3.2.46.3.8.1 RESTART_PORT Command Test

Purpose: To verify that the Network Port attempts to restart its port when commanded to.

Test Concept: With a Network Port object which supports the RESTART_PORT command, command the port to restart. The restart of the port is verified.

Configuration Requirements: If the IUT does not support the RESTART_PORT command, then this test shall be skipped.

Test Steps:

- make sure our initial conditions are good
- 1. WRITE Command = RESTART_PORT
- 2. CHECK(that the port was restarted)

7.3.2.46.3.8.2 RESTART_PORT Command Failure Test

Purpose: To verify that Network Port objects respond to the RESTART_PORT command with the correct error codes when the command is not supported.

Test Concept: With a Network Port object which does not support the RESTART_PORT command, command the port to restart. Verify that the correct error code is returned.

Configuration Requirements: If the IUT does not support the Command property, or all Network Port object support disconnection, then this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = (the Network Port object),
 - 'Property Identifier' = Command,
 - 'Property Value' = RESTART_PORT
2. RECEIVE BACnet-Error-PDU
 - 'Error Class' = PROPERTY,
 - 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED

7.3.2.46.3.9 No Commands if Changes_Pending Test

Purpose: To verify that the Network Port disallows commands, except DISCARD_CHANGES, when Changes_Pending.

Test Concept: using Network Port object NP, write values to one or more properties, P1 .. Px, which utilize the pending changes functionality. Write each of the other commands and verify they are rejected.

Configuration Requirements: Execute the test on a Network Port object which supports the Command property.

Test Steps:

- write some properties
- 1. REPEAT P = (P1 .. Px) {
 - WRITE NP, P = (any valid value)
- verify that changes are pending
- 2. VERIFY Changes_Pending = TRUE
- write each supported Command value, except DISCARD_CHANGES
- 3. REPEAT CMD = (all non-IDLE valid values that NP supports except DISCARD_CHANGES) {
 - TRANSMIT WriteProperty-Request
 - 'Object Identifier' = NP
 - 'Property' = Command,
 - 'Property Value' = CMD
 - RECEIVE BACnet-Error-PDU
 - 'Error Class' = PROPERTY,
 - 'Error Code' = INVALID_VALUE_IN_THIS_STATE
- revert the Network Port object

```

4. IF the IUT supports DISCARD_CHANGES THEN {
    WRITE Command = DISCARD_CHANGES
  } ELSE {
    MAKE(the IUT discard its changes)
  }

```

7.3.2.46.4 Hierarchical Network Port Tests

7.3.2.46.4.1 Valid Hierarchy Test

Purpose: To verify that the set of network port objects in the IUT are organized in a valid hierarchy.

Test Concept: Visit each Network Port object which represents a configured application layer port. Ensure that the top Network Port object has a Protocol_Level of BACNET_APPLICATION or NON_BACNET_APPLICATION. Visit each Network Port object in the hierarchy ensuring that the Protocol_Level properties are valid.

Test Steps:

```

1. REPEAT NP = (object id of each Network Port object which has a Protocol_Level of
    BACNET_APPLICATION or NON_BACNET_APPLICATION) DO {
    REPEAT NPx = (object id of each Network Port object in NP's hierarchy) DO {
      PL = READ (Network Port, NPx), Protocol_Level
      IF PL is BACNET_APPLICATION or NON_BACNET_APPLICATION THEN
        ERROR Invalid Protocol_Level in child Network Port object
      IF PL is PHYSICAL THEN
        VERIFY (Network Port, NPx), Reference_Port = 4194303
      }
    }
  }

```

7.3.2.46.4.2 Properties in Referenced Network Port Reflected in Top Network Port Object

Purpose: To verify that properties in referenced Network Port objects are reflected in the top Network Port object.

Test Concept: Visit each Network Port object which represents a configured BACnet application layer port. Visit each Network Port object in the hierarchy ensuring that the properties in the referenced Network Port object exist and have the same value in the top Network Port object.

Test Steps:

```

1. REPEAT NP = (object id of each Network Port object which has a Protocol_Level of
    BACNET_APPLICATION) DO {
  -- verify that the required properties exist for this Network Port object based
  -- on its Network_Type
  REPEAT P = (each required property for NP's Network_Type, see Table 12-72) DO {
    VERIFY (Network Port, NP), P = (any valid value)
  }
  REPEAT NPx = (object id of each Network Port object in NP's hierarchy) DO {
    -- verify that the expected properties exist in the Network Port object based
    -- on its Network_Type and Protocol_Level. In addition, verify that the property
    -- value is inherited into NP (unless already inherited from a different Network Port)
    REPEAT P = (each expected property in NPx based on its Network_Type and
      Protocol_Level as defined in Table 12-73) DO {
      V1 = READ (Network Port, NPx), P
      IF P is not in a higher Network Port object in this hierarchy THEN
        VERIFY (Network Port, NP), P = V1
      }
    }
  }
}

```

7.3.2.46.4.3 Changes Reflected in Top Network Port Object

Purpose: To verify that changing properties in child Network Port objects result in the new property values reflected in the top Network Port object.

Test Concept: Write a writable, inheritable property within a Network Port's hierarchy and verify that the new value is reflected in the top Network Port object after activating the change, if required.

Configuration Requirements: Select a Network Port object, O1, which represents a configured network port, has a Protocol_Level of BACNET_APPLICATION, which references a Network Port object and for which there is a writable inherited property, P, within hierarchy. Let O2 be the Network Port object which contains P.

Test Steps:

1. V1 = READ O2, P
2. VERIFY O1, P = V1
3. WRITE O2, P = (V2: any valid value different that V1)
4. IF O2, Changes Pending THEN
 - TRANSMIT ReinitializeDevice
 - 'Reinitialized State of Device' = ACTIVATE_CHANGES
 - RECEIVE BACnet-SimpleACK-PDU
 - MAKE(reconfigure the TD and other devices on the network to the new network settings)
 - WAIT **Activate Changes Fail Time**
5. VERIFY O1, P = V2

7.3.2.46.4.4 Changes Reflected in Lower Network Port Objects

Purpose: To verify that changing properties in the top Network Port object results in the new property values reflected in the child Network Port objects.

Test Concept: Write a writable, inherited property within a top Network Port object and verify that the new value is reflected in the property from which value is inherited after activating the change, if required.

Configuration Requirements: Select a Network Port object, O1, which represents a configured network port, has a Protocol_Level of BACNET_APPLICATION, which references a Network Port object and for which there is a writable inherited property, P, in O1 which inherits its value from property P in O2.

Test Steps:

1. V1 = READ O2, P
2. VERIFY O1, P = V1
3. WRITE O1, P = (V2: any valid value different that V1)
4. IF O1, Changes Pending THEN
 - TRANSMIT ReinitializeDevice
 - 'Reinitialized State of Device' = ACTIVATE_CHANGES
 - RECEIVE BACnet-SimpleACK-PDU
 - MAKE(reconfigure the TD and other devices on the network to the new network settings)
 - WAIT **Activate Changes Fail Time**
5. VERIFY O2, P = V2

7.3.2.46.5 APDU_Length Test

Purpose: To verify that the Device object does not report a Max_APDU_Length_Accepted that is larger than the largest value reported by the configured and enabled Network Port objects.

Test Concept: Determine the largest APDU_Length property for all configured and enabled Network Port objects with a Protocol_Level of BACNET_APPLICATION. Verify that each is larger than 50 and less than or equal the maximum allowed for the attached datalink. Verify that the Max_APDU_Length_Supported property of the Device object is not larger than that maximum.

Notes to Tester: the maximum allowable APDU_Length for a network type should be calculated from the maximum NPDU size minus 21 according to SSPC interpretation IC135-2020-2.

Test Steps:

1. MAX_APDU = 0

2. REPEAT NP = (all configured and enabled Network Port objects with a Protocol_Level of BACNET_APPLICATION) DO {
 - IF NP.APDU_Length < 50 THEN
 - ERROR "APDU_Length must not be less than 50."
 - IF NP.APDU_Length > (the maximum allowable for the Network_Type) THEN
 - ERROR "APDU_Length is too large for the connected Network_Type"
 - IF MAX_APDU <> NP.APDU_Length THEN
 - MAX_APDU = NP.APDU_Length
3. VERIFY (Device, 4194303), Max_APDU_Length_Supported <= MAX_APDU

7.3.2.46.6 Routing_Table Test

Purpose: To verify that routes are added to the Routing_Table property of the Network Port object when they are found.

Test Concept: Starting with a clear routing table, send an I-Am-Router-To-Network message with multiple networks listed and verify that all are added to the Routing_Table. Send the remaining Nmax-2 I-Am-Router-To-Network messages to the IUT and verify that the entries are placed into the Routing_Table. Verify that no other Network Port objects are affected by the messages.

Configuration: All enabled Network Port objects, NP1 .. NPx, have empty Routing_Table properties. NP1 is the Network Port for port A. Nmax is the smaller of the maximum number of entries the IUT can hold in its routing table, and the maximum that can be encoded in a single segment ReadProperty response. N1 .. Nmax is a set of random network numbers, none of which are in use by the IUT. R1 .. Rmax are the router MAC addresses for each of the network numbers in N1 .. Nmax. R1 and R2 shall be the same. The TD and IUT shall be on the same BACnet network and there shall be no other routers connected.

Notes to Tester: If the network cannot be configured with the TD and the IUT on the same network, the test shall be adjusted to include the router to the TDs network instead of having a cleared routing table from the start of the test.

Test Steps:

-- verify that no routes are known

1. REPEAT NP = (all enabled Network Port objects with a Protocol_Level of BACNET_APPLICATION) DO {
 - VERIFY NP, Routing_Table = ()

-- verify that the IUT notices all routes in the I-Am-Router-To-Network

2. IF the IUT supports storing the address of more than 1 router THEN
 - TRANSMIT PORT A
 - DESTINATION = LOCAL BROADCAST,
 - SOURCE R1,
 - I-Am-Router-To-Network,
 - Network Numbers = N1, N2
 - VERIFY NP1, Routing_Table = (
 - (N1, R1, AVAILABLE, (optionally, any valid index)),
 - (N2, R2, AVAILABLE, (optionally, any valid index))
 -) -- the order of the entries does not matter

ELSE

- TRANSMIT PORT A
 - DESTINATION = LOCAL BROADCAST,
 - SOURCE R1,
 - I-Am-Router-To-Network,
 - Network Numbers = N1
 - VERIFY NP1, Routing_Table = (
 - (N1, R1, AVAILABLE, (optionally, any valid index)),
 -)

-- verify that the IUT supports up to Nmax entries

4. REPEAT NP = (all enabled Network Port objects, except NP1, with a Protocol_Level of BACNET_APPLICATION) DO {


```

    VERIFY NP, Routing_Table = ()
  }
5. REPEAT N,R = (N2 up to Nmax, R2 up to Rmax) DO {
    TRANSMIT PORT A
    DESTINATION = LOCAL BROADCAST,
    SOURCE = R
    I-Am-Router-To-Network,
    Network Numbers = N
  }
6. VERIFY NP1, Routing_Table = (
    N1, R1, AVAILABLE, (optionally, any valid index),
    N2, R2, AVAILABLE, (optionally, any valid index),
    ...
    Nmax, Rmax, AVAILABLE, (optionally, any valid index)
  ) -- the order of the entries does not matter

-- verify that the other Network Port objects are unaffected
7. REPEAT NP = (all enabled Network Port objects, except NP1, with a Protocol_Level of
    BACNET_APPLICATION) DO {
    VERIFY NP, Routing_Table = ()
  }

```

7.3.2.46.7 DHCP Tests

7.3.2.46.7.1 Basic IPv4 DHCP Test

Purpose: Verify that the IUT is able to participate in IPv4 DHCP and correctly report its DHCP status.

Test Concept: The DHCP server is removed from network. The IUT is then configured with an IPv4 network requiring DHCP, and if required, its DHCP settings are cleared. The related Network Port object is queried to verify that the DHCP related properties have the appropriate values indicating DHCP has not completed. The DHCP is connected to the network. It is verified that the IUT obtains network settings from the DHCP server, and that the DHCP properties reflect the current status.

Configuration Requirements: The DHCP is disconnected from the network or turned off. The IUT is configured for DHCP and any settings it previously received via DHCP are cleared.

Test Steps:

1. IF the IUT has a second enabled network port THEN
 VERIFY IP_DHCP_Enable = True
 IF IP_DHCP_Lease_Time property is present THEN
 VERIFY IP_DHCP_Lease_Time = 0
 IF IP_DHCP_Lease_Time_Remaining property is present THEN
 VERIFY IP_DHCP_Lease_Time_Remaining = 0
 IF IP_DHCP_Server property is present THEN
 VERIFY IP_DHCP_Server = X'00000000'
2. MAKE(connect the DHCP server to the network)
3. WAIT until the IUT obtains DHCP information
4. IF IP_DHCP_Lease_Time property is present THEN
 VERIFY IP_DHCP_Lease_Time = (0 or the value provided by the DHCP server)
5. IF IP_DHCP_Lease_Time_Remaining property is present THEN
 VERIFY IP_DHCP_Lease_Time_Remaining = (0 or a value less than that provided by the DHCP server)
6. IF IP_DHCP_Server property is present THEN
 VERIFY IP_DHCP_Server = (the DHCP server's address or X'00000000')
7. VERIFY IP_Address = (the value served by the DHCP server)
8. VERIFY IP_Default_Gateway = (the value served by the DHCP server)

7.3.2.46.7.2 Basic IPv6 DHCP Test

Purpose: Verify that the IUT is able to participate in IPv6 DHCP and correctly report its DHCP status

Test Concept: The DHCP server is removed from network. The IUT is then configured with an IPv6 network requiring DHCP, and if required, its DHCP settings are cleared. The related Network Port object is queried to verify that the DHCP related properties have the appropriate values indicating DHCP has not completed. The DHCP is connected to the network. It is verified that the IUT obtains network settings from the DHCP server, and that the DHCP properties reflect the current DHCP status.

Configuration Requirements: The DHCP is disconnected from the network or turned off. The IUT is configured for DHCP and any settings it previously received via DHCP are cleared.

Test Steps:

1. IF the IUT has a second enabled network port THEN
 - VERIFY IPv6_DHCP_Enable = True
 - IF IPv6_DHCP_Lease_Time property is present THEN
 - VERIFY IPv6_DHCP_Lease_Time = 0
 - IF IPv6_DHCP_Lease_Time_Remaining property is present THEN
 - VERIFY IPv6_DHCP_Lease_Time_Remaining = 0
 - IF IPv6_DHCP_Server property is present THEN
 - VERIFY IPv6_DHCP_Server = X'00000000'
2. MAKE (connect the DHCP server to the network)
3. WAIT until the IUT obtains DHCP information
4. IF IPv6_DHCP_Lease_Time property is present THEN
 - VERIFY IPv6_DHCP_Lease_Time = (0 or the value provided by the DHCP server)
5. IF IPv6_DHCP_Lease_Time_Remaining property is present THEN
 - VERIFY IPv6_DHCP_Lease_Time_Remaining = (0 or a value less than that provided by the DHCP server)
6. IF IPv6_DHCP_Server property is present THEN
 - VERIFY IPv6_DHCP_Server = (the DHCP server's address or X'00000000')
7. VERIFY IPv6_Address = (the value served by the DHCP server)
8. VERIFY IPv6_Default_Gateway = (the value served by the DHCP server)

7.3.2.47 Timer Object Tests

7.3.2.47.1 Positive Tests

7.3.2.47.1.1 Timer State_Change_Values

Purpose: Verify all State_Change_Values property transitions.

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

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

Test Steps:

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

14. VERIFY Present_Value = 0

7.3.2.47.1.2 Timer Running then Expired Test

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

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

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

Test Steps:

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

7.3.2.47.1.3 Default_Timeout Test

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

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

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

Test Steps:

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

7.3.2.47.1.4 Running Timer by Writing the Present_Value

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

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

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

Test Steps:

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

7.3.2.47.1.5 Restarting An Expired Timer

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

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

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

Test Steps:

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

7.3.2.47.1.6 Already Running Timer Restarted by Writing the Present_Value

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

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

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

Test Steps:

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

7.3.2.47.1.7 Already Running Timer Restarted with Default_Timeout

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

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

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

Test Steps:

1. VERIFY Timer_State = RUNNING
2. READ DV = Default_Timeout
3. VERIFY Initial_Timeout <> DV
4. WRITE Timer_Running = TRUE
5. CHECK (IUT exhibits any changes configured in RUNNING_TO_RUNNING transition)

6. VERIFY Initial_Timeout = DV
7. VERIFY Present_Value $\approx$ DV
8. VERIFY Timer_Running = TRUE
9. VERIFY Last_State_Change = RUNNING_TO_RUNNING

7.3.2.47.1.8 Timer Accepts all the Required Datatypes in an Internal Reference

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

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

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

Test Steps:

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

7.3.2.47.1.9 Timer Supports Writing an External Device

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

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

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

Test Steps:

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

7.3.2.47.1.10 Forcing Timer Expiration by Writing Zero

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

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

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

Test Steps:

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

```

    IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time = UT
    }
    ELSE IF (Expiration_Time property is present in T1) THEN
        VERIFY Expiration_Time ~= (the current date and time)

```

7.3.2.47.1.11 Forcing Timer Expiration by Writing FALSE

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

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

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

Test Steps:

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

7.3.2.47.1.12 Forcing Timer Expiration by Writing IDLE

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

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

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

Test Steps:

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

7.3.2.47.1.13 Resetting Timer by Writing IDLE

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

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

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

Test Steps:

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

7.3.2.47.1.14 Timer Object Operation Unaffected by Changes to Local_Time and Local_Date

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

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

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

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

Test Steps:

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

7.3.2.47.1.15 Changes made by State_Change_Values are at Correct Priority

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

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

Configuration Requirements: Start this test in the IDLE state. O1 P1 is any commandable property amongst the elements of List_Of_Object_Property_References. O1 should contain in its Priority_Array at the index which will change, a value

which is different from the values in State_Change_Values, for the operations which will take place, for ease of ensuring that the Timer commanded the change.

Test Steps:

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

7.3.2.47.1.16 Changing Default_Timeout Test

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

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

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

Test Steps:

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

7.3.2.47.2 Negative Tests

7.3.2.47.2.1 Writing Timer with an Unsupported External Reference

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

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

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

Test Steps:

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

7.3.2.47.2.2 Writing an Unsupported Datatype to State_Change_Values

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

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

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

Test Steps:

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

7.3.2.47.2.3 Invalid Property Writing in a Timer

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

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

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

Test Steps:

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

7.3.2.47.2.4 Expired Timer Ignores Writing Zero

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

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

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

Test Steps:

1. VERIFY Timer_State = EXPIRED
2. WRITE Present_Value = 0
3. VERIFY Timer_State = EXPIRED

7.3.2.47.2.5 Expired Timer Ignores Writing FALSE

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

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

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

Test Steps:

1. VERIFY Timer_State = EXPIRED
2. WRITE Timer_Running = FALSE
3. VERIFY Timer_State = EXPIRED

7.3.2.47.2.6 Idle Timer Ignores Writing Zero

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

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

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

Test Steps:

1. VERIFY Timer_State = IDLE
2. WRITE Present_Value = 0
3. VERIFY Timer_State = IDLE

7.3.2.47.2.7 Idle Timer Ignores Writing FALSE

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

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

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

Test Steps:

1. VERIFY Timer_State = IDLE
2. WRITE Timer_Running = FALSE
3. VERIFY Timer_State = IDLE

7.3.2.47.2.8 Idle Timer Ignores Writing IDLE

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

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

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

Test Steps:

1. VERIFY Timer_State = IDLE
2. WRITE Timer_State = IDLE
3. VERIFY Timer_State = IDLE

7.3.2.47.2.9 Default_Timeout Written Outside Supported Range

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

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

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

Test Steps:

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

7.3.2.48 Audit Log Object Tests

7.3.2.48.1 One Audit Log Holds all of an Objects History Test

Purpose: Ensure that, for any arbitrary object, there is at least one Audit Log into which all of the object's audit notifications are placed.

Test Concept: Send a sequence of audit notifications which contain entries for multiple objects to the IUT. At least some of the objects shall have multiple audit records in the sequence. For each object instance represented in the audit notifications sent to the IUT, verify that there is at least one Audit Log which contains all of the audit notifications for the object.

Configuration Requirements: S is a sequence of audit notifications which contain entries for multiple objects to the IUT where at least some of the objects shall have multiple audit records in the sequence.

Test Steps:

1. REPEAT AN = (each notification in S) DO {
 TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = AN
}
2. REPEAT O = (each object represented in S) DO {
 SO = (the sequence of notifications in S for object O)
 FOUND = (false)
 REPEAT AL = (each Audit Log object) DO {
 IF (AL contains all notifications in SO) THEN
 FOUND = (true)
 }
 IF (FOUND is false) THEN {
 ERROR "no audit log was found which contains all notifications for object"
 }
}

7.3.2.48.2 Audit Notification Basic Combining Test

Purpose: Ensure that Audit Log objects correctly combine related audit notification records.

Test Concept: Send a sequence, SEQ1, of unrelated audit notifications to the IUT and verify that the notifications are not combined. Send a source audit notification, SN1, followed by a sequence, SEQ2, of unrelated audit notifications and verify that the notifications are not combined. Send a target audit notification, TN1, which should be combined with SN1. Verify that SN1 and TN1 are combined in the Audit Log. Repeat the process with new notifications but send the target notification before the source notification.'

Configuration Requirements: An audit log that should receive the combined SN/TN notification is AL. The Target Value and Current Value fields in SN and TN shall not be greater than 500 octets. D1 shall be the device sending the source notification and D2 shall be the device sending the target notification. It is acceptable if D1 is the same as D2.

Test Steps:

-- the source notification is sent before the target notification

1. REPEAT AN = (each notification in SEQ1) DO {
 TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = (D1 or D2),
 'Notifications' = AN
 }
2. TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = D1,
 'Notifications' = SN1
3. REPEAT AN = (each notification in SEQ2) DO {
 SOURCE = (D1 or D2),
 TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = AN
 }
4. TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = D2,
 'Notifications' = TN1
5. CHECK(that no record exists in AL which is just SN1)
6. CHECK(that no record exists in AL which is just TN1)
7. CHECK(that a record exists in AL which is the combination of SN1 and TN1)
8. CHECK(that the combined record has all of the source and target fields provided in SN1 and TN1, and no more.)

-- the target notification is sent before the source notification

9. REPEAT AN = (each notification in SEQ3) DO {
 TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = (D1 or D2),
 'Notifications' = AN
 }
10. TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = D1,
 'Notifications' = TN2
11. REPEAT AN = (each notification in SEQ4) DO {
 SOURCE = (D1 or D2),
 TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = AN
 }
12. TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = D2,
 'Notifications' = SN2
13. CHECK(that no record exists in AL which is just SN2)
14. CHECK(that no record exists in AL which is just TN2)
15. CHECK(that a record exists in AL which is the combination of SN2 and TN2)
16. CHECK(that the combined record has all of the source and target fields provided in SN2 and TN2, and no more.)

7.3.2.48.3 Audit Notification Combining Failure Test

Purpose: Ensure that Audit Log objects correctly combine related audit notification records which indicate failed actions.

Test Concept: Send a source audit notification, SN. Send a target audit notification, TN, which should be combined with SN, and which indicates that the action failed. Verify that SN and TN are combined in the Audit Log.

Configuration Requirements: An audit log that should receive the combined SN/TN notification is AL. The Target Value and Current Value fields in SN and TN shall not be greater than 500 octets.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,
'Notifications' = SN
2. TRANSMIT UnconfirmedAuditNotification-Request,
'Notifications' = TN
3. CHECK(that no record exists in AL which is just SN)
4. CHECK(that no record exists in AL which is just TN)
5. CHECK(that a record exists in AL which is the combination of SN and TN)
6. CHECK(that the combined record has all of the source and target fields provided in SN and TN, and no more.)

7.3.2.48.4 Audit Notification Non-combining Test

Purpose: Ensure that Audit Log objects correctly do not combine unrelated audit notification records.

Test Concept: Send a sequence of unrelated audit notifications to the IUT each differing by 1 field in the matching criteria. Verify that the notifications are not combined.

Configuration Requirements: SEQ is a sequence of audit notifications where each record differs from the previous record by 1 field. For the sequence, SN is a source notification with user-id, user-role, target-value fields, and without source-comment field, and TN is the matching target notification with user-id, user-role, target-value fields. The sequence is:

```
{
    (SN),
    (SN but with source-comment field),
    (TN with differing operation-source),
    (TN with differing operation),
    (TN with differing invoke-id),
    (TN with differing target-device),
    (TN with differing target-property),
    (TN with differing user-id),
    (TN with differing user-role),
    (TN with differing target-value),
    (TN with target-timestamp equal to source-timestamp + APDU_Timeout * 3)
}
```

Test Steps:

1. REPEAT AN = (each notification in SEQ) DO {
TRANSMIT UnconfirmedAuditNotification-Request,
'Notifications' = AN
}
2. REPEAT AN = (each notification in SEQ) DO {
CHECK(that AN is in an Audit Log and is not combined)
}

7.3.2.48.5 Audit Notification Combining Duplicate Test

Test Concept: Send a source audit notification SN. Verify it is placed in the log. Send a sequence, SEQ1, of unrelated audit notifications and verify SN is not combined with any. Resend SN and verify that SN was not re-added to the log.

Send a target audit notification, TN, which should be combined with SN. Verify that SN and TN are combined in the Audit Log. Send a sequence, SEQ2, of unrelated audit notifications. Resend TN and verify that TN was not re-added to the log.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,
'Notifications' = SN
2. CHECK(that SN is in the Audit Log)
3. REPEAT AN = (each notification in SEQ1) DO {
TRANSMIT UnconfirmedAuditNotification-Request,

- 'Notifications' = AN
 }
4. CHECK(that SN is in the Audit Log and is not combined)
 5. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = SN
 6. CHECK(that SN is in the Audit Log only once and is at its original position)
 7. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = TN
 8. CHECK(that SN is in the Audit Log only once, is combined with TN, and is at its original position)
 9. CHECK(that the combined record has all of the source and target fields provided in SN and TN, and no more.)
 10. REPEAT AN = (each notification in SEQ2) DO {
 TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = AN
 }
 11. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = TN
 12. CHECK(that TN is in the Audit Log only once, is combined with SN and is at SN's original position)

7.3.2.48.6 Audit Notification Combining Target Value Preference Test

Purpose: Ensure that Audit Log objects use the Current Value from a target notification when it is provided in both the source and target notifications.

Test Concept: Send a target audit notification TN1 which includes the Current Value field with a value CV1-T. Send a source audit notification, SN1, which should be combined with TN1, and which contains a Current Value field with a value CV1-S (CV1-S is different than CV1-T). Verify that SN1 and TN1 are combined in the Audit Log and that the Current Value in the log uses CV1-T. Repeat the steps sending a target notification TN2 before the source notification SN2 where CV2-S is different than CV2-T.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = TN1
2. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = SN1
3. CHECK(that TN1 is in the Audit Log only once, is combined with SN1, and that target-value is CV1-T)
4. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = SN2
5. TRANSMIT UnconfirmedAuditNotification-Request,
 'Notifications' = TN2
6. CHECK(that SN2 is in the Audit Log only once, is combined with TN2, and that target-value is CV2-T)

7.3.2.48.7 Accepts Audit Notifications from an Audit Forwarder Test

Purpose: Ensure that Audit Log accepts forwarded audit notifications.

Test Concept: The notification forwarder, AF1, sends a forwarded source notification, SN1, from the original sending device D1, to the IUT. Verify that the IUT places the notification in the Audit Log. AF1 then sends a forwarded target notification, TN1, from the original target device D2, to the IUT. Verify that the IUT combines the target with the source notification.

Configuration Requirements: The test network consists of a source device D1, a target device D2, and a notification forwarder, AF1.

Test Steps:

1. TRANSMIT UnconfirmedAuditNotification-Request,
 SOURCE = AF1,
 'Notifications' = SN1

2. TRANSMIT UnconfirmedAuditNotification-Request,
SOURCE = AF1,
'Notifications' = TN1
3. CHECK(that SN1 and TN1 are combined in the Audit Log)

7.3.2.48.8 Hierarchical Logging Test

Purpose: Ensure that an Audit Log configured with a parent correctly forwards notifications to the parent log.

Test Concept: An Audit Log, AL1, configured to reference a parent log located in another device, is sent a sequence of audit notifications. Within the sequence will some notifications which should be combined and some which should not be combined. Verify that the IUT forwards the notifications to the parent before the vendor specified maximum forwarding delay.

Configuration Requirements: The Audit Log, AL1, is configured with a Member_Of set to AL2, where AL2 is in the TD. AL1's Delete_On_Forward shall be set to FALSE. SEQ is a sequence of audit notifications where at least 2 are related and should be combined.

Notes to Tester: The standard does not provide guidance on how long an Audit Log object has before it must forward audit notifications to its parent. As such, the vendor is allowed to specify the maximum time as long as it is not unreasonable (delays on the order of days are clearly unreasonable; delays on the order of minutes are clearly acceptable).

Notes to Tester: When receiving notifications from the IUT, those notifications which should be combined, may be sent combined or not at the IUT's discretion.

Test Steps:

1. REPEAT AN = (each notification in SEQ) DO {
TRANSMIT UnconfirmedAuditNotification-Request,
SOURCE = (a value appropriate to the notification),
'Notifications' = AN
}
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
BEFORE **Audit Notification Forwarder Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = (one or more of the notifications from SEQ)
} ELSE {
BEFORE **Audit Notification Forwarder Fail Time**
RECEIVE ConfirmedAuditNotification-Request,
'Notifications' = (one or more of the notifications from SEQ)
TRANSMIT BACnet-SimpleACK-PDU
}
3. WHILE (not all notifications in SEQ have been sent by the IUT) {
IF the IUT is configured to send unconfirmed audit notifications THEN {
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = (one or more of the as yet unreceived notifications from SEQ)
} ELSE {
RECEIVE ConfirmedAuditNotification-Request,
'Notifications' = (one or more of the as yet unreceived notifications from SEQ)
TRANSMIT BACnet-SimpleACK-PDU
}
}
4. CHECK(that the notifications in SEQ are still in AL1)
5. CHECK(that the notifications in SEQ which are to be combined are combined in AL1)

7.3.2.49 Audit Reporter Object Tests

7.3.2.49.1 Target Audit Reporting - Basic Notification Test

Purpose: Verify that target audit notifications are properly formed.

Test Concept: The IUT is made to send a target audit notification. It is verified that the target fields are present, no source fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable operations. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, on IUT)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({

-- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = O,
 -- source-comment absent
 target-comment = (any valid value, or absent unless O is GENERAL),
 invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property.
 shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 May be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op or absent if no target value.
 may be absent if the value size is larger than 32 encoded octets),
 result = (the reason for failure if the op failed, otherwise absent)

 })
- } ELSE {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = ({

-- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = O,
 -- source-comment absent
 target-comment = (any valid value, or absent unless O is GENERAL),
 invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property.
 Shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op or absent if no target value.

 })

May be absent if the value size is larger than 32 encoded octets),
result = (the reason for failure if the op failed, otherwise absent)

})

TRANSMIT BACnet-SimpleACK-PDU

}

7.3.2.49.2 Target Audit Reporting - Unconfirmed Service Operation Test

Purpose: Verify that target audit notifications for unconfirmed services do not contain InvokeId information.

Test Concept: An auditable unconfirmed service is performed on the IUT, and it is verified that the resulting target audit notification does not contain an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for an unconfirmed service. If the IUT does not support audit reporting for any unconfirmed services, this test shall be skipped. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, which uses an unconfirmed service, on IUT)
2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {
BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = O,
-- source-comment absent
target-comment = (any valid value, or absent unless O is GENERAL),
-- invoke-id absent,
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise
absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property,
shall be 16 or absent if no priority supplied and the target is a property),
target-value = (the target value or absent if no target value for the operation.
May be absent if the value size is larger than 32 encoded octets),
current-value = (the value before the op or absent if no target value.
may be absent if the value size is larger than 32 encoded octets),
result = (the reason for failure if the op failed, otherwise absent)
)
} ELSE {
BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
RECEIVE ConfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = O,
-- source-comment absent
target-comment = (any valid value, or absent unless O is GENERAL),
-- invoke-id absent,
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise
absent),
}

```

        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
        Shall be 16 or absent if no priority supplied and the target is a property),
        target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op or absent if no target value.
        May be absent if the value size is larger than 32 encoded octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.3 Target Audit Reporting - Confirmed Service Operation Audit Notification

Purpose: Verify that target audit notifications for confirmed services contain InvokeId information.

Test Concept: An auditable confirmed service is performed on the IUT, and it is verified that the resulting target audit notification contains an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for an unconfirmed service. The IUT is configured so that the notification will be reported through Audit Reporter AR.

Test Steps:

1. MAKE(perform an auditable operation, O, which uses an unconfirmed service, on IUT)
 2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({ -- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = O,
 -- source-comment absent
 target-comment = (any, valid value, or absent unless O is GENERAL),
 -- invoke-id absent,
 source-user-id = (the value from the operation if provided, otherwise
 absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a
 property),
 target-priority = (the priority supplied, or absent if the target is not a
 property. shall be 16 or absent if no priority supplied and the
 target is a property),
 target-value = (the target value or absent if no target value for the
 operation. may be absent if the value size is larger than 32
 encoded octets),
 current-value = (the value before the op or absent if no target value.
 may be absent if the value size is larger than 32 encoded
 octets),
 result = (the reason for failure if the op failed, otherwise absent)
 })
- } ELSE {
- BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = ({ -- source-timestamp absent

```

target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = O,
-- source-comment absent
target-comment = (any valid value, or absent unless O is GENERAL),
-- invoke-id absent,
source-user-id = (the value from the operation if provided, otherwise
absent),
source-user-role = (the value from the operation if provided, otherwise
absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a
property),
target-priority = (the priority supplied, or absent if the target is not a
property. shall be 16 or absent if no priority supplied and the
target is a property),
target-value = (the target value or absent if no target value for the
operation. may be absent if the value size is larger than 32
encoded octets),
current-value = (the value before the op or absent if no target value.
May be absent if the value size is larger than 32 encoded octets),
result = (the reason for failure if the op failed, otherwise absent)
} )
TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.4 Target Audit Reporting - Operations with Priority Test

Purpose: Verify that target audit notifications which for writes which convey a priority include the priority in the notification.

Test Concept: An auditable write, which includes a priority, is performed on a commandable object in the IUT and it is verified that the resulting target audit notification contains a priority, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. The IUT is configured so that the notification will be reported through Audit Reporter AR using unconfirmed notifications. If the IUT does not support the Priority_Array property in any object for which audit reporting can be configured, this test shall be skipped.

Test Steps:

1. TRANSMIT WriteProperty-Request,
 - 'Invoke Id' = I,
 - 'Object Identifier' = O1,
 - 'Property Identifier' = Present_Value,
 - 'Property Value' = (V: any valid value),
 - 'Priority' = (PRIO: a priority in the range 1 - 15)
2. BEFORE **Internal Processing Fail Time**
 - RECEIVE BACnet-SimpleACK-PDU
 - (BACnet-Error-PDU,
 - 'Error Type' = (E : any error)
 -)
3. IF the IUT is configured to send unconfirmed audit notifications THEN {
 - BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 - RECEIVE UnconfirmedAuditNotification-Request,
 - 'Notifications' = ({
 - source-timestamp absent

```

        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = WRITE,
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = I,
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = O1,
        target-property = Present_Value,
        target-priority = PRIO,
        target-value = V,
        current-value = (the value before the write. may be absent if the value size is
            larger than 32 encoded octets),
        result = (E, if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = WRITE,
        -- source-comment absent
        target-comment = (any valid value, or absent),
        invoke-id = I,
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = O1,
        target-property = Present_Value,
        target-priority = PRIO,
        target-value = V,
        current-value = (the value before the write. may be absent if the value size is
            larger than 32 encoded octets),
        result = (E, if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.5 Target Audit Reporting - Target_Value and Current_Value Test

Purpose: Verify that the IUT reports target values and current values, when the audited operation contains a value (such as for writes).

Test Concept: An auditable operation, which contains a value, is performed on object O1 and property P1. The resulting audit notification is verified to contain the provided value and the value before the operation.

Configuration Requirements: The IUT is configured to report all audit notifications. If possible, a property which is not changing shall be the target of the operation so that the current value field can be validated. If the IUT does not have any objects which support reporting of operation which contain target value, this test shall be skipped. AR is the Audit Report through which O1 reports audit notifications.

Test Steps:

1. IF P1 is not changing outside of the operation THEN {
 READ IV = O1, P1
2. MAKE(perform an auditable operation, on O1, P1, which provides a target value, V,
 which is less than 32 octets in size)
3. IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({
 -- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = (the operation performed),
 -- source-comment absent
 target-comment = (any valid value, or absent),
 invoke-id = (the invoke id from the operation, or absent if it was
 unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise
 absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = IUT,
 target-object = O1,
 target-property = P1,
 target-priority = (the priority from the operation, or absent if 16 or not
 provided in the operation),
 target-value = V,
 current-value = (CV: any valid value),
 result = (E, if the op failed, otherwise absent)
 })
} ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = ({
 -- source-timestamp absent
 target-timestamp = (IUT's local time),
 source-device = TD,
 -- source-object absent
 operation = (the operation performed),
 -- source-comment absent
 target-comment = (any valid value, or absent),
 invoke-id = (the invoke id from the operation, or absent if it was
 unconfirmed),
 source-user-id = (the value from the operation if provided, otherwise
 absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = IUT,
 target-object = O1,
 target-property = P1,
 target-priority = (the priority from the operation, or absent if 16 or not
 provided in the operation),
 target-value = V,
 current-value = (CV: any valid value),
 result = (E, if the op failed, otherwise absent)
 })
 TRANSMIT BACnet-SimpleACK-PDU
}

4. IF the P1 is not changing outside of the operation THEN
CHECK(CV equals IV)

7.3.2.49.6 Target Audit Reporting - Error Audit Notification Test

Purpose: Verify that operations that fail are properly reported in audit notifications.

Test Concept: An auditable operation, which will fail with a Result(-) or Result(+) with error information is performed on the IUT. It is verified that an audit notification is sent which contains the error that occurred. The auditable operation performed shall be one for which the IUT will report failures via audit notifications.

Configuration Requirements: The IUT is configured to report all audit notifications.

Test Steps:

1. MAKE(perform an auditable operation, O, on IUT which will fail (via return of a BACnetErrorPDU or a Result(+) with error information)
2. IF the IUT is configured to send unconfirmed audit notifications for operation O THEN {
BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = O,
-- source-comment absent
target-comment = (any valid value, or absent),
invoke-id = (the invoke id from the operation, or absent if it was unconfirmed),
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise absent),
target-device = IUT,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property, shall be 16 or absent if no priority supplied and the target is a property),
target-value = (the target value or absent if no target value for the operation, may be absent if the value size is larger than 32 encoded octets),
current-value = (the value before the op or absent if no target value, may be absent if the value size is larger than 32 encoded octets),
result = (the error reported for the operation)
})
} ELSE {
BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
RECEIVE ConfirmedAuditNotification-Request,
'Notifications' = ({
-- source-timestamp absent
target-timestamp = (IUT's local time),
source-device = TD,
-- source-object absent
operation = (the operation performed),
-- source-comment absent
target-comment = (any valid value, or absent),
invoke-id = (the invoke id from the operation, or absent if it was

```

        unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a
            property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op or absent if no target value.
            may be absent if the value size is larger than 32 encoded
            octets),
        result = (the error reported for the operation)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.7 Target Audit Reporting - GENERAL Operation Test

Purpose: Verify that GENERAL operation audit notifications contain a Target Comment.

Test Concept: An auditable GENERAL operation is performed on the IUT. It is verified that an audit notification is sent which contains the error that occurred.

Configuration Requirements: The IUT is configured to report all audit notifications. If the IUT does not generate GENERAL audit notifications, this test shall be skipped.

Test Steps:

1. MAKE(make the IUT generate a GENERAL audit notification)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = {

 - source-timestamp absent
 - target-timestamp = (IUT's local time),
 - source-device = TD,
 - source-object absent
 - operation = GENERAL,
 - source-comment absent
 - target-comment = (any valid value),
 - invoke-id = (the invoke id from the operation, or
 - absent if it was unconfirmed),
 - source-user-id = (the value from the operation if provided, otherwise absent),
 - source-user-role = (the value from the operation if provided, otherwise
 - absent),
 - target-device = IUT,
 - target-object = (the target object or absent if the target is not an object),
 - target-property = (the target property or absent if the target is not a
 - property),
 - target-priority = (the priority supplied, or absent if the target is not a
 - property. shall be 16 or absent if no priority supplied and the
 - target is a property),
 - target-value = (the target value or absent if no target value for the
 - operation. may be absent if the value size is larger than 32

```

        encoded octets),
        current-value = (the value before the op or absent if no target value.
            may be absent if the value size is larger than 32 encoded octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        -- source-timestamp absent
        target-timestamp = (IUT's local time),
        source-device = TD,
        -- source-object absent
        operation = GENERAL,
        -- source-comment absent
        target-comment = (any valid value),
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise
            absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = IUT,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. Shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the operation.
            May be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op or absent if no target value.
            May be absent if the value size is larger than 32 encoded octets),
        result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.8 Source Audit Reporting - Basic Notification Test

Purpose: Verify that source audit notifications are properly formed.

Test Concept: The IUT is made to send a source audit notification. It is verified that the source fields are present, no target fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable source operations. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {


```

                BEFORE AR.Maximum_Send_Delay + Notification Fail Time
                RECEIVE UnconfirmedAuditNotification-Request,
                'Notifications' = ( {
                    source-timestamp = (IUT's local time),
                    -- target-timestamp absent
                    source-device = IUT,
                    source-object = (the object which initiated the op or absent if not initiated by
                        an object),
                    operation = O,
                    source-comment = (any valid value or absent),

```

```

-- target-comment absent
invoke-id = (the invoke id from the operation, or absent if it was
    unconfirmed),
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise
    absent),
target-device = TD,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property.
    Shall be 16 or absent if no priority supplied and the target is a property),
target-value = (the target value or absent if no target value for the operation.
    may be absent if the value size is larger than 32 encoded octets),
current-value = (the value before the op if the op targeted a property, or
    absent. May be absent even if targeting a property),
result = (the reason for failure if the op failed, otherwise absent)
    )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by
            an object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
            shall be 16 or absent if no priority supplied and the target is a property),
        target-value = (the target value or absent if no target value for the operation.
            may be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the reason for failure if the op failed, otherwise absent)
    )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.9 Source Audit Reporting - Same Device Notification Test

Purpose: Verify that source and target fields are in audit notifications are performed by the IUT on the IUT.

Test Concept: The IUT is made to perform an auditable operation on itself. It is verified that the source and target fields are present, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report all auditable operations. The IUT is configured with AR as the source Audit Reporter object. If the IUT is unable to perform an auditable operation on itself, this test shall be skipped.

Notes to Tester: The IUT is allowed to send the notifications as 2 separate notifications in the same audit notification message, or in separate messages. When sending separate notifications, one shall be a correctly formed target notification and the other a correctly formed source notification for the operation performed

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the itself)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({

source-timestamp = (T: IUT's local time),
 target-timestamp = T,
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not initiated by
 an object),
 operation = O,
 source-comment = (any valid value or absent),
 target-comment = (any valid value or absent),
 -- invoke-id absent
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property.
 Shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op if the op targeted a property, or
 absent),
 result = (the reason for failure if the op failed, otherwise absent)

 })
- } ELSE {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = ({

source-timestamp = (T: IUT's local time),
 target-timestamp = T,
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not initiated by
 an object),
 operation = O,
 source-comment = (any valid value or absent),
 target-comment = (any valid value or absent),
 -- invoke-id absent
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = IUT,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property.
 shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op if the op targeted a property, or
 absent),
 result = (the reason for failure if the op failed, otherwise absent)

 })
- TRANSMIT BACnet-SimpleACK-PDU

7.3.2.49.10 Source Audit Reporting - Unconfirmed Service Operation Test

Purpose: Verify that source audit notifications for unconfirmed services do not contain InvokeId information.

Test Concept: An auditable unconfirmed service is performed by the IUT, and it is verified that the resulting source audit notification does not contain an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report source audit notifications for an unconfirmed service. If the IUT does not support source audit reporting for any unconfirmed services, this test shall be skipped. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD which uses an unconfirmed service)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = ({

source-timestamp = (IUT's local time),

-- target-timestamp absent

source-device = IUT,

source-object = (the object which initiated the op or absent if not initiated by an object),

operation = O,

source-comment = (any valid value or absent),

-- target-comment absent

-- invoke-id absent

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = TD,

target-object = (the target object or absent if the target is not an object),

target-property = (the target property or absent if the target is not a property),

target-priority = (the priority supplied, or absent if the target is not a property.

shall be 16 or absent if no priority supplied and the target is a property),

target-value = (the target value or absent if no target value for the operation.

may be absent if the value size is larger than 32 encoded octets),

current-value = (the value before the op if the op targeted a property, or

absent. May be absent even if targeting a property),

result = (the reason for failure if the op failed, otherwise absent)

})

} ELSE {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

RECEIVE ConfirmedAuditNotification-Request,

'Notifications' = ({

source-timestamp = (IUT's local time),

-- target-timestamp absent

source-device = IUT,

source-object = (the object which initiated the op or absent if not initiated by an object),

operation = O,

source-comment = (any valid value or absent),

-- target-comment absent

-- invoke-id absent

source-user-id = (the value from the operation if provided, otherwise absent),

source-user-role = (the value from the operation if provided, otherwise absent),

target-device = TD,

target-object = (the target object or absent if the target is not an object),

target-property = (the target property or absent if the target is not a property),

target-priority = (the priority supplied, or absent if the target is not a property.

shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op if the op targeted a property, or
 absent. May be absent even if targeting a property),
 result = (the reason for failure if the op failed, otherwise absent)

```
    })
    TRANSMIT BACnet-SimpleACK-PDU
}
```

7.3.2.49.11 Source Audit Reporting - Confirmed Service Operation Audit Notification

Purpose: Verify that source audit notifications for confirmed services contain InvokeId information.

Test Concept: An auditable confirmed service is performed by the IUT, and it is verified that the resulting source audit notification contains an InvokeId, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report source audit notifications for a confirmed service. If the IUT does not support source audit reporting for any confirmed services, this test shall be skipped. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

1. MAKE(the IUT perform an auditable operation, O, on the TD which uses an confirmed service)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = ({
 source-timestamp = (IUT's local time),
 -- target-timestamp absent
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not initiated by
 an object),
 operation = O,
 source-comment = (any valid value or absent),
 -- target-comment absent
 invoke-id = (the Invoke Id from the operation),
 source-user-id = (the value from the operation if provided, otherwise absent),
 source-user-role = (the value from the operation if provided, otherwise
 absent),
 target-device = TD,
 target-object = (the target object or absent if the target is not an object),
 target-property = (the target property or absent if the target is not a property),
 target-priority = (the priority supplied, or absent if the target is not a property.
 shall be 16 or absent if no priority supplied and the target is a property),
 target-value = (the target value or absent if no target value for the operation.
 may be absent if the value size is larger than 32 encoded octets),
 current-value = (the value before the op if the op targeted a property, or
 absent. May be absent even if targeting a property),
 result = (the reason for failure if the op failed, otherwise absent)
 })
 } ELSE {
 BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**
 RECEIVE ConfirmedAuditNotification-Request,
 'Notifications' = ({
 source-timestamp = (IUT's local time),
 -- target-timestamp absent
 source-device = IUT,
 source-object = (the object which initiated the op or absent if not initiated by
 an object),
 operation = O,


```

    source-comment = (any valid value or absent),
    -- target-comment absent
    invoke-id = (the Invoke Id from the operation),
    source-user-id = (the value from the operation if provided, otherwise absent),
    source-user-role = (the value from the operation if provided, otherwise
        absent),
    target-device = TD,
    target-object = (the target object or absent if the target is not an object),
    target-property = (the target property or absent if the target is not a property),
    target-priority = (the priority supplied, or absent if the target is not a property.
        shall be 16 or absent if no priority supplied and the target is a property),
    target-value = (the target value or absent if no target value for the operation.
        may be absent if the value size is larger than 32 encoded octets),
    current-value = (the value before the op if the op targeted a property, or
        absent. May be absent even if targeting a property),
    result = (the reason for failure if the op failed, otherwise absent)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.12 Source Audit Reporting - Operations with Priority Test

Purpose: Verify that source audit notifications which for writes which convey a priority include the priority in the notification.

Test Concept: An auditable write, which includes a priority, is performed on a commandable object by the IUT and it is verified that the resulting source audit notification contains a priority, and other notification fields represent the auditable operation performed.

Configuration Requirements: The IUT is configured to report audit notifications for writes on a commandable object, O1. The IUT is configured with AR as the source Audit Reporter object. If the IUT does not provide priorities in auditable operations it performed, this test shall be skipped.

Test Steps:

1. MAKE(the IUT perform an auditable operation in containing a priority, other than 16, on the TD)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

RECEIVE UnconfirmedAuditNotification-Request,

'Notifications' = {

 - source-timestamp = (IUT's local time),
 - target-timestamp absent
 - source-device = IUT,
 - source-object = (the object which initiated the op or absent if not initiated by an object),
 - operation = O,
 - source-comment = (any valid value or absent),
 - target-comment absent
 - invoke-id = (the Invoke Id from the operation),
 - source-user-id = (the value from the operation if provided, otherwise absent),
 - source-user-role = (the value from the operation if provided, otherwise absent),
 - target-device = TD,
 - target-object = (the target object),
 - target-property = (the target property),
 - target-priority = (the priority supplied),
 - target-value = (the target value.
 - may be absent if the value size is larger than 32 encoded octets),
 - current-value = (any valid value, or absent),
 - result = (the reason for failure if the op failed, otherwise absent)

})

```

} ELSE {
  BEFORE AR.Maximum_Send_Delay + Notification Fail Time
  RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
      source-timestamp = (IUT's local time),
      -- target-timestamp absent
      source-device = IUT,
      source-object = (the object which initiated the op or absent if not initiated by
        an object),
      operation = O,
      source-comment = (any valid value or absent),
      -- target-comment absent
      invoke-id = (the Invoke Id from the operation),
      source-user-id = (the value from the operation if provided, otherwise absent),
      source-user-role = (the value from the operation if provided, otherwise
        absent),
      target-device = TD,
      target-object = (the target object),
      target-property = (the target property),
      target-priority = (the priority supplied),
      target-value = (the target value.
        may be absent if the value size is larger than 32 encoded octets),
      current-value = (any valid value, or absent),
      result = (the reason for failure if the op failed, otherwise absent)
    } )
  TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.13 Source Audit Reporting - Error Audit Notification Test

Purpose: Verify that operations performed by the IUT which fail are properly reported in audit notifications.

Test Concept: The IUT is made to perform an auditable operation on the TD and the TD returns an Error-PDU. It is verified that a source audit notification is sent which contains the error that occurred. This is repeated twice more with the TD returning a Reject PDU, and then an Abort PDU.

Configuration Requirements: The IUT is configured to report all audit notifications. The IUT is configured with AR as the source Audit Reporter object.

Test Steps:

-- Error-PDU

1. MAKE(the IUT perform an auditable operation on TD which will fail (via return of a BACnetErrorPDU, or a Result(+) with error information)
2. IF the IUT is configured to send unconfirmed audit notifications THEN {

BEFORE AR.Maximum_Send_Delay + **Notification Fail Time**

RECEIVE UnconfirmedAuditNotification-Request,

```

'Notifications' = ( {
  source-timestamp = (IUT's local time),
  -- target-timestamp absent
  source-device = IUT,
  source-object = (the object which initiated the op or absent if not
    initiated by an object),
  operation = O,
  source-comment = (any valid value or absent),
  -- target-comment absent
  invoke-id = (the invoke id from the operation, or absent if it was
    unconfirmed),
  source-user-id = (the value from the operation if provided, otherwise absent),
  source-user-role = (the value from the operation if provided, otherwise
    absent),

```

```

target-device = TD,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a
    property. shall be 16 or absent if no priority supplied and the
    target is a property),
target-value = (the target value or absent if no target value for the
    operation. may be absent if the value size is larger than 32
    encoded octets),
current-value = (the value before the op if the op targeted a property, or
    absent. May be absent even if targeting a property),
result = (the error reported for the operation)
    } )
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not
            initiated by an object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a
            property. shall be 16 or absent if no priority supplied and the
            target is a property),
        target-value = (the target value or absent if no target value for the
            operation. may be absent if the value size is larger than 32
            encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the error reported for the operation)
    } )
    TRANSMIT BACnet-SimpleACK-PDU
}

-- Reject-PDU
3. MAKE (the IUT perform an auditable operation on TD which will fail (via return of a BACnetRejectPDU))
4. IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by
            an object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent

```

```

        invoke-id = (the invoke id from the operation, or absent if it was
                     unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
                             absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
                           shall be 16 or absent if no priority supplied and the target is a property),
        target-value = (the target value or absent if no target value for the operation.
                        may be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op if the op targeted a property, or
                         absent. May be absent even if targeting a property),
        result = (the error reported for the operation)
    })
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by
                        an object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
                     unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
                             absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
                           shall be 16 or absent if no priority supplied and the target is a property),
        target-value = (the target value or absent if no target value for the operation.
                        may be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op if the op targeted a property, or
                         absent. May be absent even if targeting a property),
        result = (the error reported for the operation)
    })
    TRANSMIT BACnet-SimpleACK-PDU
}

-- Abort-PDU
3. MAKE(the IUT perform an auditable operation on TD which will fail (via return of a BACnetAbortPDU))
4. IF the IUT is configured to send unconfirmed audit notifications THEN {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE UnconfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by
                        an object),
        operation = O,
        source-comment = (any valid value or absent),

```

```

-- target-comment absent
invoke-id = (the invoke id from the operation, or absent if it was
    unconfirmed),
source-user-id = (the value from the operation if provided, otherwise absent),
source-user-role = (the value from the operation if provided, otherwise
    absent),
target-device = TD,
target-object = (the target object or absent if the target is not an object),
target-property = (the target property or absent if the target is not a property),
target-priority = (the priority supplied, or absent if the target is not a property.
    shall be 16 or absent if no priority supplied and the target is a property),
target-value = (the target value or absent if no target value for the operation.
    may be absent if the value size is larger than 32 encoded octets),
current-value = (the value before the op if the op targeted a property, or
    absent. May be absent even if targeting a property),
result = (the error reported for the operation)
    })
} ELSE {
    BEFORE AR.Maximum_Send_Delay + Notification Fail Time
    RECEIVE ConfirmedAuditNotification-Request,
    'Notifications' = ( {
        source-timestamp = (IUT's local time),
        -- target-timestamp absent
        source-device = IUT,
        source-object = (the object which initiated the op or absent if not initiated by
            an object),
        operation = O,
        source-comment = (any valid value or absent),
        -- target-comment absent
        invoke-id = (the invoke id from the operation, or absent if it was
            unconfirmed),
        source-user-id = (the value from the operation if provided, otherwise absent),
        source-user-role = (the value from the operation if provided, otherwise
            absent),
        target-device = TD,
        target-object = (the target object or absent if the target is not an object),
        target-property = (the target property or absent if the target is not a property),
        target-priority = (the priority supplied, or absent if the target is not a property.
            Shall be 16 or absent if no priority supplied and the target is a property),
        target-value = (the target value or absent if no target value for the operation.
            may be absent if the value size is larger than 32 encoded octets),
        current-value = (the value before the op if the op targeted a property, or
            absent. May be absent even if targeting a property),
        result = (the error reported for the operation)
    })
    TRANSMIT BACnet-SimpleACK-PDU
}

```

7.3.2.49.14 Source Audit Reporting - Single Source Audit Reporter Object Test

Purpose: Verify that the IUT contains a single Audit Report for source audit reporting.

Test Concept: Check all Audit Reporter objects in the IUT and verify that only one is for source audit reporting.

Test Steps:

1. SourceAR = NONE
2. REPEAT AR = (each Audit Reporter object) DO {
 - IF AR.Audit_Source_Reporter is TRUE THEN
 - IF SourceAR is not NONE THEN
 - ERROR "Multiple Audit Reporter objects setup for source reporting."

- }
3. CHECK(SourceAR is not NONE)

7.3.2.49.15 Audit Forwarding Test

Purpose: Verify that the IUT forwards received audit notifications.

Test Concept: Send a sequence of confirmed and unconfirmed, unicast, and broadcast audit notifications to the IUT and verify they are forwarded.

Configuration Requirements: The IUT is configured with an Audit Log, AL, which is setup to forward and delete audit notifications with no delay. The IUT is configured to send notifications unconfirmed.

Test Steps:

-- Unicast confirmed

1. TRANSMIT ConfirmedAuditNotification-Request,
'Notifications' = (N1: a list of 1 or more notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = N1
4. TRANSMIT ReadRange-Request,
'Object Identifier' = AL,
'Property Identifier' = Log_Buffer,
'Reference Index' = 1,
'Count' = 10
5. RECEIVE ReadRange-Ack,
'Object Identifier' = AL,
'Property Identifier' = Log_Buffer,
'Result Flags' = (False, False, False),
'Count' = 0

-- Unicast unconfirmed

6. TRANSMIT UnconfirmedAuditNotification-Request,
'Notifications' = (N2: a list of 1 or more notifications)
7. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = N2
8. TRANSMIT ReadRange-Request,
'Object Identifier' = AL,
'Property Identifier' = Log_Buffer,
'Reference Index' = 1,
'Count' = 10
9. RECEIVE ReadRange-Ack,
'Object Identifier' = AL,
'Property Identifier' = Log_Buffer,
'Result Flags' = (False, False, False),
'Count' = 0

-- Local Broadcast

10. TRANSMIT UnconfirmedAuditNotification-Request,
DESTINATION = LOCAL BROADCAST,
'Notifications' = (N3: a list of 1 or more notifications)
11. BEFORE **Notification Fail Time**
RECEIVE UnconfirmedAuditNotification-Request,
'Notifications' = N3
12. TRANSMIT ReadRange-Request,
'Object Identifier' = AL,
'Property Identifier' = Log_Buffer,
'Reference Index' = 1,

- 'Count' = 10
13. RECEIVE ReadRange-Ack,
 'Object Identifier' = AL,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = (False, False, False),
 'Count' = 0
 - Global Broadcast
 10. TRANSMIT UnconfirmedAuditNotification-Request,
 DESTINATION = GLOBAL BROADCAST,
 'Notifications' = (N4: a list of 1 or more notifications)
 11. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedAuditNotification-Request,
 'Notifications' = N4
 12. TRANSMIT ReadRange-Request,
 'Object Identifier' = AL,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = 1,
 'Count' = 10
 13. RECEIVE ReadRange-Ack,
 'Object Identifier' = AL,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = (False, False, False),
 'Count' = 0

7.3.2.50 Staging Object Tests

7.3.2.50.1 Clamping Present_Value to Max_Pres_Value or Min_Pres_Value

Purpose: To verify that Present_Value will be modified internally to stay within the boundaries of Min_Pres_Value or Max_Pres_Value.

Test Concept: Present_Value is written with a value greater than Max_Pres_Value. If the value is accepted, Present_Value is read to verify that it clamped to Max_Pres_Value. If Stages is writable, an attempt is made to reduce the limit defined in the last stage. If successful, Present_Value is checked to verify it changed to match the new limit. Present_Value is then written with a value less than Min_Pres_Value. If the value is accepted, Present_Value is read to verify that it clamped to Min_Pres_Value. If Min_Pres_Value is writable, the value is increased and Present_Value is read to verify that it matches the new Min_Pres_Value.

Configuration Requirements: None

Test Steps:

1. READ MAXPV1 = Max_Pres_Value
2. READ PV1 = Present_Value
3. TRANSMIT WriteProperty-Request
 'Object-Identifier' = (the Staging object under test),
 'Property Identifier' = Present_Value,
 'Property Value' = (a value greater than MAXPV1)
4. RECEIVE
 BACnet-SimpleACK-PDU |
 (BACnet-Error-PDU,
 'Error Class' = Property,
 'Error Code' = VALUE_OUT_OF_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN
 VERIFY Present_Value = MAXPV1
 ELSE
 VERIFY Present_Value = PV1
 WRITE Present_Value = MAXPV1
6. IF (Stages is writable) THEN
 READ NS = Stages[0]
 READ STGN = Stages, ARRAY INDEX = NS

TRANSMIT WriteProperty-Request

'Object-Identifier' = (the Staging object under test),
 'Property Identifier' = Stages,
 'Property Array Index' = NS,
 'Property Value' = {
 Limit = (STAGEPV1: any value less than STGN.Limit)
 Values = STGN.Values,
 DeadBand = STGN.Deadband
 }

RECEIVE

BACnet-SimpleACK-PDU |
 (BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE)

IF (a BACnet-SimpleACK-PDU was received) THEN
 VERIFY Present_Value = STAGEPV1

7. READ MINPV1 = Min_Pres_Value

8. READ PV2 = Present_Value

9. TRANSMIT WriteProperty-Request

'Object-Identifier' = (the Staging object under test),
 'Property Identifier' = Present_Value,
 'Property Value' = (a value less than MINPV1)

10. RECEIVE

BACnet-SimpleACK-PDU |
 (BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE)

11. IF (a BACnet-SimpleACK-PDU was received) THEN
 VERIFY Present_Value = MINPV1

ELSE

VERIFY Present_Value = PV2

WRITE Present_Value = MINPV1

12. IF (Min_Pres_Value is writable) THEN

READ STG1 = Stages, ARRAY INDEX = 1

TRANSMIT WriteProperty-Request

'Object-Identifier' = (the Staging object under test),
 'Property Identifier' = Min_Pres_Value,
 'Property Value' = (MINPV2: MINPV1 < MINPV2 < (STG1.Limit - STG1.Deadband))

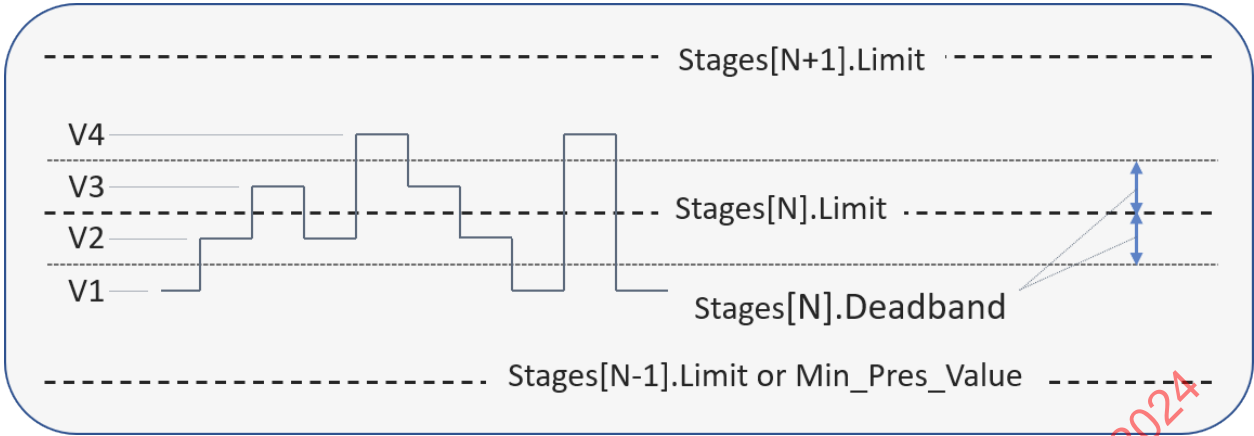
WAIT **Internal Processing Fail Time**

VERIFY Present_Value = MINPV2

7.3.2.50.2 Present_Stage Evaluation

Purpose: To verify that Present_Stage evaluates correctly based on the Present_Value, stage limits, and deadband values.

Test Concept: Present_Value is written with different values that exercise the Present_Stage evaluation algorithm. After each write to Present_Value, Present_Stage is read to verify that the algorithm evaluates correctly.



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. If supported, Deadband shall be configured with a non-zero value for stage N ($\text{Stage}[N].\text{deadband} \neq 0$). At the start of the test, the Staging object is configured with $\text{Present\_Value} = V1$.

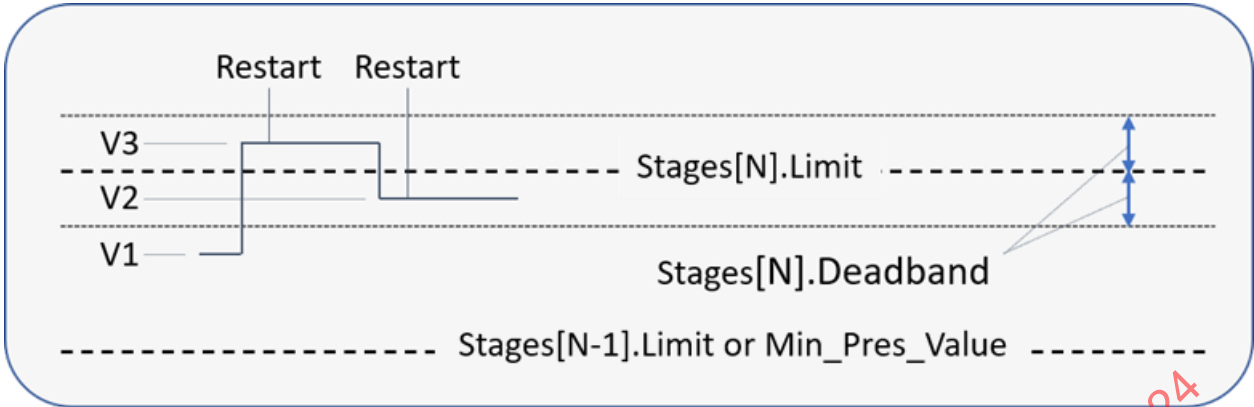
Test Steps:

1. READ $N = \text{Present\_Stage}$
2. VERIFY $\text{Present\_Value} = V1$
3. If ($\text{Stages}[N].\text{Deadband} > 0$) THEN {
 - WRITE $\text{Present\_Value} = (V2: \text{Stages}[N].\text{Limit} - \text{Stages}[N].\text{Deadband} < V2 < \text{Stages}[N].\text{Limit})$
 - VERIFY $\text{Present\_Stage} = N$
 - WRITE $\text{Present\_Value} = (V3: \text{Stages}[N].\text{Limit} < V3 < \text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband})$
 - VERIFY $\text{Present\_Stage} = N$
 - WRITE $\text{Present\_Value} = V2$
 - VERIFY $\text{Present\_Stage} = N$
 - WRITE $\text{Present\_Value} = (V4: \text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband} < V4 < \text{Stages}[N+1].\text{Limit})$
 - VERIFY $\text{Present\_Stage} = N+1$
 - WRITE $\text{Present\_Value} = V3$
 - VERIFY $\text{Present\_Stage} = N+1$
 - WRITE $\text{Present\_Value} = V2$
 - VERIFY $\text{Present\_Stage} = N+1$
 - WRITE $\text{Present\_Value} = V1$
 - VERIFY $\text{Present\_Stage} = N$
4. WRITE $\text{Present\_Value} = V4$
5. VERIFY $\text{Present\_Stage} = N+1$
6. WRITE $\text{Present\_Value} = V1$
7. VERIFY $\text{Present\_Stage} = N$

7.3.2.50.3 Present_Stage Evaluates on Restart

Purpose: To verify that Present_Stage is re-evaluated on device restart.

Test Concept: Present_Value is written with a value, $V3$, that exceeds $\text{Stages}[N].\text{limit}$ but does not exceed the deadband threshold and cause a change to Present_Stage . The IUT is restarted and Present_Stage is read to verify that it is now $(N+1)$. Present_Value is then written with a value, $V2$, that is below $\text{Stages}[N].\text{limit}$ but above the deadband threshold so Present_Stage remains at $(N+1)$. The IUT is restarted and Present_Stage is read to verify that it is now N .



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. Deadband shall be configured with a non-zero value for stage N ($\text{Stage}[N].\text{Deadband} > 0$). If deadband for stage N cannot be configured this way in a Staging object which does not support Default_Present_Value, this test shall be skipped. At the start of the test, the Staging object is configured with Present_Value = V1 and Present_Stage = N. If the IUT supports remote Target_References, then at least 1 shall be set to an object outside the IUT.

Test Steps:

1. VERIFY Present_Stage = N
2. VERIFY Present_Value = V1
3. WRITE Present_Value = (V3: $\text{Stages}[N].\text{Limit} < V3 < (\text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband})$)
4. VERIFY Present_Stage = N
5. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
 - TRANSMIT ReinitializeDevice-Request,
 - 'Reinitialized State of Device' = WARMSTART
 - 'Password' = (any valid password)
 - RECEIVE BACnet-SimpleACK-PDU
 } ELSE {
 - MAKE (power cycle the IUT to make it reinitialize)
 }
6. WAIT for the IUT to complete its restart
7. CHECK(that the IUT wrote to all Target References which are outside the device)
8. VERIFY Present_Value = V3
9. VERIFY Present_Stage = N+1
10. WRITE Present_Value = (V2: $(\text{Stages}[N].\text{Limit} - \text{Stages}[N].\text{Deadband}) < V2 < \text{Stages}[N].\text{Limit}$)
11. VERIFY Present_Stage = N+1
12. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
 - TRANSMIT ReinitializeDevice-Request,
 - 'Reinitialized State of Device' = WARMSTART
 - 'Password' = (any valid password)
 - RECEIVE BACnet-SimpleACK-PDU
 } ELSE {
 - MAKE (power cycle the IUT to make it reinitialize)
 }
13. WAIT for the IUT to complete its restart
14. CHECK(that the IUT wrote to all Target References which are outside the device)
15. VERIFY Present_Value = V2
16. VERIFY Present_Stage = N+1

7.3.2.50.4 Default_Present_Value is Abided on Restart

Purpose: To verify that Default_Present_Value defines the Staging object's value on device restart.

Test Concept: A staging object which contains Default_Present_Value. The stage associated with Default_Present_Value is S1. The staging object starts with the value V2, which evaluates to a different stage, S2. The IUT is restarted, and it is verified that the staging object takes on Default_Present_Value, changes to the stage S1 and performs the associated

writes. The IUT is restarted again, and it is verified that the staging object maintains its value, remains in stage S1 and performs the associated writes for the stage S1.

Configuration Requirements: If the IUT supports remote Target_References, then at least 1 shall be set to an object in the TD.

Test Steps:

1. VERIFY Default_Present_Value = V1
2. VERIFY Present_Value = V2
3. VERIFY Present_Stage = S2
4. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART
'Password' = (any valid password)
RECEIVE BACnet-SimpleACK-PDU
} ELSE {
MAKE (power cycle the IUT to make it reinitialize)
}
5. WAIT for the IUT to complete its restart
6. CHECK(that the IUT wrote to all Target References which are outside the device)
7. VERIFY Present_Value = V1
8. VERIFY Present_Stage = S1
9. IF (ReinitializeDevice - WARMSTART execution is supported) THEN {
TRANSMIT ReinitializeDevice-Request,
'Reinitialized State of Device' = WARMSTART
'Password' = (any valid password)
RECEIVE BACnet-SimpleACK-PDU
} ELSE {
MAKE (power cycle the IUT to make it reinitialize)
}
10. WAIT for the IUT to complete its restart
11. CHECK(that the IUT wrote to all Target References which are outside the device)
12. VERIFY Present_Value = V1
13. VERIFY Present_Stage = S1

7.3.2.50.5 Writing to Target References

Purpose: To verify that a change of Present_Stage results in the target references being written as per the stage definition.

Test Concept: A Stage object, O1, is selected for testing. O1's Present_Value is written with a value that results in a change of Present_Stage. Each Present_Value of the Target_References is monitored to verify that its value is set in accordance with Stage[Present_Stage].Values. O1's Present_Value is written again with a value that returns Present_Stage to its initial value. Again, the Target_References are monitored to verify that they have been written with the appropriate values.

Configuration Requirements: Target_References is configured with references to existing binary objects with writable Present_Value properties. The Stages property is configured with at least two stages, X and Y, such that Stages[X].Values <> Stages[Y].Values. Present_Stage shall be X at the start of the test. Throughout the test, O1 is expected to be properly configured such that Reliability is NO_FAULT_DETECTED.

Test Steps:

1. VERIFY Present_Stage = X
2. WRITE Present_Value = (any value that causes Present_Stage to change to Y)
3. VERIFY Present_Stage = Y
4. REPEAT J = (1 ... Target_References[0]) = DO {
READ O = Target_References, ARRAY INDEX = J
VERIFY O, Present_Value = Stages[Y].Values[J]
}
5. WRITE Present_Value = (any value that causes Present_Stage to change to X)

6. VERIFY Present_Stage = X
7. REPEAT J = (1..Target_References[0]) = DO {
 - READ O = Target_References, ARRAY INDEX = J
 - VERIFY O, Present_Value = Stages[X].Values[J]

7.3.2.50.6 Stage Value Bitstring is Same Length as Target_References

Purpose: To verify that the bitstring length for the Values component of each stage is equal and corresponds to the number of entries in the Target_References property.

Test Concept: For each staging object in the IUT, the Stages and Target_References properties are read. For each object, the length of the 'Values' bitstring from the first stage is extracted. This length is compared to the length of the 'Values' bitstring in every other stage and the size of the Target_References property to verify equality.

Configuration Requirements: None.

Test Steps:

1. REPEAT O = (each Staging object in the IUT) DO {
 - READ NS = O, Stages, ARRAY INDEX = 0
 - READ STG1 = Stages, ARRAY INDEX = 1
 - NUMBITS = (number bits in STG1.Values)
 - REPEAT N = (2 through NS) DO {
 - check that the length of Stages[1].Values equals length of Stages[N].Values.
 - READ STGN = Stages, ARRAY INDEX = N
 - IF number of bits in STGN.Values <> NUMBITS THEN
 - ERROR "Length of the Values bitstrings are not the same in all stages."
- VERIFY Target_References = NUMBITS, ARRAY INDEX = 0

7.3.2.50.7 Max_Pres_Value Equals Last Stage Limit

Purpose: To verify that Max_Pres_Value is equivalent to the Limit defined in the last Stage.

Test Concept: Max_Pres_Value is read and checked for equality with the Limit defined in the last element of the Stages array.

Configuration Requirements: None.

Test Steps:

1. READ N = Stages, ARRAY INDEX = 0
2. VERIFY Max_Pres_Value = Stages[N].Limit

7.3.2.50.8 CONFIGURATION_ERROR when Min_Pres_Value is too Large

Purpose: To verify that Reliability has a value of CONFIGURATION_ERROR when Min_Pres_Value has a value greater than Stages[1].Limit - Stages[1].Deadband.

Test Concept: Min_Pres_Value is made to exceed the value of Stages[1].Limit - Stages[1].Deadband by first writing directly to Min_Pres_Value, then by making a change to Stages[1].Limit, and then by making a change to Stages[1].Deadband. After each modification, if it is successful, Reliability is verified to have a value of CONFIGURATION_ERROR and then the modification is reversed, and Reliability is verified to have a value of NO_FAULT_DETECTED.

Configuration Requirements: At the start of the test, the Staging object used for this test, O1, shall be properly configured such that Reliability = NO_FAULT_DETECTED. At the start of the test, Present_Value shall be equal to Min_Pres_Value.

Test Steps:

1. READ MINPV1 = Min_Pres_Value
2. VERIFY Present_Value = MINPV1
3. VERIFY Reliability = NO_FAULT_DETECTED
4. READ STG1 = Stages, ARRAY INDEX = 1
5. SL = STG1.Limit
6. SV = STG1.Values
7. SD = STG1.Deadband
8. IF (Min_Pres_Value is writable) THEN
 - TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = O1
 - 'Property Identifier' = Min_Pres_Value ,
 - 'Property Value' = (MINPV2: where MINPV2 > (SL-SD))
 - RECEIVE
 - BACnet-SimpleACK-PDU |
 - (BACnet-Error-PDU,
 - 'Error Class' = Property,
 - 'Error Code' = VALUE_OUT_OF_RANGE)
 - IF (a BACnet-SimpleACK-PDU was received) THEN
 - VERIFY Min_Pres_Value = MINPV2
 - VERIFY Reliability = CONFIGURATION_ERROR
 - VERIFY Present_Value = MINPV2
 - VERIFY Present_Stage = 1
 - WRITE Min_Pres_Value = MINPV1
 - VERIFY Reliability = NO_FAULT_DETECTED
 - ELSE
 - VERIFY Present_Value = MINPV1
 - VERIFY Reliability = NO_FAULT_DETECTED
9. IF (Stages is writable) THEN
 - TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = (the staging object under test),
 - 'Property Identifier' = Stages,
 - 'Property Array Index' = 1,
 - 'Property Value' = {
 - Limit = (NL: where NL-SD < Min_Pres_Value),
 - Values = SV,
 - DeadBand = SD
 - RECEIVE
 - BACnet-SimpleACK-PDU |
 - (BACnet-Error-PDU,
 - 'Error Class' = PROPERTY,
 - 'Error Code' = VALUE_OUT_OF_RANGE)
 - IF (a BACnet-SimpleACK-PDU was received) THEN
 - VERIFY Stages = { Limit=NL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
 - VERIFY Reliability = CONFIGURATION_ERROR
 - VERIFY Present_Value = MINPV1
 - VERIFY Present_Stage = 1
 - WRITE Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
 - VERIFY Reliability = NO_FAULT_DETECTED
 - ELSE
 - VERIFY Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
 - VERIFY Reliability = NO_FAULT_DETECTED
 - TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = (the staging object under test),
 - 'Property Identifier' = Stages,
 - 'Property Array Index' = 1,
 - 'Property Value' = {
 - Limit=SL,

```

        Values=SV,
        Deadband=(ND: where SL-ND < Min_Pres_Value)
    }
RECEIVE
    BACnet-SimpleACK-PDU |
    (BACnet-Error-PDU,
        'Error Class' = PROPERTY,
        'Error Code' = VALUE_OUT_OF_RANGE)
IF (a BACnet-SimpleACK-PDU was received) THEN
    VERIFY Stages = { Limit=SL, Values=SV, Deadband=ND }, ARRAY INDEX = 1
    VERIFY Reliability = CONFIGURATION_ERROR
    VERIFY Present_Value = MINPV1
    VERIFY Present_Stage = 1
    WRITE Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
    VERIFY Reliability = NO_FAULT_DETECTED
ELSE
    VERIFY Stages = { Limit=SL, Values=SV, Deadband=SD }, ARRAY INDEX = 1
    VERIFY Reliability = NO_FAULT_DETECTED

```

7.3.2.50.9 COMMUNICATION_FAILURE on Failed Write to External Target Reference

Purpose: To verify that Reliability is set to COMMUNICATION_FAILURE when an attempt to write to a remote target fails.

Test Concept: The Staging object is configured with a Target_Reference aimed at an object in the TD. The Staging object's Present_Value is written such that a change to Present_Stage occurs. When the external target property is written by the IUT, the TD shall not respond. The test verifies that the write to Present_Value returns a Result(+) and Reliability is set to COMMUNICATION_FAILURE.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object, O1, in the Target_References property which is located in the TD. The Stages property shall be configured with two stages such that Stages[S].Values = {...V1...} and Stages[S+1].Values = {...V2...} where V1 and V2 correspond to the target, O1, and $V1 \neq V2$. At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED and Present_Stage = S. If no Staging object in the IUT supports external references in the Target_References property, this test shall be skipped.

Test Steps:

1. READ S = Present_Stage
2. WRITE Present_Value = (X: a value that will change Present_Stage to S+1)
3. RECEIVE WriteProperty-Request,
 'Object Identifier' = O1,
 'Property Identifier' = Present_Value,
 'Property Value' = V2
4. WAIT (Number_Of_APDU_Retries + 1) * (Internal Processing Fail Time + APDU_Timeout)
5. VERIFY Reliability = COMMUNICATION_FAILURE

7.3.2.50.10 Fault Indicated on Failed Write to Local Target Reference

Purpose: To verify that Reliability is set when an attempt to write to a local target fails.

Test Concept: The Staging object is configured with a Target_Reference aimed at an object in the IUT which is not writable or non-existent. The Staging object's Present_Value is written such that a change to Present_Stage occurs. The test verifies that the write to the Staging object's Present_Value returns a Result(+) and Reliability is set to indicate a failure to write one of the targets.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object, O1, in the Target_References property, which is local to the IUT, yet not writable or non-existent. The Stages property shall be configured with two stages such that Stages[S].Values = {...V1...} and Stages[S+1].Values = {...V2...} where V1 and V2 correspond to the target, O1, and $V1 \neq V2$. At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED and Present_Stage = S. If no Staging object can be configured to reference a non-writable or non-existent local object, this test shall be skipped.

Test Steps:

1. READ S = Present_Stage
-- cause the staging object to write to the non-existent or non-writable target object
2. WRITE Present_Value = (X: a value that will change Present_Stage to S+1)
3. VERIFY Reliability $\neq$ NO_FAULT_DETECTED

7.3.2.50.11 Out_Of_Service, Status_Flags, and Reliability for Staging Object

Purpose: To verify that Present_Value and Reliability are writable when Out_Of_Service is TRUE, to verify the relationship between Out_Of_Service, Status_Flags, and Reliability, and to verify that writes to Target_References only occur when Out_Of_Service is FALSE.

Test Concept: The Out_Of_Service property is set to TRUE and the value of the Status_Flags property is validated. Present_Value is modified to verify that Present_Stage evaluates but writes to Target_References do not occur. If the IUT supports Reliability values other than NO_FAULT_DETECTED, writability for that property is tested and the value of the Status_Flags property is validated. The Out_Of_Service property is set to FALSE and the value of the Status_Flags property is validated. The Present_Value for one of the Target_References is checked to verify that it has the correct value, indicative of a write that occurred when transitioning Out_Of_Service from TRUE to FALSE.

Configuration Requirements: The Staging object used for this test shall be configured with at least one object in the Target_References property. The Stages property shall be configured with two stages such that Stages[S].Values = {V1...} and Stages[S+1].Values = {V2...} where V1 $\neq$ V2. At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED and Present_Stage = S.

Test Steps:

1. READ SF1 = Status_Flags
2. VERIFY Reliability = NO_FAULT_DETECTED
3. VERIFY Present_Stage = S
4. READ O1 = Target_References, ARRAY INDEX = 1
5. VERIFY O1, Present_Value = V1
6. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = TRUE
ELSE
 MAKE (Out_Of_Service TRUE)
7. VERIFY Out_Of_Service = TRUE
8. VERIFY Status_Flags = (?, ?, ?, TRUE)
9. WRITE Present_Value = (PV: (Stages[S].Limit + Stages[S].Deadband) < PV < Stages[S+1].Limit)
10. VERIFY Present_Value = PV
11. VERIFY Present_Stage = S+1
12. VERIFY O1, Present_Value = V1
13. IF (the IUT supports Reliability values other than NO_FAULT_DETECTED) THEN
 REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
 NO_FAULT_DETECTED) DO {
 WRITE Reliability = X
 VERIFY Reliability = X
 VERIFY Status_Flags = (?, TRUE, ?, TRUE)
 WRITE Reliability = NO_FAULT_DETECTED
 VERIFY Reliability = NO_FAULT_DETECTED
 VERIFY Status_Flags = (?, FALSE, ?, TRUE)
 }
 }
14. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
ELSE
 MAKE (Out_Of_Service FALSE)
15. VERIFY Status_Flags = SF1
16. VERIFY Reliability = NO_FAULT_DETECTED
17. IF (Present_Stage = S+1) THEN
 VERIFY O1, Present_Value = V2

7.3.2.50.12 Stages Array Sizing Test

Purpose: This test case verifies that, when the size of the of the Stages array is changed by writing to the ARRAY INDEX, the size of the array changes accordingly and any new entries are properly initialized.

Test Concept: The Stages array is increased by writing the array size. It is verified that the Stages property is extended and that the new entries contain 'Limit' = 0.0, 'Values' = {0...0}, and 'Deadband' = 0.0. Reliability is verified to be CONFIGURATION_ERROR. Present_Stage is verified to be 1. Present_Value is verified to be Min_Pres_Value. If the Stage_Names property is present, the size of the array is checked to verify that it matches the size of the Stages array.

Throughout the test, the array size of the Stage_Names property is checked to verify it is consistent with the array size of the Stages property.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED and Present_Stage = S. The size of the Stages array is greater than 1 and less than the maximum array size.

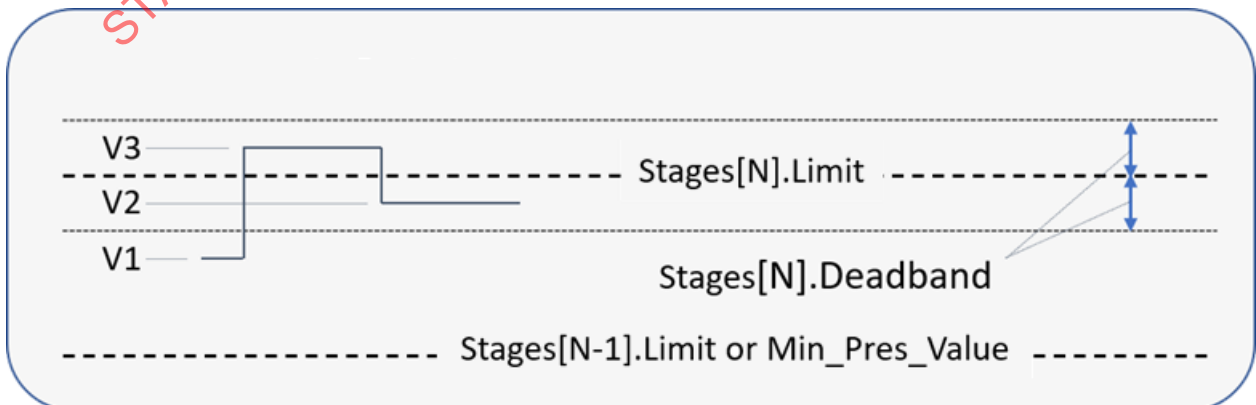
Test Steps:

1. READ N = Stages, ARRAY INDEX = 0
2. IF (Stage_Names is present) THEN {
 VERIFY Stage_Names = N, ARRAY INDEX = 0
}
3. READ NT = Target_References, ARRAY INDEX = 0
4. WRITE Stages = N+X, ARRAY INDEX = 0 -- where (X >= 1)
5. VERIFY Stages = N+X, ARRAY INDEX = 0
6. VERIFY Stages = (0.0, {0...0}, 0.0), ARRAY INDEX = N+X -- where the number of bits in Values is NT
7. VERIFY Reliability = CONFIGURATION_ERROR
8. VERIFY Present_Stage = 1
9. READ MV = Min_Pres_Value
10. VERIFY Present_Value = MV
11. IF (Stage_Names is present) THEN {
 VERIFY Stage_Names = N+X, ARRAY INDEX = 0
 WRITE Stages = N, ARRAY INDEX = 0
 VERIFY Stage_Names = N, ARRAY INDEX = 0
}

7.3.2.50.13 Present_Stage Evaluates on Change to Stages Property

Purpose: To verify that Present_Stage gets re-evaluated when the Stages property is changed.

Test Concept: Present_Value is written with a value, V3, that exceeds Stages[N].limit but does not exceed the deadband threshold and cause a change to Present_Stage. The Stages property is written with a new value such that Stage[N] is unaffected by the change. Present_Stage is read to verify that it is now (N+1). Present_Value is then written with a value, V2, that is below Stages[N].limit but above the deadband threshold so Present_Stage remains at (N+1). The Stages property is written with a new value such that Stage[N] is unaffected by the change. Present_Stage is read to verify that it is now N.



Configuration Requirements: The Staging object used for this test must be configured with at least two stages. Deadband shall be configured with a non-zero value for stage N ($\text{Stage}[N].\text{Deadband} > 0$). If deadband for stage N cannot be configured this way, this test shall be skipped. At the start of the test, the Staging object is configured with $\text{Present\_Value} = V1$ and $\text{Present\_Stage} = N$.

Test Steps:

1. VERIFY $\text{Present\_Stage} = N$
2. VERIFY $\text{Present\_Value} = V1$
3. READ $\text{STAGES1} = \text{Stages}$
4. WRITE $\text{Present\_Value} = (V3: \text{Stages}[N].\text{Limit} < V3 < (\text{Stages}[N].\text{Limit} + \text{Stages}[N].\text{Deadband}))$
5. VERIFY $\text{Present\_Stage} = N$
6. IF (Stages is writable) THEN
 WRITE $\text{Stages} = (\text{STAGES2}: \text{any valid value different from STAGES1 but with the same value for Stage}[N])$
ELSE
 MAKE $\text{Stages} = (\text{STAGES2}: \text{any valid value different from STAGES1 but with the same value for Stage}[N])$
7. VERIFY $\text{Present\_Value} = V3$
8. VERIFY $\text{Present\_Stage} = N+1$
9. WRITE $\text{Present\_Value} = (V2: (\text{Stages}[N].\text{Limit} - \text{Stages}[N].\text{Deadband}) < V2 < \text{Stages}[N].\text{Limit})$
10. VERIFY $\text{Present\_Stage} = N+1$
11. IF (Stages is writable) THEN
 WRITE $\text{Stages} = (\text{STAGES3}: \text{any valid value different from STAGES2 but with the same value for Stage}[N])$
ELSE
 MAKE $\text{Stages} = (\text{STAGES3}: \text{any valid value different from STAGES2 but with the same value for Stage}[N])$
12. VERIFY $\text{Present\_Value} = V2$
13. VERIFY $\text{Present\_Stage} = N$

7.3.2.50.14 CONFIGURATION_ERROR when Limits are Out of Order

Purpose: To verify that Stages defined in the staging object are arranged in ascending order and, if not, Reliability is set to CONFIGURATION_ERROR.

Test Concept: Write Stages out of order; use specific values that violate the limit value ascension rule. Verify that the object identifies the problem and sets Reliability.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED.

Test Steps:

1. VERIFY Reliability = NO_FAULT_DETECTED
2. READ $\text{NS} = \text{Stages}, \text{ARRAY INDEX} = 0$
3. $N = (\text{any value where } -1 \leq N < \text{NS})$
4. WRITE $\text{Stages} = \{$
 $\text{Limit} = (\text{LIM}: \text{where LIM} > \text{Stages}[N+1].\text{Limit}),$
 $\text{Values} = (\text{any valid value of the correct length}),$
 $\text{Deadband} = (\text{any valid value})$
 $\}, \text{ARRAY INDEX} = N$
5. VERIFY Reliability = CONFIGURATION_ERROR
6. VERIFY $\text{Present\_Value} = \text{Min\_Pres\_Value}$
7. VERIFY $\text{Present\_Stage} = 1$

7.3.2.50.15 CONFIGURATION_ERROR when Deadband < 0

Purpose: To verify that Stages defined in the staging object do not have a Deadband value less than 0, or if they do, when Deadband is less than 0, Reliability is set to CONFIGURATION_ERROR.

Test Concept: Write an entry in the Stages property, changing the deadband to a negative value. Verify that the either the write fails, or that Reliability is set to CONFIGURATION_ERROR.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED.

Test Steps:

1. VERIFY Reliability = NO_FAULT_DETECTED
2. READ NS = Stages, ARRAY INDEX = 0
2. N = (any value where $1 \leq N < NS$)
3. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the staging object under test),
 'Property Identifier' = Stages,
 'Property Array Index' = N,
 'Property Value' = {
 Limit = Stages[N].Limit,
 Values = Stages[N].Values,
 Deadband = (any negative value)
 }
4. RECEIVE
 BACnet-SimpleACK-PDU |
 (BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN {
 VERIFY Reliability = CONFIGURATION_ERROR
 VERIFY Present_Value = Min_Pres_Value
 VERIFY Present_Stage = 1
 }

7.3.2.50.16 CONFIGURATION_ERROR when Stages Size is less than Two

Purpose: To verify that the Stages array has a minimum length of two, and if not, Reliability is set to CONFIGURATION_ERROR.

Test Concept: Write the Stages property, without an array index, setting the length of the array to 1. Verify that either the write fails, or that Reliability is set to CONFIGURATION_ERROR.

Configuration Requirements: At the start of the test, the Staging object is properly configured such that Reliability = NO_FAULT_DETECTED.

Test Steps:

1. VERIFY Reliability = NO_FAULT_DETECTED
2. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the staging object under test),
 'Property Identifier' = Stages,
 'Property Array Index' = 0,
 'Property Value' = (0 or 1)
3. RECEIVE
 BACnet-SimpleACK-PDU |
 (BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = VALUE_OUT_OF_RANGE)
5. IF (a BACnet-SimpleACK-PDU was received) THEN {
 VERIFY Reliability = CONFIGURATION_ERROR
 VERIFY Present_Value = Min_Pres_Value
 VERIFY Present_Stage = 1
 }

7.3.2.50.17 Stage_Names and Stages Size Equality Test

Purpose: To verify that the size of the Stage_Names array is equal to the size of the Stages array.

Test Concept: Verify that the Stages array and Stage_Names array are of the same length.

Test Steps:

1. READ N = Stages, ARRAY INDEX = 0
2. VERIFY Stage_Names = N, ARRAY INDEX = 0

7.3.2.50.18 Stage_Names Array Sizing Test

Purpose: This test case verifies that, when the size of the Stage_Names array is changed by writing to the ARRAY INDEX, the size of the array and the size of the Stages array changes accordingly and new values added to the Stages array are initialized to contain 'Limit' = 0.0, 'Values' = {0...0}, and 'Deadband' = 0.0, and Reliability is set to CONFIGURATION_ERROR.

Test Concept: Resize the Stage_Names array larger by writing the size and verify that Stages is also resized. Shrink the array back and verify Stages. Resize once more by writing the whole array and verify that Stages resizes correctly. Each time verify that new stages are correct.

Configuration Requirements: If the Stages property is not resizable by writing to it, this test shall be skipped.

Test Steps:

1. READ N = Stage_Names, ARRAY INDEX = 0
2. WRITE Stage_Names = N+1, ARRAY INDEX = 0
3. VERIFY Stages = N+1, ARRAY INDEX = 0
4. VERIFY Stages = {Limit=0.0, Values={0...0}, Deadband=0.0}, ARRAY INDEX = N+1
5. WRITE Stage_Names = N, ARRAY INDEX = 0
6. VERIFY Stages = N, ARRAY INDEX = 0
7. WRITE Stage_Names = (an array, of strings, with a length, N2, which the IUT will accept other than N)
8. VERIFY Stages = N2, ARRAY INDEX = 0
9. VERIFY Stages = (an array of length N2 of stages consistent with the object's configuration)
10. IF (N2 > N) THEN {
 - REPEAT J = (N ... N2) {
 - VERIFY Stages = {Limit=0.0, Values={0...0}, Deadband=0.0}, ARRAY INDEX = J

7.3.2.50.19 Target_References Array Sizing Test

Purpose: To verify that a change to size of the Target_References array results in an equivalent change to the length of the 'Values' portion of all elements of the Stages property and that new bits in the 'Values' are set to '0'.

Test Concept: Resize the Target_References array larger and verify that the values field in each stage is updated with new bits. Resize the array smaller and verify that the values field in each stage is resized smaller.

Configuration Requirements: The staging object is configured with at least 1 Target_Reference.

Test Steps:

1. READ STAGES1 = Stages
2. NTR = (length of STAGES1[0].Values)
3. WRITE Target_References = NTR+1, ARRAY INDEX = 0
4. REPEAT J = (1 ... STAGES1[0]) DO {
 - VERIFY Stages = {
 - Limit=STAGES1[J].Limit,
 - Values=(the value of STAGES1[J].Values with 1 more 0 tacked on the end),
 - DeadBand=STAGES1 [J].Deadband
5. WRITE Target_References = NTR, ARRAY INDEX = 0
6. REPEAT J = (1 ... STAGES1[0]) DO {

```

VERIFY Stages = {
    Limit=STAGES1[J].Limit,
    Values=(the value of STAGES1[J].Values),
    DeadBand=STAGES1[J].Deadband
}, ARRAY INDEX = J
}

```

7.3.2.50.20 Writing Target_References with an Unsupported External Reference

Purpose: To verify the correct Result(-) when Target_References does not support objects in an external device.

Test Concept: Attempt writing Target_References of a Staging object with an external object reference. Verify the IUT returns the correct Result(-).

Configuration Requirements: The IUT is configured with a Staging Value object which does not support references to external objects. If the IUT cannot be configured this way, this test shall be skipped.

Test Steps:

1. READ X = Target_References
2. TRANSMIT WriteProperty-Request,
 - 'Object Identifier' = (the staging object under test),
 - 'Property Identifier' = Target_References,
 - 'Property Array Index' = 1,
 - 'Property Value' = (a reference to a binary object in the TD)
3. RECEIVE BACnet-Error-PDU,
 - 'Error Class' = PROPERTY
 - 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED

8. APPLICATION SERVICE INITIATION TESTS

The test cases defined in this clause shall be used to verify that a BACnet device correctly initiates the specified application service. BACnet devices shall be tested for the ability to initiate each application service for which the PICS indicates that initiation is supported.

For each application service included in this clause several test cases are defined that collectively test the various options and features defined for the service in the BACnet standard. A test case is a sequence of one or more messages that are exchanged between the implementation under test (IUT) and the testing device (TD) in order to determine if a particular option or feature is correctly implemented. Multiple test cases that have a similar or related purpose are collected into test groups.

For each test case a sequence of one or more messages that are to be exchanged is described. A passing result occurs when the IUT and TD exchange messages exactly as described in the test case. Any other combinations of messages constitute a failure of the test. Some test cases are not valid unless some other test defined in this standard has already been executed and the IUT passed this test. These dependencies are noted in the test case description.

Because the purpose of the tests in this clause is to test initiation of BACnet service requests, many of them indicate that the first step is to receive a particular service request without any indication of how to cause the IUT to initiate the expected request. The assumption is that the vendor has provided a way to cause the IUT to initiate the request. The method used to cause the IUT to take these actions is a local matter.

Under some circumstances an IUT may be unable to demonstrate conformance to a particular test case because the test applies to a feature that requires a particular BACnet object or optional property that is not supported in the IUT. For example, a device may support the File Access services but restrict files to stream access only. Such a device would have no way to demonstrate that it could implement the record access features of the File Access services. When this type of situation occurs the IUT shall be considered to be in conformance with BACnet provided the PICS documentation clearly indicates the restriction. Failure to document the restriction shall constitute non conformance to the BACnet standard. All features and optional parameters for BACnet application services shall be supported unless a conflict arises because of unsupported objects or unsupported optional properties.

8.1 AcknowledgeAlarm Service Initiation Tests

8.1.1 AcknowledgeAlarm Service Initiation Test

Purpose: To verify that the IUT is capable of acknowledging alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotification services.

Configuration: For this test, the tester shall choose 1 object, O1, in the TD, which is configured to send event notifications to the IUT. The tester places O1 into an alarm state such that the transition requires an acknowledgment.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class
 object for the IUT),
 'Initiating Device Identifier' = TD,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),
 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1 or OFFNORMAL if S1 is an off-normal state,
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgement' = (any valid value)

5. TRANSMIT BACnet-SimpleACK-PDU

8.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the 'Initiating Device Identifier' Parameter

Purpose: To verify that the IUT is correctly implemented to send AcknowledgeAlarm directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification, in alarms and events that are reported to the IUT via the ConfirmedEventNotification and UnconfirmedEventNotification services. Requests having the 'Initiating Device Identifier' parameter equal-to and different from SOURCE, are both tested.

Test Concept: Two times a purposefully constructed event notification is sent, and it is observed that IUT is acknowledging directly to the device indicated by the 'Initiating Device Identifier' parameter of the event notification.

Configuration: For this test, the tester shall choose an object, O1, and tester places O1 into an alarm state such that the transition requires an acknowledgment. Then purposefully the EventNotification packet which is sent is crafted to represent that a Notification Forwarder was involved, so though SOURCE is from the TD, the O1 resides in a different device TD1. Then the steps are repeated but with 'Initiating Device Identifier' representing that O1 is in TD and thus the same device. Each time it is observed that the AcknowledgeAlarm is sent to the device represented as the 'Initiating Device Identifier'.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
 'Initiating Device Identifier' = TD1, -- representing that O1 is in different device from SOURCE
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),
 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
2. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
3. MAKE (the IUT acknowledge O1)
4. RECEIVE AcknowledgeAlarm-Request, DESTINATION=TD1
 'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgement' = (any valid value)
5. TRANSMIT BACnet-SimpleACK-PDU
6. TRANSMIT ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
 'Subscriber Process Identifier' = (a value acceptable to the IUT configured in the Notification Class object for the IUT),
 'Initiating Device Identifier' = TD, -- representing that O1 is present in same device as SOURCE
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid value, T1),
 'Notification Class' = (the value configured in O1),
 'Priority' = (any value selected by the TD),

- 'Event Type' = (any value selected by the TD),
 'Notify Type' = ALARM | EVENT,
 'AckRequired' = TRUE,
 'From State' = (any valid value),
 'To State' = (any valid value, S1),
 'Event Values' = (any event values appropriate to the event type)
7. IF (the ConfirmedEventNotification choice was selected) THEN
 RECEIVE BACnet-SimpleACK-PDU
 8. MAKE (the IUT acknowledge O1)
 9. RECEIVE AcknowledgeAlarm-Request,
 'Acknowledging Process Identifier' = (any process identifier),
 'Event Object Identifier' = O1,
 'Event State Acknowledged' = S1, or OFFNORMAL if S1 is an off-normal state
 'Time Stamp' = T1,
 'Acknowledgement Source' = (any valid value),
 'Time of Acknowledgement' = (any valid value)
 10. TRANSMIT BACnet-SimpleACK-PDU

8.2 ConfirmedCOVNotification Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ConfirmedCOVNotification service requests. The ConfirmedCOVNotification tests are specific to a particular object type that provides intrinsic COV reporting capabilities. The IUT shall pass all of the tests for each object type that supports intrinsic COV reporting that is claimed to be supported in the PICS.

8.2.1 Change of Value Notification for Changes to Present_Value in Objects with a COV_Increment

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property in Numeric Objects.

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

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control by more than COV_Increment or which has a writable Out_Of_Service. In devices where the COV_Increment is always less than the minimal change that Present_Value can make, skip steps 8 through 10.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

Test Steps:

REPEAT X = (one supported object of each type) DO {

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

```

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

```

8.2.2 Change of Value Notification for Changes to Status_Flags Property

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed and a notification shall be received. For implementations where the Status_Flags cannot be made to change, this test shall be skipped.

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

Test Steps:

REPEAT X = (one supported object of each type) DO {

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

8.2.3 Change of Value Notification for Changes to Present_Value in Objects without a COV_Increment

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of objects that do not support COV_Increment.

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

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

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

Test Steps:

REPEAT X = (one supported object of each type) DO {

1. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (any value > 0 chosen by the TD),
 'Monitored Object Identifier' = X,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU

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

8.2.4 Deleted Clause

This test has been removed.

8.2.5 Deleted Clause

This test has been removed.

8.2.6 Deleted Clause

This test has been removed.

8.2.7 Change of Value Notification from a Loop Object Present_Value Property

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

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

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

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

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

Test Steps:

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

14. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Monitored Object Identifier' = O1
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
 WRITE O1, Out_Of_Service = FALSE

8.2.8 Deleted Clause

This test has been removed.

8.2.9 Missing Lifetime Test

Purpose: This test case verifies the special case of the SubscribeCOV where a missing Lifetime parameter shall imply an indefinite Lifetime subscription.

Test Concept: A subscription for COV notification is established with the IUT. The subscribe message shall omit the Lifetime parameter. The COV notification is received from the IUT and the 'Time Remaining' value is verified to be 0.

Test Steps:

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

8.2.10 ConfirmedCOVNotification Pulse Converter changing Present_Value

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

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

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

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (any value $\neq$ 0 chosen by the TD),
 'Monitored Object Identifier' = O1,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU

3. BEFORE **Notification Fail Time**

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

4. TRANSMIT BACnet-SimpleACK-PDU

5. TRANSMIT ReadProperty-Request,

'Object Identifier' = O1,
 'Property Identifier' = COV_Increment

6. RECEIVE BACnet-ComplexACK-PDU,

'Object Identifier' = O1,
 'Property Identifier' = COV_Increment,
 'Property Value' = (a value "increment" that will be used below)

7. IF (Out_Of_Service is writable) THEN

WRITE O1, Out_Of_Service = TRUE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O1,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (ReportedPV = the current Present_Value, new Status_Flags, and current Update_Time)

8. TRANSMIT BACnet-SimpleACK-PDU

9. IF (Present_Value is now writable) THEN

WRITE O1, Present_Value = (any value that differs from ReportedPV by less than "increment")

ELSE

MAKE (Present_Value = any value that differs from ReportedPV by less than "increment")

10. WAIT **Notification Fail Time**

11. CHECK (verify that no COV notification was transmitted)

12. IF (Present_Value is now writable) THEN

WRITE O1, Present_Value = (any value that differs from ReportedPV by an amount greater than "increment")

ELSE

MAKE (Present_Value = any value that differs from ReportedPV by an amount greater than "increment")

13. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O1,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the new Present_Value, new Status_Flags, and current Update_Time)

14. TRANSMIT BACnet-SimpleACK-PDU

15. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = (the same value used in step 1),
 'Monitored Object Identifier' = O1

16. RECEIVE BACnet-SimpleACK-PDU

17. IF (Out_Of_Service was changed in step 7) THEN

WRITE O1, Out_Of_Service = FALSE

8.2.11 ConfirmedCOVNotification Pulse Converter changing Status_Flags

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed, and a notification shall be received. For some implementations writing to the Out_Of_Service property will accomplish

this task. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped. The object identifier of the Pulse Converter object being tested is designated as O1 in the test steps below.

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control by more than COV_Increment. COV_Period is configured high enough that it does not trigger many COV notifications during the execution of the test.

Test Steps:

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

8.2.12 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Access Door objects.

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

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control, or which has a writable Out_Of_Service. If no object has a Door_Alarm_State property, then steps 9,10, and 11 shall be skipped. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, steps 5,6, and 7 shall be skipped

Test Steps:

REPEAT X = (one supported object of type Access Door) DO {

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

```

14. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (the same value used in the SubscribeCOV-Request),
    'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
    WRITE Out_Of_Service = FALSE
}

```

8.2.13 Change of Value Notification from an Access Point object

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags and Access_Event_Time properties of Access Point objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Access_Event_Time and Status_Flags of the monitored object is changed, and a notification shall be received. The properties may be changed using the WriteProperty service or by another means. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, steps 5,6, and 7 shall be skipped.

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE.

Test Steps:

```

1. TRANSMIT SubscribeCOV-Request,
    'Subscriber Process Identifier' = (PI: any value > 0 chosen by the TD),
    'Monitored Object Identifier' = X,
    'Issue Confirmed Notifications' = TRUE,
    'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = PI,
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = X,
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (the initial Access_Event, Status_Flags, Access_Event_Tag,
        Access_Event_Time, Access_Event_Credential and
        Access_Event_Authentication_Factor if X has an
        Access_Event_Authentication_Factor property)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out_Of_Service is writable) THEN
    WRITE X, Out_Of_Service = TRUE
ELSE
    MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. BEFORE Notification Fail Time
    RECEIVE ConfirmedCOVNotification-Request,
    'Subscriber Process Identifier' = PI,
    'Initiating Device Identifier' = IUT,
    'Monitored Object Identifier' = X,
    'Time Remaining' = (any value appropriate for the Lifetime selected),
    'List of Values' = (the initial Access_Event, new Status_Flags, initial Access_Event_Tag,
        Access_Event_Time, Access_Event_Credential and
        Access_Event_Authentication_Factor if X has a
        Access_Event_Authentication_Factor property)
7. TRANSMIT BACnet-SimpleACK-PDU
8. IF (Access_Event_Time is now writable) THEN
    WRITE Access_Event_Time = (any value that differs from initial Access_Event_Time)

```

- ELSE
 MAKE (Access_Event_Time = any value that differs from initial Access_Event_Time)
9. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PI,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the new values of Access_Event, Access_Event_Tag, Access_Event_Time,
 Access_Event_Credential, and Access_Event_Authentication_Factor if X
 Has Access_Event_Authentication_Factor property)
10. TRANSMIT BACnet-SimpleACK-PDU
11. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = PI,
 'Monitored Object Identifier' = X
12. RECEIVE BACnet-SimpleACK-PDU
13. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
14. CHECK (verify that no notification message has been transmitted)

8.2.14 Change of Value Notification from a Credential Data Input object

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status_Flags and Update_Time properties of Credential Data Input objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags and Update_Time properties of the monitored object is changed, and a notification shall be received. The properties may be changed using the WriteProperty service or by another means. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, steps 5,6, and 7 shall be skipped

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE.

Test Steps:

13. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (PI: any value > 0 chosen by the TD),
 'Monitored Object Identifier' = X,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PI,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the initial Present_Value, initial Status_Flags, and Update_Time
 (most recent update time when the Present_Value was updated))
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out_Of_Service is writable) THEN
 WRITE X, Out_Of_Service = TRUE
 ELSE
 MAKE (Status_Flags = any value that differs from initial Status_Flags)
6. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PI,

- 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the initial Present_Value, new Status_Flags, and Update_Time
 (most recent update time when the Present_Value was updated))
7. TRANSMIT BACnet-SimpleACK-PDU
 8. IF (Present_Value is now writable) THEN
 WRITE X, Present_Value = (any value that differs from initial Present_Value)
 ELSE
 MAKE (Present_Value = any value that differs from initial Present_Value)
 13. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PI,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the new Present_Value, new Status_Flags, and Update_Time
 (most recent update time when the Present_Value was updated))
 10. TRANSMIT BACnet-SimpleACK-PDU
 11. Verify Update_Time received in step 7.
 12. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = PI,
 'Monitored Object Identifier' = X
 13. RECEIVE BACnet-SimpleACK-PDU
 14. IF (Out_Of_Service is writable) THEN
 WRITE Out_Of_Service = FALSE
 15. CHECK (verify that no notification message has been transmitted)

8.2.15 Change of Value Notification of Staging Object Present_Value Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property in Staging objects that support COV_Increment.

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

Configuration Requirements: Select a Staging object where Present_Value is not expected to change outside the tester's control. The object is configured such that the change in Present_Value required to change stages is larger than COV_Increment. If the IUT cannot be configured with such a Staging object, this test shall be skipped.

Test Steps:

1. READ PV1 = Present_Value
 2. READ SF1 = Status_Flags
 3. READ PS1 = Present_Stage
- subscribe for COV and receive initial notification
4. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),
 'Monitored Object Identifier' = X,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
 5. RECEIVE BACnet-SimpleACK-PDU
 6. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,

- 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (PV1, SF1, PS1)
7. TRANSMIT BACnet-SimpleACK-PDU
- change Present_Value by less than COV_Increment, and not enough to change the stage
8. WRITE X, Present_Value = (PV2: a value that differs from PV1 by less than COV_Increment and which is in the range for the current stage)
9. WAIT **Notification Fail Time**
10. CHECK (verify that no COV notification was transmitted)
- change Present_Value by more than COV_Increment, but not enough to change the stage
11. WRITE X, Present_Value = (PV3: a value that differs from PV1 by an amount greater than COV_Increment and which is in the range for the current stage)
12. BEFORE NotificationFailTime
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (PV3, SF1, PS1)
13. TRANSMIT BACnet-SimpleACK-PDU
- cleanup the subscription
14. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = PID1,
 'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU

8.2.16 Change of Value Notification of Staging Object Status Flags Property

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

Test Concept: A COV subscription is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status_Flags property of the monitored object is then changed, and a notification shall be received. The value of the Status_Flags property can be changed by using the WriteProperty service or by another means. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, this test shall be skipped.

Configuration Requirements: Select a Staging object where Present_Value is not expected to change outside the tester's control.

Test Steps:

1. VERIFY Out_Of_Service = FALSE
 2. READ PV1 = Present_Value
 3. READ SF1 = Status_Flags
 4. READ PS1 = Present_Stage
- subscribe for COV and receive initial notification
5. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),
 'Monitored Object Identifier' = X,
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L
 6. RECEIVE BACnet-SimpleACK-PDU
 7. BEFORE **Notification Fail Time**
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,

'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (PV1, SF1, PS1)

8. TRANSMIT BACnet-SimpleACK-PDU

-- change Status_Flags and receive notification

9. IF Out_Of_Service is writable THEN

WRITE X, Out_Of_Service = TRUE

SF2 = (SF1 with the Out_Of_Service bit changed to 1)

ELSE

MAKE (Status_Flags = SF2, any value other than SF1)

13. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PID1,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = X,

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (PV1, SF2, PS1)

11. TRANSMIT BACnet-SimpleACK-PDU

-- cleanup the subscription and Out_Of_Service

12. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = PID1,

'Monitored Object Identifier' = X

13. RECEIVE BACnet-SimpleACK-PDU

14. IF (Out_Of_Service was changed via writing) THEN

WRITE X, Out_Of_Service = FALSE

8.2.17 Change of Value Notification of Staging Object Present Stage Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Stage property of Staging objects.

Test Concept: A COV subscription is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Stage property of the monitored object is then changed, and a notification shall be received.

Configuration Requirements: Select a Staging object, O1, where Present_Value is not expected to change outside the tester's control. The object shall be configured with Present_Value having a value, PV1, which is less than COV_Increment away from a value, PV2, which will change the current stage to a new stage, PS2. If no Staging object can be configured with a COV_increment larger than the resolution of Present_Value, this test shall be skipped.

Test Steps:

1. VERIFY Present_Value = PV1

2. VERIFY Present_Stage = PS2

3. READ SF1 = Status_Flags

4. CHECK(The difference between PV1 and PV2 is less than COV_Increment)

-- subscribe for COV and receive initial notification

5. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = (PID1: any value > 0 chosen by the TD),

'Monitored Object Identifier' = O1,

'Issue Confirmed Notifications' = TRUE,

'Lifetime' = L

6. RECEIVE BACnet-SimpleACK-PDU

7. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PID1,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = O1,

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (PV1, SF1, PS1)

8. TRANSMIT BACnet-SimpleACK-PDU

-- change Present_Value and receive notification

9. WRITE X, Present_Value = PV2

10. BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = PID1,

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = O1,

'Time Remaining' = (any value appropriate for the Lifetime selected),

'List of Values' = (PV2, SF1, PS2)

11. TRANSMIT BACnet-SimpleACK-PDU

-- cleanup the subscription

12. TRANSMIT SubscribeCOV-Request,

'Subscriber Process Identifier' = PID1,

'Monitored Object Identifier' = O1

13. RECEIVE BACnet-SimpleACK-PDU

8.3 UnconfirmedCOVNotification Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating UnconfirmedCOVNotification service requests. The UnconfirmedCOVNotification tests are specific to a particular object type that provides intrinsic COV reporting capabilities. The IUT shall pass all of the tests for each object type that is claimed to be supported in the PICS.

8.3.1 Change of Value Notification for Changes to Present_Value in Objects with a COV_Increment

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

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

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

8.3.2 Change of Value Notification for Changes to Status_Flags Property

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

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

8.3.3 Change of Value Notification for Change to Present_Value in Objects without a COV_Increment

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value property of objects that do not support COV_Increment.

Test Steps: The steps for this test case are identical to the test steps in 8.2.3 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification unicasts, and there is no acknowledgment of the unconfirmed services.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

8.3.4 Deleted Clause

This test has been removed.

8.3.5 Deleted Clause

This test has been removed.

8.3.6 Deleted Clause

This test has been removed.

8.3.7 Change of Value Notification from a Loop Object Present_Value Property

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

Test Steps: The steps for this test case are identical to the test steps in 8.2.7 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present_Value and Status_Flags.

8.3.8 Deleted Clause

This test has been removed.

8.3.9 Unsubscribed Change of Value Notifications

Unsubscribed COV notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, no subscription is required. Second, the 'Subscriber Process Identifier' parameter usually has a value of zero.

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests when no subscription for the COV notification has been made.

Test Concept: The IUT is configured to send unsubscribed COV notifications. The TD then waits for the notification. Given that there is no defined trigger, the vendor shall inform the tester how to make the IUT generate the notifications if they are not sent periodically.

Test Steps:

1. MAKE (the IUT send an unsubscribed COV notification)
2. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = (any valid object identifier),
 'Time Remaining' = 0,
 'List of Values' = (any valid properties and values from the monitored object)

8.3.10 Device Restart Notifications

Purpose: To verify that the IUT initiates UnconfirmedCOVNotification service requests to each entry in its Restart_Notification_Recipients property when it resets.

Test Concept: The IUT is configured to send restart notifications and is then reset. The TD checks for the restart notifications.

Device restart notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, subscription is made through the Restart_Notification_Recipients property instead of SubscribeCOV. Second, the 'Subscriber Process Identifier' parameter always has a value of zero.

Configuration Requirements: For each Recipient of the Restart_Notification_Recipients property in the IUT which is of the device form, there shall be a device on the network that will answer Who-Is requests so that the IUT can determine addressing information before sending the restart notification.

Notes to tester: Not all IUTs can accurately differentiate between the types of restart reasons and thus no requirements are placed on the value returned in the restart notification(s). The test shall pass regardless of the order in which the restart notifications are sent to the recipients. If the Restart_Notification_Recipients list has multiple recipients, then the Time Of Device Restart value is expected to be the same in all notifications resulting from the same restart.

Test Steps:

- ```

1. IF (Restart_Notification_Recipients is writable) THEN
 WRITE(Restart_Notification_Recipients = any non-empty list of Recipients)
ELSE
 MAKE (Restart_Notification_Recipients contain any non-empty list of Recipients)
2. MAKE(the IUT reset)
3. REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotification-Request,
 DESTINATION = X,
 'Subscriber Process Identifier' = 0,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = (the IUT Device Identifier),
 'Time Remaining' = 0,
 'List of Values' = (System_Status=OPERATIONAL,
 Time_Of_Device_Restart = (T2),
 Last_Restart_Reason=(any valid restart reason, R))
 }
4. VERIFY Time_Of_Device_Restart = T2
5. VERIFY Last_Restart_Reason = R
6. IF (T2 is not a sequence number) THEN
 VERIFY Local_Time $\approx$ T2
ELSE
 MAKE(the IUT reset)
 REPEAT X = (each entry in the Restart_Notification_Recipients) DO {
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotification-Request,
 DESTINATION = X,
 'Subscriber Process Identifier' = 0,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = (the IUT Device Identifier),
 'Time Remaining' = 0,
 'List of Values' = (System_Status=OPERATIONAL,
 Time_Of_Device_Restart = (T3),
 Last_Restart_Reason=(any valid restart reason, R))
 }
7. CHECK (T3 > T2)

```

### 8.3.11 COVU Recipients Notifications

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

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

Configuration Requirements: COVU\_Recipients property shall be non-empty and contain at least one device and one address based recipient. The COVU\_Period shall be non-zero.

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

### Test Steps:

1. REPEAT X = (each entry in the COVU Recipients) DO {

```

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

```

### 8.3.12 UnconfirmedCOVNotification Pulse Converter changing Present\_Value

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

Test Concept: This test is the same as 8.2.10 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

Notes to Tester: The IUT may initiate additional COVNotifications. The final COVNotification shall accurately reflect Present\_Value and Status\_Flags.

### 8.3.13 UnconfirmedCOVNotification Pulse Converter changing Status\_Flags

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

Test Concept: This test is the same as 8.2.11 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no BACnet-SimpleACK-PDU returned in acknowledgment of the unconfirmed services.

### 8.3.14 Change of Value Notification from an Access Door object Present\_Value, Status\_Flags and Door\_Alarm\_State property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value, Status\_Flag and Door\_Alarm\_State property of Access Door objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.12 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

**8.3.15 Change of Value Notification from an Access Point Object**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flag and Access\_Event\_Time properties of Access Point objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.13 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

**8.3.16 Change of Value Notification from a Credential Data Input Object**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags and Update\_Time properties of Credential Data Input objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.14 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

**8.3.17 Change of Value Notification of Staging Object Present\_Value Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.15 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

**8.3.18 Change of Value Notification of Staging Object Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.16 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

**8.3.19 Change of Value Notification of Staging Object Present\_Stage Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Stage property of Staging objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.17 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

**8.4 ConfirmedEventNotification Service Initiation Tests**

This clause defines the tests necessary to demonstrate support for initiating ConfirmedEventNotification service requests. The ConfirmedEventNotification tests are specific to the event detection algorithm used. For each object type that supports intrinsic event reporting the IUT shall pass the tests for the event detection algorithm that applies to that object type. If the IUT supports the Event Enrollment object it shall pass the tests for the event detection algorithm that corresponds to each event type supported.

For devices that implement a read-only Recipient\_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device cannot contain a Notification Forwarder object, this clause and all subclauses shall be omitted from the testing protocol.



For devices that implement a read-only Recipient\_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device can contain a Notification Forwarder object, this clause and all subclauses shall be applied to the Notification Forwarder objects and not to the Notification Class objects.

#### 8.4.1 CHANGE\_OF\_BITSTRING Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the Change of Bitstring event algorithm.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed to a value that is one of the values designated in pAlarmValues. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. pMonitoredValue is then changed to a value corresponding to a NORMAL state. After the time delay the object should enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN  
    WRITE (pMonitoredValue) = (a value from the pAlarmValues list after pBitmask is applied)  
ELSE  
    MAKE (pMonitoredValue have a value from the pAlarmValues list after the pBitmask is applied)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the object being tested),  
        'Time Stamp' = (any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
        'Event Type' = CHANGE\_OF\_BITSTRING,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = NORMAL,  
        'To State' = OFFNORMAL,  
        'Event Values' = pMonitoredValue, pStatusFlags
5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = OFFNORMAL
8. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (the timestamp in step 4, \*, \*)
9. IF (pMonitoredValue is writable) THEN  
    WRITE (pMonitoredValue) = (a value that corresponds to a NORMAL state)  
ELSE  
    MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
10. WAIT (pTimeDelayNormal)
11. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the object being tested),  
        'Time Stamp' = (any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
        'Event Type' = CHANGE\_OF\_BITSTRING,



'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = OFFNORMAL,  
 'To State' = NORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags

12. TRANSMIT BACnet-SimpleACK-PDU
13. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
14. VERIFY pCurrentState = NORMAL
15. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 4, \*, the timestamp in step 11)

Notes to Tester: The time stamps indicated by "\*" in steps 8 and 15 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 4.

#### 8.4.2 CHANGE\_OF\_STATE Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_STATE event algorithm.

Test Concept: The object begins the test in a NORMAL state. The Present\_Value (referenced property) is changed to a value that is one of the values designated in List\_Of\_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present\_Value (referenced property) is then changed to a value corresponding to a NORMAL state. After the time delay the object should enter the NORMAL state and transmit an event notification message. If the IUT claims conformance to Protocol\_Revision 12 or lower, and a Multi-state Input or Multi-state Value object is being tested, the transition to and from the FAULT state is also tested.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

If the IUT claims conformance to Protocol\_Revision 12 or lower, and supports intrinsic reporting for Multi-state Input or Multi-state Value objects, the intrinsic reporting object shall be configured with at least one of the two properties, Alarm\_Values (referred to as pAlarmValues in the test steps) and Fault\_Values (referred to as pFaultValues in the test steps), containing at least one value.

If the IUT claims conformance to Protocol\_Revision 12 or lower, and supports intrinsic reporting for Binary Input or Binary Value objects, the intrinsic reporting object shall be configured with the Alarm\_Value property (referred to as pAlarmValues in the test steps) containing at least one value.

If the IUT claims conformance to Protocol\_Revision 12 or lower, and supports algorithmic change reporting with an Event\_Type of CHANGE\_OF\_STATE, the List\_Of\_Values parameter of the Event\_Parameters property (referred to as pAlarmValues in the test steps) shall contain at least one value.

If the IUT claims conformance to Protocol\_Revision 13 or greater, and supports the CHANGE\_OF\_STATE algorithm, the IUT shall be configured with at least one value for pAlarmValues.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF ((Protocol\_Revision is present AND Protocol\_Revision >= 13)  
     OR ((Protocol\_Revision is present AND Protocol\_Revision < 13)  
     AND (pAlarmValues contains at least one value))) THEN {  
     IF (pMonitoredValue is writable) THEN  
         WRITE pMonitoredValue = (a value from pAlarmValues)  
     ELSE  
         MAKE (pMonitoredValue have a value pAlarmValues)  
     WAIT (pTimeDelay)  
     BEFORE **Notification Fail Time**  
         RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' = (any valid process ID),  
         'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
 'Time Stamp' = (T1, any valid time stamp),  
 'Notification Class' = (the configured notification class),  
 'Priority' = (the value configured to correspond to a TO\_OFFNORMAL transition),  
 'Event Type' = CHANGE\_OF\_STATE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags  
 TRANSMIT BACnet-SimpleACK-PDU  
 IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)  
 VERIFY pCurrentState = OFFNORMAL  
 IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (T1, Ta, Tb)  
 IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)  
 ELSE  
     MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)  
 WAIT (pTimeDelay)  
 BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' = (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
         'Time Stamp' = (T2, any valid time stamp),  
         'Notification Class' = (the configured notification class),  
         'Priority' = (the value configured to correspond to a TO\_NORMAL transition),  
         'Event Type' = CHANGE\_OF\_STATE,  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = EVENT | ALARM,  
         'AckRequired' = TRUE | FALSE,  
         'From State' = OFFNORMAL,  
         'To State' = NORMAL,  
         'Event Values' = pMonitoredValue, pStatusFlags  
     TRANSMIT BACnet-SimpleACK-PDU  
     IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
         VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
     VERIFY pCurrentState = NORMAL  
     IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
         VERIFY Event\_Time\_Stamps = (T1, Ta, T2)  
     }  
 3. IF ((Protocol\_Revision is present AND Protocol\_Revision < 13)  
     AND (intrinsic reporting is being tested)  
     AND (the intrinsic reporting object is configured with pFaultValues containing at least one values)) THEN  
     {  
         IF (pMonitoredValue is writable) THEN  
             WRITE pMonitoredValue = (a value from pFaultValues)  
         ELSE  
             MAKE (pMonitoredValue have a value from pFaultValues)  
         BEFORE **Notification Fail Time**  
             RECEIVE ConfirmedEventNotification-Request,  
                 'Process Identifier' = (any valid process ID),  
                 'Initiating Device Identifier' = IUT,

```

'Event Object Identifier' = (the intrinsic reporting object being tested),
'Time Stamp' = (Tfault: any valid timestamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO_FAULT transition),
'Event Type' = CHANGE_OF_STATE,
'Message Text' = (optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = NORMAL,
'To State' = FAULT,
'Event Values' = pMonitoredValue, pStatusFlags
TRANSMIT BACnet-SimpleACK-PDU
VERIFY pCurrentState = FAULT
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)
VERIFY pCurrentReliability = MULTI_STATE_FAULT
IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
ELSE
 MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested),
 'Time Stamp' = (Tnormal: any valid timestamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO_NORMAL transition),
 'Event Type' = CHANGE_OF_STATE,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = FAULT,
 'To State' = NORMAL,
 'Event Values' = pMonitoredValue, pStatusFlags
TRANSMIT BACnet-SimpleACK-PDU
VERIFY pCurrentState = NORMAL
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (Toffnormal, Tfault, Tnormal)
}

```

Notes to Tester: The time stamps indicated by "Ta" and "Tb" can have a value that indicates an unspecified time or a time that precedes the timestamp T1.

#### 8.4.3 CHANGE\_OF\_VALUE Tests (ConfirmedEventNotification)

This clause defines the tests necessary to demonstrate support for the CHANGE\_OF\_VALUE event algorithm. The CHANGE\_OF\_VALUE algorithm can be applied to both numerical and bitstring datatypes. The IUT shall pass the tests for both applications.

##### 8.4.3.1 Numerical Algorithm (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_VALUE event algorithm as applied to REAL datatypes.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed by a value that is less than pIncrement. The tester verifies that no event notification is transmitted. pMonitoredValue is changed again to a value that differs from the original value by an amount greater than pIncrement. The tester verifies that an event notification message is transmitted and that the proper event state transitions occur.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-NORMAL transition. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN  
WRITE (pMonitoredValue) = (a value that differs from the initial value by less than pIncrement)  
ELSE  
MAKE (pMonitoredValue have a value that differs from the initial value by less than pIncrement)
3. WAIT (pTimeDelayNormal + **Notification Fail Time**)
4. CHECK (verify that no event notification message is transmitted)
5. IF (the pMonitoredValue is writable) THEN  
WRITE (pMonitoredValue) = (a value that differs from the initial value in step 1 by more than pIncrement)  
ELSE  
MAKE (pMonitoredValue have a value that differs from the initial value in step 1 by more than pIncrement)
6. WAIT (pTimeDelayNormal)
7. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = (the object being tested),  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the configured notification class),  
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
    'Event Type' = CHANGE\_OF\_VALUE,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = EVENT | ALARM,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = NORMAL,  
    'Event Values' = pMonitoredValue, pStatusFlags
8. TRANSMIT BACnet-SimpleACK-PDU
9. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
10. VERIFY pCurrentState = NORMAL
11. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (\*, \*, the timestamp in step 7)

Notes to Tester: The time stamps indicated by "\*" in step 11 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 7.

#### 8.4.3.2 Bitstring Algorithm (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_VALUE event algorithm as applied to Bitstring datatypes.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed to a new value such that none of the bits in pBitmask are changed. The tester verifies that no event notification is transmitted. pMonitoredValue is changed again to a value that differs in one or more bits that are included in pBitmask. The tester verifies that an event notification message is transmitted and that the proper event state transitions occur.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO\_NORMAL transition. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. pBitmask shall be configured so that at least one but not all bits of pMonitoredValue are included in the mask. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (the pMonitoredValue is writable) THEN  
WRITE (pMonitoredValue) = (a value that differs from the initial value but only in bits that are not included in pBitmask)

ELSE  
 MAKE (the pMonitoredValue have a value that differs from the initial value but only in bits that are not included in pBitmask)  
 3. WAIT (pTimeDelayNormal + **Notification Fail Time**)  
 4. CHECK (verify that no event notification message is transmitted)  
 5. IF (the pMonitoredValue is writable) THEN  
     WRITE (pMonitoredValue) = (a value that differs from the initial value in one or more bits included in pBitmask)  
 ELSE  
     MAKE (the pMonitoredValue have a value that differs from the initial value one or more bits included in pBitmask)  
 6. WAIT (pTimeDelayNormal)  
 7. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = CHANGE\_OF\_VALUE,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = NORMAL,  
     'Event Values' = pMonitoredValue, pStatusFlags  
 8. TRANSMIT BACnet-SimpleACK-PDU  
 9. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
 10. VERIFY pCurrentState = NORMAL  
 11. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (\*, \*, the timestamp in step 7)

Notes to Tester: The time stamps indicated by "\*" in step 11 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 7.

#### 8.4.4 COMMAND\_FAILURE Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the COMMAND\_FAILURE algorithm.

Test Concept: pFeedbackValue shall be decoupled from the input signal that is normally used to verify the output. Initially pMonitoredValue and pFeedbackValue are in agreement. pMonitoredValue is changed and an event notification should be transmitted indicating a transition to an OFFNORMAL state. pFeedbackValue is changed to again agree with pMonitoredValue. A second event notification is transmitted indicating a return to a NORMAL state.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. pFeedbackValue shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

Notes to Tester: The time stamps indicated by "\*" in steps 9 and 16 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 5.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
3. IF (pMonitoredValue is writable) THEN  
     WRITE Present\_Value = (a different value)

- ELSE  
MAKE (Present\_Value take on a different value)
4. WAIT (pTimeDelay)
  5. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = (the object being tested),  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the configured notification class),  
    'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
    'Event Type' = COMMAND\_FAILURE,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = EVENT | ALARM,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = OFFNORMAL,  
    'Event Values' = pMonitoredValue, pStatusFlags, pFeedbackValue
  6. TRANSMIT BACnet-SimpleACK-PDU
  7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
  8. VERIFY pCurrentState = OFFNORMAL
  9. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (the timestamp in step 5, \*, \*)
  10. IF (pFeedbackValue is writable) THEN  
    WRITE pFeedbackValue = (a value consistent with pMonitoredValue)
  - ELSE  
    MAKE (pFeedbackValue take on a value consistent with pMonitoredValue)
  11. WAIT (pTimeDelay)
  12. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = (the object being tested),  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the configured notification class),  
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
    'Event Type' = COMMAND\_FAILURE,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = EVENT | ALARM,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = OFFNORMAL,  
    'To State' = NORMAL,  
    'Event Values' = pMonitoredValue, pStatusFlags, pFeedbackValue
  13. TRANSMIT BACnet-SimpleACK-PDU
  14. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
  15. VERIFY pCurrentState = NORMAL
  16. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (the timestamp in step 5, \*, the timestamp in step 12)

#### 8.4.5 FLOATING\_LIMIT Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the Floating Limit event algorithm.

Test Concept: The object begins the test in a NORMAL state. The referenced property is raised to a value that is below but within pDeadband of pHighDiffLimit. At this point the object should still be in a NORMAL state. pMonitoredValue is raised to a value that is above pHighDiffLimit. After the pTimeDelay expires the object should enter the HIGH\_LIMIT state and transmit an event notification message. pMonitoredValue is lowered to a value that is below pHighDiffLimit but still within pDeadband of pHighDiffLimit. The object should remain in the HIGH\_LIMIT state. pMonitoredValue is lowered further to a normal value that is not within pDeadband of pHighDiffLimit. After pTimeDelayNormal expires the



object should enter the NORMAL state and issue an event notification. The same process is repeated to test pLowDiffLimit.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < (pSetpoint + pHighDiffLimit))  
   ELSE  
     MAKE (pMonitoredValue have a value x:  
       (pSetpoint + pHighDiffLimit – pDeadband) < x < (pSetpoint + pHighDiffLimit))
3. WAIT (pTimeDelay + **Notification Fail Time**)
4. CHECK (verify that no notification message has been transmitted)
5. VERIFY pCurrentState = NORMAL
6. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: x > (pSetpoint + pHighDiffLimit))  
   ELSE  
     MAKE (pMonitoredValue have a value x: x > (pSetpoint + pHighDiffLimit))
7. WAIT (pTimeDelay)
8. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
       'Process Identifier' = (any valid process ID),  
       'Initiating Device Identifier' = IUT,  
       'Event Object Identifier' = (the object being tested),  
       'Time Stamp' = (any valid time stamp),  
       'Notification Class' = (the configured notification class),  
       'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
       'Event Type' = FLOATING\_LIMIT,  
       'Message Text' = (optional, any valid message text),  
       'Notify Type' = EVENT | ALARM,  
       'AckRequired' = TRUE | FALSE,  
       'From State' = NORMAL,  
       'To State' = HIGH\_LIMIT,  
       'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pHighDiffLimit
9. TRANSMIT BACnet-SimpleACK-PDU
10. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY pCurrentState = pHighDiffLimit
12. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 8, \*, \*)
13. IF (pMonitoredValue is writable) THEN  
     WRITE (pMonitoredValue) = (a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < pSetpoint + pHighDiffLimit))  
   ELSE  
     MAKE (pMonitoredValue have a value x: (pSetpoint + pHighDiffLimit – pDeadband) < x < pSetpoint + pHighDiffLimit))
14. WAIT (pTimeDelayNormal + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY pCurrentState = pHighDiffLimit
17. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))  
   ELSE  
     MAKE (pMonitoredValue have a value x:  
       (pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit – pDeadband))



18. WAIT (pTimeDelayNormal)
19. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = (the object being tested),  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the configured notification class),  
    'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
    'Event Type' = FLOATING\_LIMIT,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = EVENT | ALARM,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = HIGH\_LIMIT,  
    'To State' = NORMAL,  
    'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pHighDiffLimit,  
20. TRANSMIT BACnet-SimpleACK-PDU  
21. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
22. VERIFY pCurrentState = NORMAL  
23. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (the timestamp in step 8, \*, the timestamp in step 19)  
24. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit < x < (pSetpoint - pLowDiffLimit + pDeadband))  
    ELSE  
    MAKE (pMonitoredValue have a value x: (pSetpoint - pLowDiffLimit < x < (pSetpoint - pLowDiffLimit + pDeadband))  
25. WAIT (pTimeDelay + **Notification Fail Time**)  
26. CHECK (verify that no notification message has been transmitted)  
27. VERIFY pCurrentState = NORMAL  
28. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: x < (pSetpoint - pLowDiffLimit))  
    ELSE  
    MAKE (pMonitoredValue have a value x: x < (pSetpoint - pLowDiffLimit))  
29. WAIT (pTimeDelay)  
30. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process ID),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = (the object being tested),  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the configured notification class),  
    'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
    'Event Type' = FLOATING\_LIMIT,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = EVENT | ALARM,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = LOW\_LIMIT,  
    'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pLowDiffLimit  
31. TRANSMIT BACnet-SimpleACK-PDU  
32. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)  
33. VERIFY pCurrentState = LOW\_LIMIT  
34. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (the timestamp in step 30, \*, the timestamp in step 19)  
35. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (pSetpoint - pLowDiffLimit) < x < (pSetpoint - pLowDiffLimit + pDeadband))

ELSE  
 MAKE (pMonitoredValue have a value  $x$ :  $(pSetpoint - pLowDiffLimit) < x < (pSetpoint - pLowDiffLimit + pDeadband)$ )  
 36. WAIT (pTimeDelayNormal + **Notification Fail Time**)  
 37. CHECK (verify that no notification message has been transmitted)  
 38. VERIFY pCurrentState = pLowDiffLimit  
 39. IF (pMonitoredValue is writable) THEN  
   WRITE pMonitoredValue = (a value  $x$ :  $(pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit - pDeadband)$ )  
 ELSE  
   MAKE pMonitoredValue have a value  $x$ :  $(pSetpoint - pLowDiffLimit + pDeadband) < x < (pSetpoint + pHighDiffLimit - pDeadband)$   
 40. WAIT (pTimeDelayNormal)  
 41. BEFORE **Notification Fail Time**  
   RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = FLOATING\_LIMIT,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = LOW\_LIMIT,  
     'To State' = NORMAL,  
     'Event Values' = pMonitoredValue, pStatusFlags, pSetpoint, pLowDiffLimit  
 42. TRANSMIT BACnet-SimpleACK-PDU  
 43. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN  
   VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
 44. VERIFY pCurrentState = NORMAL  
 45. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
   VERIFY Event\_Time\_Stamps = (the timestamp in step 30, \*, the timestamp in step 41)

Notes to Tester: The time stamps indicated by "\*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

#### 8.4.6 OUT\_OF\_RANGE Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the OUT\_OF\_RANGE event algorithm.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is raised to a value that is below but within pDeadband of the pHighLimit. At this point the object should still be in a NORMAL state. pMonitoredValue is raised to a value that is above the pHighLimit. After pTimeDelay expires the object should enter the HIGH\_LIMIT state and transmit an event notification message. pMonitoredValue is lowered to a value that is below the pHighLimit but still within pDeadband of pHighLimit. The object should remain in the HIGH\_LIMIT state. pMonitoredValue is lowered further to a normal value that is not within pDeadband of pHighLimit. After pTimeDelayNormal expires the object should enter the NORMAL state and issue an event notification. The same process is repeated to test pLowLimit.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. For objects using intrinsic reporting the Limit\_Enable property shall have a value of TRUE for both HIGH\_LIMIT and LOW\_LIMIT events. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN  
   WRITE pMonitoredValue = (a value  $x$ :  $(pHighLimit - pDeadband) < x < pHighLimit$ )

- ELSE  
 MAKE (pMonitoredValue have a value x:  $(pHighLimit - pDeadband) < x < pHighLimit$ )
3. WAIT (pTimeDelay + **Notification Fail Time**)
  4. CHECK (verify that no notification message has been transmitted)
  5. VERIFY pCurrentState = NORMAL
  6. IF (pMonitoredValue is writable) THEN  
 WRITE pMonitoredValue = (a value x such  $x > pHighLimit$ )  
 ELSE  
 MAKE (pMonitoredValue have a value x:  $x > pHighLimit$ )
  7. WAIT (pTimeDelay)
  8. BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object being tested),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the configured notification class),  
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
 'Event Type' = OUT\_OF\_RANGE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = HIGH\_LIMIT,  
 'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pHighLimit
  9. TRANSMIT BACnet-SimpleACK-PDU
  10. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq 13$ ) THEN  
 VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
  11. VERIFY pCurrentState = HIGH\_LIMIT
  12. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq 1$ ) THEN  
 VERIFY Event\_Time\_Stamps = (the timestamp in step 8, \*, \*)
  13. IF (pMonitoredValue is writable) THEN  
 WRITE pMonitoredValue = (a value x:  $(pHighLimit - pDeadband) < x < pHighLimit$ )  
 ELSE  
 MAKE (pMonitoredValue have a value x:  $(pHighLimit - pDeadband) < x < pHighLimit$ )
  14. WAIT (pTimeDelayNormal + **Notification Fail Time**)
  15. CHECK (verify that no notification message has been transmitted)
  16. VERIFY pCurrentState = HIGH\_LIMIT
  17. IF (pMonitoredValue is writable) THEN  
 WRITE pMonitoredValue = (a value x:  $(pLowDiffLimit + pDeadband) < x < (pHighLimit - pDeadband)$ )  
 ELSE  
 MAKE (pMonitoredValue have a value x:  $(pLowDiffLimit + pDeadband) < x < (pHighLimit - pDeadband)$ )
  18. WAIT (pTimeDelayNormal)
  19. BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object being tested),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the configured notification class),  
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
 'Event Type' = OUT\_OF\_RANGE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = HIGH\_LIMIT,  
 'To State' = NORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pHighLimit
  20. TRANSMIT BACnet-SimpleACK-PDU

21. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
22. VERIFY pCurrentState = NORMAL
23. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 8, \*, the timestamp in step 19)
24. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))  
   ELSE  
     MAKE (pMonitoredValue have a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
25. WAIT (pTimeDelay + **Notification Fail Time**)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY pCurrentState = NORMAL
28. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x such x < pLowDiffLimit)  
   ELSE  
     MAKE (pMonitoredValue have a value x: x < pLowDiffLimit)
29. WAIT (pTimeDelay)
30. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' = (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object being tested),  
         'Time Stamp' = (any valid time stamp),  
         'Notification Class' = (the configured notification class),  
         'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
         'Event Type' = OUT\_OF\_RANGE,  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = EVENT | ALARM,  
         'AckRequired' = TRUE | FALSE,  
         'From State' = NORMAL,  
         'To State' = LOW\_LIMIT,  
         'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pLowDiffLimit
31. TRANSMIT BACnet-SimpleACK-PDU
32. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN  
     VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
33. VERIFY pCurrentState = LOW\_LIMIT
34. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 30, \*, the timestamp in step 19)
35. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))  
   ELSE  
     MAKE (pMonitoredValue have a value x: pLowDiffLimit < x < (pLowDiffLimit + pDeadband))
36. WAIT (pTimeDelayNormal + **Notification Fail Time**)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY pCurrentState = LOW\_LIMIT
39. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))  
   ELSE  
     MAKE (pMonitoredValue have a value x: (pLowDiffLimit + pDeadband) < x < (pHighLimit – pDeadband))
40. WAIT (pTimeDelayNormal)
41. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' = (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object being tested),  
         'Time Stamp' = (any valid time stamp),  
         'Notification Class' = (the configured notification class),  
         'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
         'Event Type' = OUT\_OF\_RANGE,  
         'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = LOW\_LIMIT,  
 'To State' = NORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags, pDeadband, pLowDiffLimit

42. TRANSMIT BACnet-SimpleACK-PDU

43. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)

44. VERIFY pCurrentState = NORMAL

45. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (the timestamp in step 30, \*, the timestamp in step 41)

Notes to Tester: The time stamps indicated by "\*" in steps 12, 23, 34 and 45 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 8.

#### 8.4.7 BUFFER\_READY Tests (ConfirmedEventNotification)

Purpose: To verify the correct operation of the BUFFER\_READY event algorithm.

Test Concept: The object that performs the notification ("the event initiating object") begins the test in a NORMAL state. The object containing the buffer ("the logging object") acquires enough records to generate a notification, at which time the notifying object performs a TO-NORMAL transition and sends a BUFFER\_READY notification.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-NORMAL transition. The event initiating object shall be in a NORMAL state at the start of the test. The 'Issue Confirmed Notifications' parameter of the element of the Notification Class' Recipient\_List property referring to the TD shall be set to TRUE.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (logging object collect enough records to generate a notification)
3. RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = BUFFER\_READY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = NORMAL,  
     'Event Values' = (pLogBuffer),  
                     (pPreviousCount: any valid value),  
                     (pMonitoredValue: any valid value CN1)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (logging object collect the number of records specified by pThreshold)
6. RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = BUFFER\_READY,  
     'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = NORMAL,  
 'Event Values' = (pLogBuffer),  
                     (pPreviousCount: CN1),  
                     (pMonitoredValue: CN1 + pThreshold)

#### 7. TRANSMIT BACnet-SimpleACK-PDU

### 8.4.8 CHANGE\_OF\_LIFE\_SAFETY Tests (ConfirmedEventNotification)

#### 8.4.8.1 NORMAL to OFFNORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning between the NORMAL and OFFNORMAL event states.

Test Concept: The object begins the test in a NORMAL state. The pMonitoredValue parameter is changed to one of the values designated in pAlarm\_Values. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue have a value from the pAlarmValue list)
4. WAIT pTimeDelay
5. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
     'Time Stamp' = (T1: any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
     'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
     'Message Text' = (S1: optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = OFFNORMAL,  
     'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
8. VERIFY pCurrentState = OFFNORMAL
9. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (T1, \*, \*)
10. IF (Event\_Message\_Texts property exists) THEN  
     VERIFY Event\_Message\_Texts = (S1, \*, \*)

#### 8.4.8.2 OFFNORMAL to NORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to NORMAL event states.

Test Concept: The object begins the test in an OFFNORMAL state. pMonitoredValue is made to be in the NORMAL state. If latching is supported, the object should remain in the OFFNORMAL state until object is reset. This test needs to



be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the NORMAL state and transmit an event notification message. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMonitoredValue have a value of NORMAL)
3. WAIT pTimeDelay
4. IF (latching is supported) THEN {
  - CHECK (pCurrentState = OFFNORMAL)
  - MAKE (the object reset)
  - BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the configured notification class),
    - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
    - 'Message Text' = (S1: optional, any valid message text),
    - 'Notify Type' = EVENT | ALARM,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = OFFNORMAL,
    - 'To State' = NORMAL,
    - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
  - TRANSMIT BACnet-SimpleACK-PDU.
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY Status\_Flags = (FALSE, FALSE, ?, ?),
  - VERIFY pCurrentState = NORMAL,
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
    - VERIFY Event\_Time\_Stamps = (\*, \*, T1)
  - IF (Event\_Message\_Texts property exists) THEN
    - VERIFY Event\_Message\_Texts = (\*, \*, S1)
- }
  - ELSE {
    - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedEventNotification-Request,
      - 'Process Identifier' = (any valid process ID),
      - 'Initiating Device Identifier' = IUT,
      - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
      - 'Time Stamp' = (T2: any valid time stamp),
      - 'Notification Class' = (the configured notification class),
      - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
      - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
      - 'Message Text' = (S2: optional, any valid message text),
      - 'Notify Type' = EVENT | ALARM,
      - 'AckRequired' = TRUE | FALSE,
      - 'From State' = OFFNORMAL,
      - 'To State' = NORMAL,
      - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
    - TRANSMIT BACnet-SimpleACK-PDU
    - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN



```

 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
 VERIFY pCurrentState = NORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (*, *, T2)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (*, *, S2)
 }

```

#### 8.4.8.3 NORMAL to LIFE\_SAFETY\_ALARM Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to LIFE\_SAFETY\_ALARM event states.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed to one of the values designated in pLifeSafetyAlarmValues. After the time delay expires the object should enter the LIFE\_SAFETY\_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMonitoredValue have a value from the pLifeSafetyAlarmValues)
4. WAIT pTimeDelay
4. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process ID),
  - 'Initiating Device Identifier' = IUT,
  - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
  - 'Time Stamp' = (T1: any valid time stamp),
  - 'Notification Class' = (the configured notification class),
  - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
  - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
  - 'Message Text' = (S1: optional, any valid message text),
  - 'Notify Type' = EVENT | ALARM,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = NORMAL,
  - 'To State' = LIFE\_SAFETY\_ALARM,
  - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
  - VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
8. VERIFY pCurrentState = LIFE\_SAFETY\_ALARM
9. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
  - VERIFY Event\_Time\_Stamps = (T1, \*, \*)
10. IF (Event\_Message\_Texts property exists) THEN
  - VERIFY Event\_Message\_Texts = (S1, \*, \*)

#### 8.4.8.4 LIFE\_SAFETY\_ALARM to NORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to NORMAL event states.

Test Concept: The object begins the test in a LIFE\_SAFETY\_ALARM state. pMonitoredValue is made to be in the NORMAL state. If latching is supported, the object should remain in the LIFE\_SAFETY\_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a LIFE\_SAFETY\_ALARM state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
- 3 . WAIT pTimeDelay
4. IF (latching is supported) THEN {
 

CHECK (pCurrentState = LIFE\_SAFETY\_ALARM)  
 MAKE (the object reset)  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
   'Process Identifier' = (any valid process ID),  
   'Initiating Device Identifier' = IUT,  
   'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment  
 object  
   being tested),  
   'Time Stamp' = (T1: any valid time stamp),  
   'Notification Class' = (the configured notification class),  
   'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
   'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
   'Message Text' = (S1: optional, any valid message text),  
   'Notify Type' = EVENT | ALARM,  
   'AckRequired' = TRUE | FALSE,  
   'From State' = LIFE\_SAFETY\_ALARM,  
   'To State' = NORMAL,  
   'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected  
 TRANSMIT BACnet-SimpleACK-PDU  
 IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
   VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
 VERIFY pCurrentState = NORMAL  
 IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
   VERIFY Event\_Time\_Stamps = (\*, \*, T1)  
 IF (Event\_Message\_Texts property exists) THEN  
   VERIFY Event\_Message\_Texts = (\*, \*, S1)  
 }  
 ELSE {  
   BEFORE **Notification Fail Time**  
   RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment  
 object  
     being tested),  
     'Time Stamp' = (T2: any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
     'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
     'Message Text' = (S2: optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = LIFE\_SAFETY\_ALARM,  
     'To State' = NORMAL,  
     'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected  
   TRANSMIT BACnet-SimpleACK-PDU  
   IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
 }

```

VERIFY pCurrentState = NORMAL
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (*, *T2)
IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (*, *, S2)
}

```

#### 8.4.8.5 LIFE\_SAFETY\_ALARM to OFFNORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from the LIFE\_SAFETY\_ALARM to OFFNORMAL event states.

Test Concept: The object begins the test in a LIFE\_SAFETY\_ALARM state. pMonitoredValue is made to be in the OFFNORMAL state. If latching is supported, the object should remain in the LIFE\_SAFETY\_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a LIFE\_SAFETY\_ALARM state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMonitoredValue have a value that corresponds to an OFFNORMAL state)
3. WAIT pTimeDelay
4. IF (latching is supported) THEN {
  - CHECK (pCurrentState = LIFE\_SAFETY\_ALARM)
  - MAKE (the object reset)
  - BEFORE Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the configured notification class),
    - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
    - 'Message Text' = (S1: optional, any valid message text),
    - 'Notify Type' = EVENT | ALARM,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = LIFE\_SAFETY\_ALARM,
    - 'To State' = OFFNORMAL,
    - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
  - TRANSMIT BACnet-SimpleACK-PDU
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
  - VERIFY pCurrentState = OFFNORMAL
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
    - VERIFY Event\_Time\_Stamps = (T1, \*, \*)
  - IF (Event\_Message\_Texts property exists) THEN
    - VERIFY Event\_Message\_Texts = (S1, \*, \*)
- ELSE {
  - BEFORE Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),

```

'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
'Time Stamp' = (T2: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,
'Message Text' = (S2: optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = LIFE_SAFETY_ALARM,
'To State' = OFFNORMAL,
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected.
TRANSMIT BACnet-SimpleACK-PDU
IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
VERIFY pCurrentState = OFFNORMAL
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamp = (T2, *, *)
IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S2, *, *)
}

```

#### 8.4.8.6 OFFNORMAL to LIFE\_SAFETY\_ALARM Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to LIFE\_SAFETY\_ALARM event states.

Test Concept: The object begins the test in an OFFNORMAL state. pMonitoredValue is made to be in the LIFE\_SAFETY\_ALARM state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, then after the time delay expires, the object should enter the LIFE\_SAFETY\_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in an OFFNORMAL state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMonitoredValue have a value that corresponds to a LIFE\_SAFETY state)
3. WAIT pTimeDelay
4. IF (latching is supported) THEN {
  - CHECK (pCurrentState = OFFNORMAL)
  - MAKE (the object reset)
  - BEFORE Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
   
object
   
being tested),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the configured notification class),
    - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
    - 'Message Text' = (S1: optional, any valid message text),
    - 'Notify Type' = EVENT | ALARM,
    - 'AckRequired' = TRUE | FALSE,

```

 'From State' = OFFNORMAL,
 'To State' = LIFE_SAFETY_ALARM,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = LIFE_SAFETY_ALARM
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T1, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S1, *, *)
 }
 ELSE {
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
object
being tested),
 'Time Stamp' = (T2: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S2: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = LIFE_SAFETY_ALARM,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = LIFE_SAFETY_ALARM
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T2, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S2, *, *)
 }
}

```

#### 8.4.8.7 Mode Transition Tests when Event State is Maintained

Purpose: To verify the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning between the NORMAL, OFFNORMAL and LIFE\_SAFETY\_ALARM event states when a mode change occurs. Tests are conducted when a mode change occurs, but the event state does not change. Tests are also conducted when a mode change occurs simultaneously with an event state change. In this latter case, the test verifies that the notification is immediate rather than waiting for the time delay.

Test Concept: The object begins the test in a NORMAL state. The Mode is changed. After the time delay expires, the object should transmit an event notification message. This operation is tested in the OFFNORMAL and LIFE\_SAFETY\_ALARM states as well.

The test is then repeated by changing the Mode property and simultaneously selecting a pMonitoredValue designated in pAlarmValues. The object should immediately enter the OFFNORMAL state and transmit an event notification message. pMonitoredValue is then changed to a value corresponding to a NORMAL state, and the Mode is simultaneously written. The object should immediately enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. CHECK (pCurrentState = NORMAL)
3. MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
4. IF (IUT supports another pMode value which maintains the NORMAL state) THEN {
  - MAKE (pMode = different value that maintains pCurrentState as NORMAL)
  - WAIT (pTimeDelayNormal)
  - BEFORE Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
    - 'Time Stamp' = (T1: any valid time stamp),
    - 'Notification Class' = (the configured notification class),
    - 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
    - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
    - 'Message Text' = (S1: optional, any valid message text),
    - 'Notify Type' = EVENT | ALARM,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = NORMAL,
    - 'To State' = NORMAL,
    - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
  - TRANSMIT BACnet-SimpleACK-PDU
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
  - VERIFY pCurrentState = NORMAL
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
    - VERIFY Event\_Time\_Stamps = (\*, \*, T1)
  - IF (Event\_Message\_Texts property exists) THEN
    - VERIFY Event\_Message\_Texts = (\*, \*, S1)
5. MAKE (pMonitoredValue have a value that corresponds to an OFFNORMAL state)
6. VERIFY pCurrentState = OFFNORMAL
7. IF (IUT supports another pMode value which maintains the OFFNORMAL state) THEN {
  - MAKE (pMode = different value that maintains pCurrentState as OFFNORMAL)
  - WAIT (pTimeDelay)
  - BEFORE Notification Fail Time**
  - RECEIVE ConfirmedEventNotification-Request,
    - 'Process Identifier' = (any valid process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
    - 'Time Stamp' = (T2: any valid time stamp),
    - 'Notification Class' = (the configured notification class),
    - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
    - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
    - 'Message Text' = (S2: optional, any valid message text),
    - 'Notify Type' = EVENT | ALARM,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = OFFNORMAL,
    - 'To State' = OFFNORMAL,
    - 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
  - TRANSMIT BACnet-SimpleACK-PDU
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
  - VERIFY pCurrentState = OFFNORMAL
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
    - VERIFY Event\_Time\_Stamps = (T2, \*, \*)
  - IF (Event\_Message\_Texts property exists) THEN
    - VERIFY Event\_Message\_Texts = (S2, \*, \*)



```

}
8. MAKE (pMonitoredValue have a value that corresponds to a LIFE_SAFETY_ALARM state)
9. IF (IUT supports another pMode value which maintains the LIFE_SAFETY_ALARM state) THEN {
 MAKE (pMode = different value that maintains pCurrentState = LIFE_SAFETY_ALARM)
 WAIT (pTimeDelay)
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object or the Event Enrollment object being tested),
 'Time Stamp' = (T3: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S3: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamped = (T3, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S3, *, *)
 }
}

```

#### 8.4.8.8 NORMAL to OFFNORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to OFFNORMAL event states by changing the Mode.

Test Concept: The object begins the test in a NORMAL state. The Mode is changed to a value forcing the object into an OFFNORMAL state. The object should immediately enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMode a different value that forces the state to OFFNORMAL)
4. BEFORE Notification Fail Time
 

```

RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
 'Time Stamp' = (T1: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S1: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,

```



- 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
5. TRANSMIT BACnet-SimpleACK-PDU
  6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
  7. VERIFY pCurrentState = OFFNORMAL
  8. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (T1, \*, \*)
  9. IF (Event\_Message\_Texts property exists) THEN  
     VERIFY Event\_Message\_Texts = (S1, \*, \*)

#### 8.4.8.9 OFFNORMAL to NORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to NORMAL event states by changing the Mode.

Test Concept: The object begins the test in an OFFNORMAL state. The Mode is changed to a value forcing the object into a NORMAL state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the NORMAL state and transmit an event notification message

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMode a different value that forces the state to NORMAL)
3. IF (latching is supported) THEN {  
     CHECK (pCurrentState = OFFNORMAL)  
     MAKE (the object reset)  
     BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
         'Process Identifier' = (any valid process ID),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
         'Time Stamp' = (T1: any valid time stamp),  
         'Notification Class' = (the configured notification class),  
         'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
         'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
         'Message Text' = (S1: optional, any valid message text),  
         'Notify Type' = EVENT | ALARM,  
         'AckRequired' = TRUE | FALSE,  
         'From State' = OFFNORMAL,  
         'To State' = NORMAL,  
         'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected  
     TRANSMIT BACnet-SimpleACK-PDU  
     IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
         VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
     VERIFY pCurrentState = NORMAL  
     IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
         VERIFY Event\_Time\_Stamps = (\*, \*, T1)  
     IF (Event\_Message\_Texts property exists) THEN  
         VERIFY Event\_Message\_Texts = (\*, \*, S1)  
     }

```

ELSE {
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
 'Time Stamp' = (T2: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S2: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = NORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
 VERIFY pCurrentState = NORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamped = (*, *, T2)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (*, *, S2)
}

```

#### 8.4.8.10 NORMAL to LIFE\_SAFETY\_ALARM Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to LIFE\_SAFETY\_ALARM event states by changing the Mode

Test Concept: The object begins the test in a NORMAL state. The Mode is changed to a value forcing the object into a LIFE\_SAFETY\_ALARM state. The object should immediately enter the LIFE\_SAFETY\_ALARM state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT, and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. VERIFY pCurrentState = NORMAL
3. MAKE (pMode a different value that forces the state to LIFE\_SAFETY\_ALARM)
4. BEFORE Notification Fail Time
 

```

 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
 'Time Stamp' = (T1: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S1: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = LIFE_SAFETY_ALARM,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected

```

5. TRANSMIT BACnet-SimpleACK-PDU
6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = LIFE\_SAFETY\_ALARM
8. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (T1, \*, \*)
9. IF (Event\_Message\_Texts property exists) THEN  
    VERIFY Event\_Message\_Texts = (S1, \*, \*)

#### 8.4.8.11 LIFE\_SAFETY\_ALARM to NORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to NORMAL event states by changing the Mode property.

Test Concept: The object begins the test in a LIFE\_SAFETY\_ALARM state. The Mode property is changed to a value forcing the object into a NORMAL state. If latching is supported, the object should remain in the LIFE\_SAFETY\_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the NORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a LIFE\_SAFETY\_ALARM state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMode a different value that forces the state to NORMAL)
3. IF (latching is supported) THEN {  
    CHECK (pCurrentState = LIFE\_SAFETY\_ALARM)  
    MAKE (the object reset)  
    **BEFORE Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),  
        'Time Stamp' = (T1: any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),  
        'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,  
        'Message Text' = (S1: optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = LIFE\_SAFETY\_ALARM,  
        'To State' = NORMAL,  
        'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected  
    TRANSMIT BACnet-SimpleACK-PDU  
    IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
        VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)  
    VERIFY pCurrentState = NORMAL  
    IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
        VERIFY Event\_Time\_Stamps = (\*, \*, T1)  
    IF (Event\_Message\_Texts property exists) THEN  
        VERIFY Event\_Message\_Texts = (\*, \*, S1)  
    }  
ELSE {  
    **BEFORE Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,

```

'Process Identifier' = (any valid process ID),
'Initiating Device Identifier' = IUT,
'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
'Time Stamp' = (T2: any valid time stamp),
'Notification Class' = (the configured notification class),
'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' = CHANGE_OF_LIFE_SAFETY,
'Message Text' = (S2: optional, any valid message text),
'Notify Type' = EVENT | ALARM,
'AckRequired' = TRUE | FALSE,
'From State' = LIFE_SAFETY_ALARM,
'To State' = NORMAL,
'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
TRANSMIT BACnet-SimpleACK-PDU
IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
VERIFY pCurrentState = NORMAL
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (*, *, T2)
IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (*, *, S2)
}

```

#### 8.4.8.12 LIFE\_SAFETY\_ALARM to OFFNORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to OFFNORMAL event states.

Test Concept: The object begins the test in a LIFE\_SAFETY\_ALARM state. The Mode property is changed to a value forcing the object into an OFFNORMAL state. If latching is supported, the object should remain in the LIFE\_SAFETY\_ALARM state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a LIFE\_SAFETY\_ALARM state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMode a different value that forces the state to OFFNORMAL)
  - IF (latching is supported) THEN {
    - CHECK (pCurrentState = OFFNORMAL)
    - MAKE (the object reset)
    - BEFORE **Notification Fail Time**
      - RECEIVE ConfirmedEventNotification-Request,
        - 'Process Identifier' = (any valid process ID),
        - 'Initiating Device Identifier' = IUT,
        - 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object being tested),
        - 'Time Stamp' = (T1: any valid time stamp),
        - 'Notification Class' = (the configured notification class),
        - 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
        - 'Event Type' = CHANGE\_OF\_LIFE\_SAFETY,
        - 'Message Text' = (S1: optional, any valid message text),
        - 'Notify Type' = EVENT | ALARM,
        - 'AckRequired' = TRUE | FALSE,

```

 'From State' = LIFE_SAFETY_ALARM,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
TRANSMIT BACnet-SimpleACK-PDU
IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T1, *, *)
IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S1, *, *)
}
ELSE {
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
 'Time Stamp' = (T2: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S2: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = LIFE_SAFETY_ALARM,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T2, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S2, *, *)
 }
}

```

#### 8.4.8.13 OFFNORMAL to LIFE\_SAFETY\_ALARM Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to LIFE\_SAFETY\_ALARM event states by changing the Mode property.

Test Concept: The object begins the test in an OFFNORMAL state. The Mode is changed to a value forcing the object into a LIFE\_SAFETY\_ALARM state. If latching is supported, the object should remain in the OFFNORMAL state until the LifeSafetyOperation service has been executed to reset the object. This test needs to be repeated for all reset operations supported. If latching is not supported, the object should immediately enter the LIFE\_SAFETY\_ALARM state and transmit an event notification message

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. The event-generating objects shall be in a LIFE\_SAFETY\_ALARM state at the start of the test. It is expected that the TD will often be able to reset the object by using the LifeSafetyOperation service with a reset or reset-alarm request.

Test Steps:

1. VERIFY Event\_Detection\_Enable = TRUE
2. MAKE (pMode a different value that forces the state to LIFE\_SAFETY\_ALARM)

```

3. IF (latching is supported) THEN
 CHECK (pCurrentState = OFFNORMAL)
 MAKE (the object reset)
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
 'Time Stamp' = (T1: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S1: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = LIFE_SAFETY_ALARM,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T1, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S1, *, *)
 ELSE {
 BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment
 object being tested),
 'Time Stamp' = (T2: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_LIFE_SAFETY,
 'Message Text' = (S2: optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = LIFE_SAFETY_ALARM,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
 TRANSMIT BACnet-SimpleACK-PDU
 IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (T2, *, *)
 IF (Event_Message_Texts property exists) THEN
 VERIFY Event_Message_Texts = (S2, *, *)
 }

```

#### 8.4.9 EXTENDED Test (ConfirmedEventNotification)

Purpose: To verify the correct generation of EXTENDED event notifications.

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



Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for whichever transition shall be used for the test. The 'Issue\_Confirmed\_Notifications' parameter shall have a value of TRUE. D1 and D2 are vendor specific delays (either of them or both may be zero).

Notes to Tester: The time stamps indicated by "\*" can have any valid value.

Test Steps:

1. IF (the object generates TO-OFFNORMAL transitions) THEN {
  - READ CS1 = pCurrentState
  - MAKE (an OFFNORMAL condition exist)
  - WAIT D1
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedEventNotification-Request,
      - 'Process Identifier' = (any valid process ID),
      - 'Initiating Device Identifier' = IUT,
      - 'Event Object Identifier' = (the event generating object),
      - 'Time Stamp' = (TS1: the current local time),
      - 'Notification Class' = (the configured notification class),
      - 'Priority' = (the value configured for TO\_OFFNORMAL),
      - 'Event Type' = EXTENDED,
      - 'Message Text' = (optional, any valid message text),
      - 'Notify Type' = EVENT | ALARM,
      - 'AckRequired' = TRUE | FALSE,
      - 'From State' = CS1,
      - 'To State' = (CS2: any offnormal valid event state),
      - 'Event Values' = ((pVendorId: any valid vendor id), (pEventType: any valid event-type), (a list of 0 or more valid parameters as defined by the Vendor))
  - TRANSMIT BACnet-SimpleACK-PDU
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
  - VERIFY pCurrentState = CS2
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
    - VERIFY Event\_Time\_Stamps = (TS1, \*, \*)
- }
2. IF (the object generates TO\_NORMAL transitions) THEN {
  - READ CS2 = pCurrentState
  - MAKE (a NORMAL condition exist)
  - WAIT D2
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedEventNotification-Request,
      - 'Process Identifier' = (any valid process ID),
      - 'Initiating Device Identifier' = IUT,
      - 'Event Object Identifier' = (the intrinsic reporting object being tested),
      - 'Time Stamp' = (TS2: the current local time),
      - 'Notification Class' = (the configured notification class),
      - 'Priority' = (the value configured for TO-NORMAL),
      - 'Event Type' = EXTENDED,
      - 'Message Text' = (optional, any valid message text),
      - 'Notify Type' = EVENT | ALARM,
      - 'AckRequired' = TRUE | FALSE,
      - 'From State' = CS2,
      - 'To State' = NORMAL,
      - 'Event Values' = ((pVendorId: any valid vendor id), (pEventType: any valid event-type), (a list of 0 or more valid parameters as defined by the Vendor))
  - TRANSMIT BACnet-SimpleACK-PDU
  - IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN
    - VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)



```

 VERIFY pCurrentState = NORMAL
 IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (*, *, TS2)
 }

```

#### 8.4.10 DOUBLE\_OUT\_OF\_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the DOUBLE\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event\_Type is DOUBLE\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.4.11 SIGNED\_OUT\_OF\_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the SIGNED\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event\_Type is SIGNED\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.4.12 UNSIGNED\_OUT\_OF\_RANGE Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.4.6, except that the Event\_Type is UNSIGNED\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.4.13 CHANGE\_OF\_CHARACTERSTRING Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_CHARACTERSTRING event algorithm.

Test Concept: The object begins the test in a NORMAL state. pMonitoredValue is changed to a value that is one of the values designated in pAlarmValues. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. The pMonitoredValue is then changed to a different value in the pAlarmValues. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message. pMonitoredValue is then changed to a value corresponding to a NORMAL state. After the time delay, the object should enter the NORMAL state and transmit an event notification message. If the IUT claims conformance to Protocol\_Revision 12 or lower, the transition to and from the FAULT state is also tested.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT, and TO-NORMAL transitions. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. The property and Event\_Parameters element, respectively, represented by pAlarmValues shall be non-empty. The event-generating object shall be in a NORMAL state at the start of the test.

If the IUT claims conformance to Protocol\_Revision 12 or lower and supports intrinsic reporting for CharacterString Value objects, the intrinsic reporting object shall be configured with at least one of the two properties, Alarm\_Values (referred to as pAlarmValues in the test steps) and Fault\_Values (referred to as pFaultValues in the test steps), containing at least one characterstring.

If the IUT claims conformance to Protocol\_Revision 12 or lower and supports algorithmic change reporting with an Event\_Type of CHANGE\_OF\_CHARACTERSTRING, the List\_Of\_Alarm\_Values parameter of the Event\_Parameters property (referred to as pAlarmValues in the test steps) shall contain at least one characterstring.

If the IUT claims conformance to Protocol\_Revision 13 or greater and supports the CHANGE\_OF\_CHARACTERSTRING algorithm, the IUT shall be configured with at least one characterstring for pAlarmValues.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF ((Protocol\_Revision is present AND Protocol\_Revision >= 13)  
OR ((Protocol\_Revision is present AND Protocol\_Revision < 13)  
AND (pAlarmValues contains at least one characterstring))) THEN {
3. IF (pMonitoredValue is writable) THEN  
WRITE pMonitoredValue = (a value from pAlarmValues)

```

ELSE
 MAKE (pMonitoredValue have a value from pAlarmValues)
4. WAIT (pTimeDelay)
5. BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the object being tested),
 'Time Stamp' = (Toffnormal: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL transition),
 'Event Type' = CHANGE_OF_CHARACTERSTRING,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = NORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pStatusFlags, pAlarmValues(matching list element)
6. TRANSMIT BACnet-SimpleACK-PDU
7. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE,?,?)
8. VERIFY pCurrentState = OFFNORMAL
9. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (Toffnormal, *, *)
10. IF (pAlarmValues has more than 1 entry) THEN {
11. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value from pAlarmValues not used in prior steps)
 ELSE
 MAKE (pMonitoredValue have a value from pAlarmValues not used in prior steps)
12. WAIT (pTimeDelay)
13. BEFORE Notification Fail Time
 RECEIVE ConfirmedEventNotification-Request,
 'Process Identifier' = (any valid process ID),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = (the object being tested),
 'Time Stamp' = (Toffnormal: any valid time stamp),
 'Notification Class' = (the configured notification class),
 'Priority' = (the value configured to correspond to a TO-OFFNORMAL
transition),
 'Event Type' = CHANGE_OF_CHARACTERSTRING,
 'Message Text' = (optional, any valid message text),
 'Notify Type' = EVENT | ALARM,
 'AckRequired' = TRUE | FALSE,
 'From State' = OFFNORMAL,
 'To State' = OFFNORMAL,
 'Event Values' = pMonitoredValue, pStatusFlags, pAlarmValues(matching list
element)
14. TRANSMIT BACnet-SimpleACK-PDU
15. IF (Protocol_Revision is present AND Protocol_Revision >= 13) THEN
 VERIFY Status_Flags = (TRUE, FALSE,?,?)
16. VERIFY pCurrentState = OFFNORMAL
17. IF (Protocol_Revision is present AND Protocol_Revision >= 1) THEN
 VERIFY Event_Time_Stamps = (Toffnormal, *, *)
 }
18. IF (pMonitoredValue is writable) THEN
 WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)
 ELSE
 MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
19. WAIT (pTimeDelayNormal)

```

20. **BEFORE Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (Tnormal: any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
     'Event Type' = CHANGE\_OF\_CHARACTERSTRING,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = OFFNORMAL,  
     'To State' = NORMAL,  
     'Event Values' = pMonitoredValue, pStatusFlags, pAlarmValues(matching element)
21. TRANSMIT BACnet-SimpleACK-PDU
22. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
     VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
23. VERIFY pCurrentState = NORMAL
24. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (Toffnormal, \*, Tnormal)
- }
25. IF ((Protocol\_Revision is present AND Protocol\_Revision < 13)  
 AND (intrinsic reporting is being tested)  
 AND (the intrinsic reporting object is configured with pFaultValues containing at least one characterstring))  
 THEN {
26. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value from pFaultValues)  
 ELSE  
     MAKE (pMonitoredValue have a value from pFaultValues)
27. **BEFORE Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested),  
     'Time Stamp' = (Tfault: any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO-FAULT transition),  
     'Event Type' = CHANGE\_OF\_CHARACTERSTRING,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = pMonitoredValue, pStatusFlags, (any characterstring)
28. TRANSMIT BACnet-SimpleACK-PDU
29. VERIFY pCurrentState = FAULT
30. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (Toffnormal, Tfault, Tnormal)
31. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
32. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value that corresponds to a NORMAL state)  
 ELSE  
     MAKE (pMonitoredValue have a value that corresponds to a NORMAL state)
33. **BEFORE Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the intrinsic reporting object being tested),  
     'Time Stamp' = (Tnormal: any valid time stamp),

'Notification Class' = (the configured notification class),  
 'Priority' = (the value configured to correspond to a TO-NORMAL transition),  
 'Event Type' = CHANGE\_OF\_CHARACTERSTRING,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = pMonitoredValue, pStatusFlags, any characterstring

34. TRANSMIT BACnet-SimpleACK-PDU  
 35. VERIFY pCurrentState = NORMAL  
 36. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     VERIFY Event\_Time\_Stamps = (Toffnormal, Tfault, Tnormal)  
 }

Notes to Tester: The time stamps indicated by "\*" in steps 9 and 17 can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification. pCurrentReliability refers to the Reliability property of the event-generating object for this test.

#### 8.4.14 UNSIGNED\_RANGE Test (ConfirmedEventNotification Test)

Purpose: To verify the correct operation of the UNSIGNED\_RANGE event algorithm.

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

Configuration Requirements: If possible, the IUT shall be configured such that the Event\_Enable property has a value of TRUE for the TO\_OFFNORMAL and TO\_NORMAL transitions. If possible, pLimitEnable shall have a value of TRUE for both HighLimit and LowLimit events. The 'Issue Confirmed Notifications' parameter in the Recipient\_List of the configured Notification Class shall have a value of TRUE. The Recipient\_List of the configured Notification Class shall contain the TD, thus ensuring that notifications are emitted. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (x > pHighLimit))  
   ELSE  
     MAKE (pMonitoredValue have a value x: (x > pHighLimit))
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (Toffnormal: any valid time stamp),  
     'Notification Class' = (the configured notification class),  
     'Priority' = (the value configured to correspond to a TO\_OFFNORMAL transition),  
     'Event Type' = UNSIGNED\_RANGE,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = HIGH\_LIMIT,  
     'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
5. TRANSMIT BACnet-SimpleACK-PDU

6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13)) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = HIGH\_LIMIT
8. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (Toffnormal, \*, \*)
9. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (pLowLimit < x < pHighLimit))  
ELSE  
    MAKE (pMonitoredValue have a value x: (pLowLimit < x < pHighLimit))
10. WAIT (pTimeDelayNormal)
11. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the object being tested),  
        'Time Stamp' = (Tnormal: any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured to correspond to a TO\_NORMAL transition),  
        'Event Type' = UNSIGNED\_RANGE,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = HIGH\_LIMIT,  
        'To State' = NORMAL,  
        'Event Values' = pMonitoredValue, pStatusFlags, pHighLimit
12. TRANSMIT BACnet-SimpleACK-PDU
13. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13)) THEN  
    VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)
14. VERIFY pCurrentState = NORMAL
15. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (Toffnormal, \*, Tnormal)
16. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (x < pLowLimit))  
ELSE  
    MAKE (pMonitoredValue have a value x: (x < pLowLimit))
17. WAIT (pTimeDelay)
18. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the object being tested),  
        'Time Stamp' = (Tlowlimit: any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured to correspond to a TO\_OFFNORMAL transition),  
        'Event Type' = UNSIGNED\_RANGE,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = NORMAL,  
        'To State' = LOW\_LIMIT,  
        'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit
19. TRANSMIT BACnet-SimpleACK-PDU
20. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13)) THEN  
    VERIFY Status\_Flags = (TRUE, FALSE, ?, ?)
21. VERIFY pCurrentState = LOW\_LIMIT
22. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
    VERIFY Event\_Time\_Stamps = (Tlowlimit, \*, Tnormal)
23. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (Low\_Limit < x < High\_Limit))

ELSE

MAKE (pMonitoredValue have a value x: (Low\_Limit < x < High\_Limit))

24. WAIT (pTimeDelayNormal)

25. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the object being tested),

'Time Stamp' = (Tlowtonormal: any valid time stamp),

'Notification Class' = (the configured notification class),

'Priority' = (the value configured to correspond to a TO\_NORMAL transition),

'Event Type' = UNSIGNED\_RANGE,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = LOW\_LIMIT,

'To State' = NORMAL,

'Event Values' = pMonitoredValue, pStatusFlags, pLowLimit

26. TRANSMIT BACnet-SimpleACK-PDU

27. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13)) THEN

VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)

28. VERIFY pCurrentState = NORMAL

29. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN

VERIFY Event\_Time\_Stamps = (Tlowlimit, \*, Tlowtonormal)

Notes to Tester: The time stamps indicated by "\*" can have a value that indicates an unspecified time or a time that precedes the timestamp of the first received notification.

#### 8.4.15 CHANGE\_OF\_STATUS\_FLAGS Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_STATUS\_FLAGS event algorithm.

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

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

Test Steps:

1. VERIFY pCurrentState = NORMAL

2. MAKE (pMonitoredValue AND pSelectedFlags <> {FALSE, FALSE, FALSE, FALSE})

3. WAIT (pTimeDelay)

4. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured for the transition),

'Event Type' = CHANGE\_OF\_STATUS\_FLAGS,

'Message Text' = (optional, any valid message text),

'Notify Type' = EVENT | ALARM,

'AckRequired' = TRUE | FALSE,

'From State' = NORMAL,



- 'To State' = OFFNORMAL,  
'Event Values' = (optional, pPresentValue), pMonitoredValue
5. TRANSMIT BACnet-SimpleACK-PDU
  6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = {TRUE, FALSE, ?, ?}
  7. VERIFY pCurrentState = OFFNORMAL
  8. MAKE (pMonitoredValue AND pSelectedFlags = {FALSE, FALSE, FALSE, FALSE})
  9. WAIT (pTimeDelayNormal)
  10. **BEFORE Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = O1  
        'Time Stamp' = (any valid time stamp),  
        'Notification Class' = (the notification class configured for O1),  
        'Priority' = (the value configured for the transition),  
        'Event Type' = CHANGE\_OF\_STATUS\_FLAGS,  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = OFFNORMAL,  
        'To State' = NORMAL,  
        'Event Values' = (optional, pPresentValue), pMonitoredValue
  11. TRANSMIT BACnet-SimpleACK-PDU
  12. IF (Protocol\_Revision is present AND Protocol\_Revision >= 13) THEN  
    VERIFY Status\_Flags = {FALSE, FALSE, ?, ?}
  13. VERIFY pCurrentState = NORMAL

#### 8.4.16 Proprietary Algorithms Test (ConfirmedEventNotifications)

Purpose: This test verifies the correct generation of ConfirmedEventNotification messages conveying a notification of the complex form. This applies to any non-Event Enrollment object that uses a proprietary event algorithm.

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

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for whichever transition shall be used for the test. The 'Issue Confirmed Notifications' parameter shall have a value of TRUE. D1 is a vendor specific delay (may be zero). The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (a condition exist that will cause the event generating object to transition)
3. WAIT D1
4. **BEFORE Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the event generating object being tested),  
        'Time Stamp' = (any valid time stamp),  
        'Notification Class' = (the configured notification class),  
        'Priority' = (the value configured for this transition),  
        'Event Type' = (the proprietary event type specified by the vendor),  
        'Message Text' = (optional, any valid message text),  
        'Notify Type' = EVENT | ALARM,  
        'AckRequired' = TRUE | FALSE,  
        'From State' = NORMAL,  
        'To State' = (the state the object was made to transition to),



'Event Values' = (a list of BACnetPropertyValue reporting the set of property values as defined by the vendor)

## 5. TRANSMIT BACnet-SimpleACK-PDU

### 8.4.17 CHANGE\_OF\_RELIABILITY Tests (ConfirmedEventNotifications)

#### 8.4.17.1 CHANGE\_OF\_RELIABILITY with No Fault Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the NONE fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.1

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.2 CHANGE\_OF\_RELIABILITY with the FAULT\_CHARACTERSTRING Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_CHARACTERSTRING fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.2

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.3 CHANGE\_OF\_RELIABILITY with the FAULT\_EXTENDED Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_EXTENDED fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.3

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.3, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.3, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.3, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.4 CHANGE\_OF\_RELIABILITY with the FAULT\_LIFE\_SAFETY Algorithm (ConfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_LIFE\_SAFETY fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.4

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.4, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.4, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.4, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### **8.4.17.5 CHANGE\_OF\_RELIABILITY with the FAULT\_STATE Algorithm (ConfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_STATE fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.5

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.5, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.5, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.5, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### **8.4.17.6 CHANGE\_OF\_RELIABILITY with the FAULT\_STATUS\_FLAGS Algorithm (ConfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_STATUS\_FLAGS fault algorithm.

Test Concept: The test concept corresponds to 8.5.17.6

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.6, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.6, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.6, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### **8.4.17.7 Event Enrollment Fault Condition Precedence Tests**

##### **8.4.17.7.1 Internal Faults Take Precedence Over Monitored Object Faults**

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

Test Concept: The test concept corresponds to 8.5.17.7.1

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

##### **8.4.17.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms**

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

Test Concept: The test concept corresponds to 8.5.17.7.2

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.7.3 Internal Faults Take Precedence Over Fault Algorithms

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

Test Concept: The test concept corresponds to 8.5.17.7.3

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.7.3, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.7.3, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.7.3, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.8 CHANGE\_OF\_RELIABILITY of Event Enrollment Object, Monitored Object Fault (ConfirmedEventNotifications)

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

Test Concept: The test concept corresponds to 8.5.17.8

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.8, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.8, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.8, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.9 CHANGE\_OF\_RELIABILITY of Event Enrollment Object Fault (ConfirmedEventNotifications)

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

Test Concept: The test concept corresponds to 8.5.17.9

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.9, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.9, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.9, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

**8.4.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (ConfirmedEventNotifications)**

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

Test Concept: The test concept corresponds to 8.5.17.10

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.10, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.10, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.10, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

**8.4.17.11 CHANGE\_OF\_RELIABILITY with Internal Object Fault (ConfirmedEventNotifications)**

Purpose: To verify that fault conditions, unrelated to fault algorithms, are detected and reported.

Test Concept: The test concept corresponds to 8.5.17.11

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.11, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.11, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.11, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

**8.4.17.12 CHANGE\_OF\_RELIABILITY - FAULT\_LISTED Tests (ConfirmedEventNotification)****8.4.17.12.1 NORMAL to FAULT Transition (ConfirmedEventNotification)**

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from NORMAL to FAULT event states.

Test Concept: The test concept corresponds to 8.5.17.12.1.

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.12.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.12.1, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.12.1, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

**8.4.17.12.2 FAULT-to-FAULT transition (ConfirmedEventNotification)**

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from FAULT to FAULT event states.

Test Concept: The test concept corresponds to 8.5.17.12.2.

Configuration Requirements: The configuration requirements are identical to those in 8.5.17.12.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of TRUE.

Test Steps: The test steps for this test case are identical to the test steps in 8.5.17.12.2, except that the UnconfirmedEventNotification requests are ConfirmedEventNotification requests, and the TD acknowledges receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.5.17.12.2, except that the event notifications shall be conveyed using a ConfirmedEventNotification service request.

#### 8.4.17.13 CHANGE\_OF\_RELIABILITY with the FAULT\_OUT\_OF\_RANGE Algorithm (ConfirmedEventNotification)

Test Concept: Select a fault detecting object O1 which is configured to use the FAULT\_OUT\_OF\_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value less than pMinimumNormalValue  
     | a value greater than pMaximumNormalValue)  
 ELSE  
     MAKE (pMonitoredValue = a value less than pMinimumNormalValue  
     | a value greater than pMaximumNormalValue)
4. BEFORE **Notification Fail Time**,  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the class corresponding to the object O1 being tested),  
     'Priority' = (the value configured to correspond to a TO\_FAULT transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = EVENT | ALARM,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = (pCurrentReliability, Status\_Flags, (A list of valid values for  
     properties required to be reported for O1, and 0 or more other properties of O1))
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY pCurrentReliability = UNDER\_RANGE | OVER\_RANGE
7. VERIFY pCurrentState = FAULT
8. VERIFY Status\_Flags = (TRUE, TRUE,?,?)
9. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value greater than or equal to pMinimumNormalValue, and pMonitoredValue  
     is less than or equal to pMaximumNormalValue)  
 ELSE  
     MAKE (pMonitoredValue = a value, greater than or equal to pMinimumNormalValue, and pMonitoredValue  
     is less than or equal to pMaximumNormalValue)
10. BEFORE **Notification Fail Time**,  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the class corresponding to the object O1 being tested),  
     'Priority' = (the value configured to correspond to a TO\_NORMAL transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,

'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (pCurrentReliability, Status\_Flags, (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))

11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
13. VERIFY pCurrentState = NORMAL
14. VERIFY Status\_Flags = (FALSE, FALSE, ?, ?)

#### 8.4.18 CHANGE\_OF\_DISCRETE\_VALUE Test (ConfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE\_OF\_DISCRETE\_VALUE event algorithm. This test applies to Event Enrollment objects with an Event\_Type of CHANGE\_OF\_DISCRETE\_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelay (pTimeDelay is the value for Time\_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a ConfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event\_Enrollment, EE1 is configured with an Object\_Property\_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date, Time, BACnetObjectIdentifier, or BACnetDateTime. Event\_Enable is configured with a value of (T,T,T) and Event\_Algorithm\_Inhibit = FALSE. The Event\_Parameters are configured with an Event Algorithm of CHANGE\_OF\_DISCRETE\_VALUE and a value for Time\_Delay that is within the allowable range for the IUT. The configured notification class is configured to send confirmed notifications to the TD. EE1 shall have an Event\_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY pCurrentState = Normal
2. MAKE (the referenced property has a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE Notification Fail Time  
 RECEIVE ConfirmedEventNotification  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = EE1,  
     'Time Stamp' = (the current local datetime or time or sequence number),  
     'Notification Class' = (the notification class configured for EE1),  
     'Priority' = (the value configured to correspond to TO-NORMAL),  
     'Event Type' = CHANGE\_OF\_DISCRETE\_VALUE,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'Ack Required' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = NORMAL,  
     'Event Values' = (x, Status\_Flags of O1)
5. TRANSMIT BACnet-SimpleAck-PDU
6. VERIFY pCurrentState = Normal

#### 8.4.19 ACCESS\_EVENT Test (ConfirmedEventNotification)

Purpose: To verify the correct operation of the ACCESS\_EVENT event algorithm.

Test Concept: The object, O1, begins the test in a NORMAL state. An access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT. A second access event, of a type not listed in pAccessEvents, is made to occur, if such is supported by the IUT. It is verified that no notification is generated. A third access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT.



Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for TO-NORMAL transitions. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. pAccessEvents shall be configured with at least 1 access event type that can be made to occur. If possible, at least access event type that can be made to occur shall not be included in pAccessEvents.

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

Test Steps:

-- Cause an access-event to occur that should be reported

1. VERIFY O1, Event\_State = NORMAL
2. MAKE(an access event occur which is listed in pAccessEvents)
3. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the configured process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = (Tnormal1: the current local time),  
'Notification Class' = (the configured notification class),  
'Priority' = (the value configured for a TO-NORMAL transition),  
'Event Type' = ACCESS\_EVENT,  
'Notify Type' = EVENT | ALARM,  
'AckRequired' = TRUE | FALSE,  
'From State' = NORMAL,  
'To State' = NORMAL,  
'Event Values' = ( pMonitoredValue, pStatusFlags, pAccessEventTag,  
pAccessEventTime, pAccessCredential  
-- and optionally pAuthenticationFactor)
4. TRANSMIT BACnet-SimpleACK-PDU
5. VERIFY O1, Status\_Flags = (FALSE, FALSE,?,?)
6. VERIFY O1, Event\_State = NORMAL
7. VERIFY O1, Event\_Time\_Stamps = (\*, \*, Tnormal1)

-- Cause an access-event to occur that should not be reported

8. IF (the IUT can detect access events which are not in pAccessEvents) THEN {  
MAKE(an access event occur which is not listed in pAccessEvents)  
CHECK(no notification is generated)  
}  
9. VERIFY O1, Status\_Flags = (FALSE, FALSE,?,?)
10. VERIFY O1, Event\_State = NORMAL
11. VERIFY O1, Event\_Time\_Stamps = (\*, \*, Tnormal1)

-- Cause an access-event to occur that should be reported

12. MAKE(an access event occur which is listed in pAccessEvents, and if possible different from the firstaccess event)
13. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the configured process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = O1,  
'Time Stamp' = (Tnormal2: the current local time),  
'Notification Class' = (the configured notification class),  
'Priority' = (the value configured for a TO-NORMAL transition),  
'Event Type' = ACCESS\_EVENT,  
'Notify Type' = EVENT | ALARM,  
'AckRequired' = TRUE | FALSE,  
'From State' = NORMAL,  
'To State' = NORMAL,



'Event Values' = (pMonitoredValue, pStatusFlags, pAccessEventTag,  
pAccessEventTime, pAccessCredential  
-- and optionally pAuthenticationFactor)

14. TRANSMIT BACnet-SimpleACK-PDU
15. VERIFY O1, Status\_Flags = (FALSE, FALSE, ?, ?)
16. VERIFY O1, Event\_State = NORMAL
17. VERIFY O1, Event\_Time\_Stamped = (\*, \*, Tnormal2)

#### 8.4.20 CHANGE\_OF\_TIMER ConfirmedNotification Tests

##### 8.4.20.1 CHANGE\_OF\_TIMER ConfirmedEventNotification Test

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

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

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

Test Steps:

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

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

available,

pExpirationTime, if one is available)

11. TRANSMIT BACnet-SimpleACK-PDU
12. VERIFY Status\_Flags = {FALSE, FALSE, ?, ?}
13. VERIFY Event\_State = NORMAL

#### 8.4.20.2 CHANGE\_OF\_TIMER Offnormal-to-Offnormal ConfirmedEventNotification

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

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

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

Test Steps:

1. VERIFY Event\_State = NORMAL
2. MAKE (pMonitoredValue equal to any of the values contained in pAlarmValues)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**

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

available,

pExpirationTime, if one is available)

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

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

16. TRANSMIT BACnet-SimpleACK-PDU
17. VERIFY Status\_Flags = {FALSE, FALSE, ?, ?}
18. VERIFY Event\_State = NORMAL

### 8.5 UnconfirmedEventNotification Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating UnconfirmedEventNotification service requests. The UnconfirmedEventNotification tests are specific to the event detection algorithm used. For each object type that supports intrinsic event reporting the IUT shall pass the tests for the event detection algorithm that applies to that object type. If the IUT supports the Event Enrollment object it shall pass the tests for the event detection algorithm that corresponds to each event type supported.

For devices that implement a read-only Recipient\_List property for all instances of Notification Class objects, and are exclusively configured for UnconfirmedEventNotification service initiation, and the device can not contain a Notification Forwarder object, the Recipient\_List property is required to have a single entry with the recipient value corresponding to a local broadcast that is effective for all days, all times, and all transitions, and shall issue unconfirmed notifications using Process Identifier 0 exclusively.

#### 8.5.1 CHANGE\_OF\_BITSTRING Tests (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_BITSTRING event algorithm.

Test Concept: The test concept corresponds to 8.4.1.

Configuration Requirements: The configuration requirements are identical to those in 8.4.1, except that the Issue\_Confirmed\_Notifications property 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.1 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.1 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.2 CHANGE\_OF\_STATE Tests (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_STATE event algorithm.

Test Concept: The test concept corresponds to 8.4.2.

Configuration Requirements: The configuration requirements are identical to those in 8.4.2, except that the Issue\_Confirmed\_Notifications property 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.2 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.2 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.3 CHANGE\_OF\_VALUE Tests (UnconfirmedEventNotification)

This clause defines the tests necessary to demonstrate support for the CHANGE\_OF\_VALUE event algorithm. The CHANGE\_OF\_VALUE algorithm can be applied to both numerical and bitstring datatypes. The IUT shall pass the tests for both applications.

##### 8.5.3.1 Numerical Algorithm (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_VALUE event algorithm applied to REAL datatypes.

Test Concept: The test concept corresponds to 8.4.3.1.

Configuration Requirements: The configuration requirements are identical to those in 8.4.3.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.1 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.3.1 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

**8.5.3.2 Bitstring Algorithm (UnconfirmedEventNotification)**

Purpose: To verify the correct operation of the CHANGE\_OF\_STATE event algorithm applied to Bitstring datatypes.

Test Concept: The test concept corresponds to 8.4.3.2.

Configuration Requirements: The configuration requirements are identical to those in 8.4.3.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.2 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.3.2 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

**8.5.4 COMMAND\_FAILURE Tests (UnconfirmedEventNotification)**

Purpose: To verify the correct operation of the COMMAND\_FAILURE algorithm.

Test Concept: The test concept corresponds to 8.4.4

Configuration Requirements: The configuration requirements are identical to those in 8.4.4, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.4 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.4 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

**8.5.5 FLOATING\_LIMIT Tests (UnconfirmedEventNotification)**

Purpose: To verify the correct operation of the Floating Limit event algorithm.

Test Concept: The test concept corresponds to 8.4.5

Configuration Requirements: The configuration requirements are identical to those in 8.4.5, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.5 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.5 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

**8.5.6 OUT\_OF\_RANGE Tests (UnconfirmedEventNotification)**

Purpose: To verify the correct operation of the OUT\_OF\_RANGE event algorithm.

Test Concept: The test concept corresponds to 8.4.6

Configuration Requirements: The configuration requirements are identical to those in 8.4.6, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.6 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.6 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.7 BUFFER\_READY Tests (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the BUFFER\_READY event algorithm.

Test Concept: The test concept corresponds to 8.4.7

Configuration Requirements: The configuration requirements are identical to those in 8.4.7, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.7, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.7, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.8 CHANGE\_OF\_LIFE\_SAFETY TESTS (UnconfirmedEventNotification)

##### 8.5.8.1 NORMAL to OFFNORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to OFFNORMAL event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.1, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

##### 8.5.8.2 OFFNORMAL to NORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to NORMAL event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.2, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

##### 8.5.8.3 NORMAL to LIFE\_SAFETY\_ALARM Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to LIFE\_SAFETY\_ALARM event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.3, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

##### 8.5.8.4 LIFE\_SAFETY\_ALARM to NORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to NORMAL event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.4, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

##### 8.5.8.5 LIFE\_SAFETY\_ALARM to OFFNORMAL Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to OFFNORMAL event states.



The test concept, configuration requirements and test steps are identical to those in 8.4.8.5, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.6 OFFNORMAL to LIFE\_SAFETY\_ALARM Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to LIFE\_SAFETY\_ALARM event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.6, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.7 Mode Transition Tests when Event State is Maintained

Purpose: To verify the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning between NORMAL, OFFNORMAL and LIFE\_SAFETY\_ALARM event states when a mode change occurs. Tests are conducted when a mode change occurs, but the event state does not change. Tests are also conducted when a mode change occurs simultaneously with an event state change. In this latter case, the test verifies that the notification is immediate rather than waiting for the time delay.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.7, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.8 NORMAL to OFFNORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to OFFNORMAL event states by changing the Mode.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.8, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.9 OFFNORMAL to NORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to NORMAL event states by changing the Mode.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.9, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.10 NORMAL to LIFE\_SAFETY\_ALARM Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from NORMAL to LIFE\_SAFETY\_ALARM event states by changing the Mode.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.10, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.11 LIFE\_SAFETY\_ALARM to NORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to NORMAL event states by changing the Mode property.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.11, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.12 LIFE\_SAFETY\_ALARM to OFFNORMAL Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from LIFE\_SAFETY\_ALARM to OFFNORMAL event states.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.12, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.8.13 OFFNORMAL to LIFE\_SAFETY\_ALARM Mode Transition Test

Purpose: This test case verifies the correct operation of the CHANGE\_OF\_LIFE\_SAFETY event algorithm for objects transitioning from OFFNORMAL to LIFE\_SAFETY\_ALARM event states by changing the Mode property.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.13, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

#### 8.5.9 EXTENDED Test (UnconfirmedEventNotification)

Purpose: To verify the correct generation of EXTENDED event notifications.

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

Configuration Requirements: The configuration requirements are identical to those in 8.4.9, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.9, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.9, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.10 DOUBLE\_OUT\_OF\_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the DOUBLE\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event\_Type is DOUBLE\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.5.11 SIGNED\_OUT\_OF\_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the SIGNED\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event\_Type is SIGNED\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.5.12 UNSIGNED\_OUT\_OF\_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED\_OUT\_OF\_RANGE event algorithm.

Test Concept: This test is the same as 8.5.6, except that the Event\_Type is UNSIGNED\_OUT\_OF\_RANGE instead of OUT\_OF\_RANGE.

#### 8.5.13 CHANGE\_OF\_CHARACTERSTRING Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_CHARACTERSTRING event algorithm.

Test Concept: The test concept corresponds to 8.4.13.

Configuration Requirements: The configuration requirements are identical to those in 8.4.13, except that the 'Issue\_Confirmed\_Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.13, except that the event notification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.13, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.14 UNSIGNED\_RANGE Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the UNSIGNED\_RANGE event algorithm.

Test Concept: The test concept corresponds to 8.4.14.

Configuration Requirements: The configuration requirements are identical to those in 8.4.14, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.14 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.14 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.15 CHANGE\_OF\_STATUS\_FLAGS Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the CHANGE\_OF\_STATUS\_FLAGS event algorithm.

Test Concept: The test concept corresponds to 8.4.15.

Configuration Requirements: The configuration requirements are identical to those in 8.4.15, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.15 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.15 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.16 Proprietary Algorithm Tests (UnconfirmedEventNotifications)

Purpose: This test verifies the correct generation of UnconfirmedEventNotification messages conveying a notification of the complex form. This applies to any non-Event Enrollment object that uses a proprietary algorithm.

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

Configuration Requirements: The configuration requirements are identical to those in 8.4.16, except that the 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.16, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.16, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.17 CHANGE\_OF\_RELIABILITY Tests (UnconfirmedEventNotification)

The CHANGE\_OF\_RELIABILITY tests apply to devices that claim conformance to Protocol\_Revision 13 or greater only.

**8.5.17.1 CHANGE\_OF\_RELIABILITY with No Fault Algorithm (UnconfirmedEventNotifications)**

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

Test Concept: Select an object, O1, capable of generating fault using the NONE fault algorithm. Ensure that no other fault conditions exist for the object. Create a fault condition. Verify the transition to fault is generated with Reliability set to R1. Remove the fault condition and verify the object transitions out of fault.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present and the Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Notes to Tester: The mechanism to enter the no fault state is a local matter.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (O1 enter a fault condition)
4. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = NORMAL,
    - 'To State' = FAULT,
    - 'Event Values' = (R1 any valid BACnetReliability, (T, T, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (O1 clear the fault condition)
8. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request
    - 'Process Identifier' = (any valid process identifier),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = O1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for O1),
    - 'Priority' = (the value configured for the transition),
    - 'Event Type' = CHANGE\_OF\_RELIABILITY,
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM | EVENT,
    - 'AckRequired' = TRUE | FALSE,
    - 'From State' = FAULT,
    - 'To State' = NORMAL,
    - 'Event Values' = (NO\_FAULT\_DETECTED, (F, F, ?, ?), (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

### 8.5.17.2 CHANGE\_OF\_RELIABILITY with the FAULT\_CHARACTERSTRING Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_CHARACTERSTRING fault algorithm.

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

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

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

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = FV1  
ELSE  
    MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = NORMAL,  
    'To State' = FAULT,  
    'Event Values' = (MULTI\_STATE\_FAULT,  
    (T, T, ?, ?),  
    (A list of valid values for properties required to be reported  
    for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
6. VERIFY pCurrentState = FAULT
7. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NFV1  
ELSE  
    MAKE (pMonitoredValue = NFV1)
8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

#### 8.5.17.3 CHANGE\_OF\_RELIABILITY with the FAULT\_EXTENDED Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_EXTENDED fault algorithm.

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

Configuration Requirements: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present and has an Event\_State of NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (a fault condition exist)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = (any valid event state),  
 'To State' = FAULT,  
 'Event Values' = ((R1: any valid reliability value),  
 (T, T, ?, ?),  
 (a vendor specified set of values))
5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (remove the fault condition)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,



'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 (a vendor specified set of values))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

#### 8.5.17.4 CHANGE\_OF\_RELIABILITY with the FAULT\_LIFE\_SAFETY Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_LIFE\_SAFETY fault algorithm.

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

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present and has an Event\_State of NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue, which does not indicate a fault condition. . The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = FV1  
   ELSE  
     MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
   RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = (MULTI\_STATE\_FAULT,  
     (T, T, ?, ?),  
     (A list of valid values for properties required to be reported  
     for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
6. VERIFY pCurrentState = FAULT
7. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = NV1  
   ELSE  
     MAKE (pMonitoredValue = NV1)
8. BEFORE **Notification Fail Time**  
   RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

#### 8.5.17.5 CHANGE\_OF\_RELIABILITY with the FAULT\_STATE Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the FAULT\_STATE fault algorithm.

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

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and an Event\_State of NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = FV1  
 ELSE  
     MAKE (pMonitoredValue = FV1)
4. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = (any valid time stamp),  
     'Notification Class' = (the notification class configured for O1),  
     'Priority' = (the value configured for the transition),  
     'Event Type' = CHANGE\_OF\_RELIABILITY,  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ALARM | EVENT,  
     'AckRequired' = TRUE | FALSE,  
     'From State' = NORMAL,  
     'To State' = FAULT,  
     'Event Values' = (MULTI\_STATE\_FAULT,  
     (T, T, ?, ?),  
     (A list of valid values for properties required to be reported  
     for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MULTI\_STATE\_FAULT
6. VERIFY pCurrentState = FAULT
7. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = NV1

ELSE

MAKE (pMonitoredValue = NV1)

8. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured for the transition),

'Event Type' = CHANGE\_OF\_RELIABILITY,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,

'From State' = FAULT,

'To State' = NORMAL,

'Event Values' = (NO\_FAULT\_DETECTED,

(F, F, ?, ?),

(A list of valid values for properties required to be reported  
for O1, and 0 or more other properties of O1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

**8.5.17.6 CHANGE\_OF\_RELIABILITY with the FAULT\_STATUS\_FLAGS Algorithm  
(UnconfirmedEventNotifications)**

Purpose: To verify the correct operation of the FAULT\_STATUS\_FLAGS fault algorithm.

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

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. FV1 is a value for pMonitoredValue, which indicates a fault condition, and NV1 is a value for pMonitoredValue which does not indicate a fault condition. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

2. VERIFY pCurrentState = NORMAL

3. IF (pMonitoredValue is writable) THEN

WRITE pMonitoredValue = FV1

ELSE

MAKE (pMonitoredValue = FV1)

4. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured for the transition),

'Event Type' = CHANGE\_OF\_RELIABILITY,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,

'From State' = NORMAL,

'To State' = FAULT,

- 'Event Values' = (MEMBER\_FAULT,  
(T, T, ?, ?),  
(A list of valid values for properties required to be reported  
for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = MEMBER\_FAULT
  6. VERIFY pCurrentState = FAULT
  7. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
  8. **BEFORE Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request  
    'Process Identifier' = (any valid process identifier),  
    'Initiating Device Identifier' = IUT,  
    'Event Object Identifier' = O1,  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = FAULT,  
    'To State' = NORMAL,  
    'Event Values' = (NO\_FAULT\_DETECTED,  
    (F, F, ?, ?),  
    (A list of valid values for properties required to be reported  
    for O1, and 0 or more other properties of O1))
  9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
  10. VERIFY pCurrentState = NORMAL

#### 8.5.17.7 Event Enrollment Fault Condition Precedence Tests

##### 8.5.17.7.1 Internal Faults Take Precedence Over Monitored Object Faults

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

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

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition exist which will cause O1 to transition into fault)
4. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R1)
7. VERIFY pCurrentReliability = R1
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
10. MAKE(clear the condition that caused O1 to transition into fault)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

##### 8.5.17.7.2 Monitored Object Faults Take Precedence Over Fault Algorithms

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

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

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)
4. VERIFY pCurrentReliability = R1
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause O1 to transition into fault)
7. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
8. MAKE(clear the condition that caused O1 to transition into fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

#### 8.5.17.7.3 Internal Faults Take Precedence Over Fault Algorithms

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

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

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE(a condition that results in a fault due to a configured fault algorithm with a reliability value, R1)
4. VERIFY pCurrentReliability = R1
5. VERIFY pCurrentState = FAULT
6. MAKE(a condition exist which will cause EE1 to transition into internal fault R2, different from R1)
7. VERIFY pCurrentReliability = R2
8. MAKE(clear the condition that caused EE1 to enter into an internal fault)
9. VERIFY pCurrentReliability = R1
10. MAKE(clear the condition for the fault algorithm)
11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL

#### 8.5.17.8 CHANGE\_OF\_RELIABILITY of Event Enrollment Object, Monitored Object Fault (UnconfirmedEventNotifications)

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

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

Configuration Requirements: EE1 is configured to process faults in M1 and to report those. EE1 and M1 are each initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (M1 enter any fault state)
4. BEFORE **Notification Fail Time**  
RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = EE1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for EE1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = FAULT,  
 'Event Values' = (MONITORED\_OBJECT\_FAULT,  
 (T, T, ?, ?),  
 M1,  
 (optional, property value of M1),  
 (optional, M1 Status\_Flags, (?, T, ?, ?)),  
 (0 or more other properties of M1))

5. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT

6. VERIFY pCurrentState = FAULT

7. MAKE (M1 clear fault state)

8. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = EE1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for EE1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 M1,  
 (optional, property value of M1),  
 (optional, M1 Status\_Flags, (?, F, ?, ?)),  
 (0 or more other properties of M1))

9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

10. VERIFY pCurrentState = NORMAL

#### 8.5.17.9 CHANGE\_OF\_RELIABILITY of Event Enrollment Object Fault (UnconfirmedEventNotifications)

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

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

Configuration Requirements: EE1 is configured to be able to enter a fault state and to report those. EE1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

2. VERIFY pCurrentState = NORMAL



3. MAKE (EE1 enter any fault state)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
   'Process Identifier' = (any valid process identifier),  
   'Initiating Device Identifier' = IUT,  
   'Event Object Identifier' = EE1,  
   'Time Stamp' = (any valid time stamp),  
   'Notification Class' = (the notification class configured for EE1),  
   'Priority' = (the value configured for the transition),  
   'Event Type' = CHANGE\_OF\_RELIABILITY,  
   'Message Text' = (optional, any valid message text),  
   'Notify Type' = ALARM | EVENT,  
   'AckRequired' = TRUE | FALSE,  
   'From State' = NORMAL,  
   'To State' = FAULT,  
   'Event Values' = ((R1: any value other than  
     MONITORED\_OBJECT\_FAULT  
     and NO\_FAULT\_DETECTED),  
     (T, T, ?, ?),  
     (M1, any valid monitored object),  
     (optional, property value of M1),  
     (optional, M1 Status\_Flags, (?, F, ?, ?)),  
     (0 or more other properties of M1))
5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (EE1 clear fault state)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
   'Process Identifier' = (any valid process identifier),  
   'Initiating Device Identifier' = IUT,  
   'Event Object Identifier' = EE1,  
   'Time Stamp' = (any valid time stamp),  
   'Notification Class' = (the notification class configured for EE1),  
   'Priority' = (the value configured for the transition),  
   'Event Type' = CHANGE\_OF\_RELIABILITY,  
   'Message Text' = (optional, any valid message text),  
   'Notify Type' = ALARM | EVENT,  
   'AckRequired' = TRUE | FALSE,  
   'From State' = FAULT,  
   'To State' = NORMAL,  
   'Event Values' = (NO\_FAULT\_DETECTED,  
     (E, F, ?, ?),  
     M1,  
     (optional, property value of M1),  
     (optional, M1 Status\_Flags, (?, F, ?, ?)),  
     (0 or more other properties of M1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

#### 8.5.17.10 After FAULT-to-NORMAL, Re-Notification of OFFNORMAL (UnconfirmedEventNotifications)

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

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

Configuration Requirements: O1 is configured to detect and report faults. O1 is configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (O1 transition to an offnormal state)
4. WAIT pTimeDelay
5. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = (ET1, any valid off normal event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = (property-values appropriate for O1 )
6. VERIFY pCurrentState = OFFNORMAL
7. MAKE (O1 enter a fault state)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = OFFNORMAL,  
 'To State' = FAULT,  
 'Event Values' = ((R1, any valid BACnetReliability),  
 (? , T, ? , ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))
9. MAKE (O1 clear the fault condition)
10. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (TS1, any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ? , ?),  
 (A list of valid values for properties required to be reported

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

11. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
12. VERIFY pCurrentState = NORMAL
13. WAIT until TS1 + pTimeDelay
14. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = ET1,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = OFFNORMAL,  
 'Event Values' = (property-values appropriate for O1)
15. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
16. VERIFY pCurrentState = OFFNORMAL

#### 8.5.17.11 CHANGE\_OF\_RELIABILITY with Internal Object Fault (UnconfirmedEventNotifications)

Purpose: To verify that fault conditions, unrelated to fault algorithms, are detected and reported.

Test Concept: An object in the IUT, O1, which can detect at least one internal fault is selected. One of O1's detectable internal faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO\_FAULT\_DETECTED. The internal fault condition, R1, is made to exist, and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO\_FAULT\_DETECTED and the appropriate event notification message is generated.

Configuration Requirements: O1 is configured to detect and report faults. O1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = FAULT,  
 'Event Values' = (R1,  
 (? , T , ? , ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = R1

6. VERIFY pCurrentState = FAULT
7. MAKE (pCurrentReliability = NO\_FAULT\_DETECTED)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
   'Process Identifier' = (any valid process ID),  
   'Initiating Device Identifier' = IUT,  
   'Event Object Identifier' = O1,  
   'Time Stamp' = (any valid time stamp),  
   'Notification Class' = (the notification class configured for O1),  
   'Priority' = (the value configured for the transition),  
   'Event Type' = CHANGE\_OF\_RELIABILITY,  
   'Message Text' = (optional, any valid message text),  
   'Notify Type' = EVENT | ALARM,  
   'AckRequired' = TRUE | FALSE,  
   'From State' = FAULT,  
   'To State' = NORMAL,  
   'Event Values' = (NO\_FAULT\_DETECTED,  
                     (? , F , ? , ?),  
                     (A list of valid values for properties required to be reported  
                     for O1, and 0 or more other properties of O1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

#### 8.5.17.12 CHANGE\_OF\_RELIABILITY - FAULT\_LISTED Tests (UnconfirmedEventNotification)

##### 8.5.17.12.1 NORMAL to FAULT Transition (UnconfirmedEventNotification)

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from NORMAL to FAULT event states.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_LISTED algorithm. Ensure that no fault conditions exist in the object. Set pMonitoredList to FV1, a non-empty list of supported faults. Verify the correct transition is generated. The fault condition is removed by setting pMonitoredList to an empty list. Verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have no fault conditions present and has an Event\_State of NORMAL.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. IF (pMonitoredList is writable) THEN  
   WRITE pMonitoredList = FV1  
 ELSE  
   MAKE (pMonitoredList = FV1)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
   'Process Identifier' = (any valid process Identifier),  
   'Initiating Device Identifier' = IUT  
   'Event Object Identifier' = O1  
   'Time Stamp' = (any valid time stamp),  
   'Notification Class' = (the notification class configured for O1),  
   'Priority' = (the value configured for the transition),  
   'Event Type' = CHANGE\_OF\_RELIABILITY,  
   'Message Text' = (optional, any valid message text),  
   'Notify Type' = ALARM | EVENT,  
   'AckRequired' = TRUE | FALSE,  
   'From State' = NORMAL,  
   'To State' = FAULT,  
   'Event Values' = (FAULT\_LISTED,

- (T, T, ? ?),  
(A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
5. VERIFY pCurrentReliability = FAULTS\_LISTED
  6. VERIFY pCurrentState = FAULT
  7. IF (pMonitoredList is writable) THEN  
    WRITE pMonitoredList = (an empty list)  
ELSE  
    MAKE (pMonitoredList = (an empty list)
  8. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process Identifier),  
    'Initiating Device Identifier' = IUT  
    'Event Object Identifier' = O1  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),  
    'Notify Type' = ALARM | EVENT,  
    'AckRequired' = TRUE | FALSE,  
    'From State' = FAULT,  
    'To State' = NORMAL,  
    'Event Values' = (NO\_FAULT\_DETECTED,  
    (F, F, ? ?),  
    (A list of valid values for properties required to be reported for O1, and 0 or more other properties of O1))
  9. pCurrentReliability = NO\_FAULT\_DETECTED
  10. VERIFY pCurrentState = NORMAL

#### 8.5.17.12.2 FAULT-to-FAULT transition (UnconfirmedEventNotification)

Purpose: This test case verifies the correct operation of the FAULT\_LISTED event algorithm for objects transitioning from FAULT to FAULT event states.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_LISTED algorithm. Ensure that a fault condition, FV1, exists in the object. Set pMonitoredList to FV2, a non-empty list different from FV1. Verify the correct transition is generated.

Configuration Requirements: O1 is configured to detect faults and to report those using unconfirmed event notifications. O1 is initially configured to have a fault by having pMonitoredList contain a non-empty list, FV1, and has an Event\_State of FAULT.

Test Steps:

1. VERIFY pCurrentReliability = FAULTS\_LISTED
2. VERIFY pCurrentState = FAULT
3. IF (pMonitoredList is writable) THEN  
    WRITE pMonitoredList = FV2  
ELSE  
    MAKE (pMonitoredList = FV2)
4. BEFORE **Notification Fail Time**  
    RECEIVE UnconfirmedEventNotification-Request,  
    'Process Identifier' = (any valid process Identifier),  
    'Initiating Device Identifier' = IUT  
    'Event Object Identifier' = O1  
    'Time Stamp' = (any valid time stamp),  
    'Notification Class' = (the notification class configured for O1),  
    'Priority' = (the value configured for the transition),  
    'Event Type' = CHANGE\_OF\_RELIABILITY,  
    'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = FAULT,  
 'Event Values' = (FAULT\_LISTED,  
 (T, T, ? ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))

5. VERIFY pCurrentReliability = FAULTS\_LISTED

6. VERIFY pCurrentState = FAULT

#### 8.5.17.13 CHANGE\_OF\_RELIABILITY with the FAULT\_OUT\_OF\_RANGE Algorithm (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the FAULT\_OUT\_OF\_RANGE event algorithm.

Test Concept: Select a fault detecting object, O1, which is configured to use the FAULT\_OUT\_OF\_RANGE algorithm. Ensure that no other fault conditions exist in the object. Set pMonitoredValue to outside the range of values considered to be normal for the object. Verify the correct transition is generated. The fault condition is then removed. It is verified that O1 generates the correct notifications.

Configuration Requirements: O1 is configured to detect and report faults, to have no fault conditions present. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.17.12.3, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.17.12.3, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

#### 8.5.17.14 CHANGE\_OF\_RELIABILITY with First Stage Object Fault (UnconfirmedEventNotifications)

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

Test Concept: An object in the IUT, O1, which can detect at least one first stage fault is selected. One of O1's detectable first stage faults, R1, is selected for the test. O1 begins the test in the NORMAL state with pCurrentReliability equal to NO\_FAULT\_DETECTED. The first stage fault condition, R1, is made to exist and it is verified that the pCurrentReliability changes to R1. It is verified that O1 generates the appropriate event notification. The fault condition is removed, and it is verified that the pCurrentReliability returns to NO\_FAULT\_DETECTED and the appropriate event notification message is generated.

Configuration Requirements: O1 is configured to detect faults and to report. O1 is initially configured to have no fault conditions present, and Event\_State is NORMAL. The 'Issue Confirmed Notifications' parameter shall have a value of FALSE.

Test Steps:

1. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
2. VERIFY pCurrentState = NORMAL
3. MAKE (pCurrentReliability = R1)
4. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' = (any valid process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = O1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for O1),

'Priority' = (the value configured for the transition),

'Event Type' = CHANGE\_OF\_RELIABILITY,



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

5. VERIFY pCurrentReliability = R1
6. VERIFY pCurrentState = FAULT
7. MAKE (pCurrentReliability = NO\_FAULT\_DETECTED)
8. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = O1,  
 'Time Stamp' = (any current time stamp),  
 'Notification Class' = (the notification class configured for O1),  
 'Priority' = (the value configured for the transition),  
 'Event Type' = CHANGE\_OF\_RELIABILITY,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = EVENT | ALARM,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 (A list of valid values for properties required to be reported  
 for O1, and 0 or more other properties of O1))
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
10. VERIFY pCurrentState = NORMAL

#### 8.5.18 CHANGE\_OF\_DISCRETE\_VALUE Test (UnconfirmedEventNotification)

Purpose: To verify correct operation of the CHANGE\_OF\_DISCRETE\_VALUE event algorithm. This test applies to Event Enrollment objects with an Event\_Type of CHANGE\_OF\_DISCRETE\_VALUE.

Test Concept: pMonitoredValue is changed to a value different from the initial value. After pTimeDelayNormal (pTimeDelay is the value for Time\_Delay specified in the EventParameters), a TO-NORMAL transition occurs and a UnconfirmedEventNotification is generated by the IUT.

Configuration Requirements: An Event\_Enrollment, EE1 is configured with an Object\_Property\_Reference, (O1, P1), such that P1 is of one of the following datatypes: BOOLEAN, Unsigned, Integer, Enumerated, CharacterString, Octet String, Date, Time, BACnetObjectIdentifier, or BACnetDateTime. Event\_Enable is configured with a value of (T,T,T) and Event\_Algorithm\_Inhibit = FALSE. The Event\_Parameters are configured with an Event Algorithm of CHANGE\_OF\_DISCRETE\_VALUE and a value for Time\_Delay that is within the allowable range for the IUT. The configured notification class is configured to send unconfirmed notifications to the TD. EE1 shall have an Event\_State of NORMAL at the start of the test.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. MAKE (the referenced property has a value x: x differs from the initial value)
3. WAIT (pTimeDelay)
4. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification  
 'Process Identifier' = (any valid process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = EE1,  
 'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for EE1),  
 'Priority' = (the value configured to correspond to TO-NORMAL),  
 'Event Type' = CHANGE\_OF\_DISCRETE\_VALUE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'Ack Required' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = NORMAL,  
 'Event Values' = (x, Status\_Flags of O1)

5. VERIFY pCurrentState = NORMAL

#### 8.5.19 ACCESS\_EVENT Test (UnconfirmedEventNotification)

Purpose: To verify the correct operation of the ACCESS\_EVENT event algorithm.

Test Concept: The object, O1, begins the test in a NORMAL state. An access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT. A second access event, of a type not listed in pAccessEvents, is made to occur, if such is supported by the IUT. It is verified that no notification is generated. A third access event, of a type listed in pAccessEvents is made to occur. It is verified that the IUT sends a confirmed notification of type ACCESS\_EVENT.

Configuration Requirements: The IUT shall be configured such that the Event\_Enable property has a value of TRUE for TO-NORMAL transitions. The Issue\_Confirmed\_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. pAccessEvents shall be configured with at least 1 access event type that can be made to occur. If possible, at least access event type that can be made to occur shall not be included in pAccessEvents.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.19 except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.19 except that the event notification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

#### 8.5.20 CHANGE\_OF\_TIMER Tests

##### 8.5.20.1 CHANGE\_OF\_TIMER UnconfirmedEventNotification Test

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

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

Test Steps: The test steps for this test case are identical to the test steps in 8.4.20.1, except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.20.1, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

##### 8.5.20.2 CHANGE\_OF\_TIMER Offnormal-to-Offnormal UnconfirmedEventNotification Test

Purpose: To verify the correct operation of the CHANGE\_OF\_TIMER event algorithm. This test applies to objects that support an Event\_Type of CHANGE\_OF\_TIMER.

Test Concept: The object O1 begins the test in a NORMAL state. The pMonitoredValue is changed multiple times, typically by putting the Timer into operation where it conducts its usual operations, such that at multiple points the pMonitoredValue is equal to any of the values contained in pAlarmValues. After the object enters the OFFNORMAL state and transmits a first event notification message, when pMonitoredValue changes but is again equal to any of the values contained in pAlarmValues, it is observed to transmit another event notification message. Finally, the

pMonitoredValue becomes a value that is not equal to any of the values contained in pAlarmValues. After pTimeDelayNormal time holding at that value, the object shall enter the NORMAL state and transmit an event notification message.

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

Test Steps: The test steps for this test case are identical to the test steps in 8.4.20.2 except that the ConfirmedEventNotification requests are UnconfirmedEventNotification requests, and the TD does not acknowledge receiving the notifications.

Notes to Tester: The passing results for this test case are identical to the ones in 8.4.20.2, except that the event notifications shall be conveyed using an UnconfirmedEventNotification service request. The MAC address used for these messages shall be either a broadcast that reaches the local network of the TD or the MAC address of the TD.

## 8.6 GetAlarmSummary Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating GetAlarmSummary service requests.

### 8.6.1 Basic GetAlarmSummary Service Initiation

Purpose: To verify that the IUT can initiate GetAlarmSummary service requests.

Test Steps:

1. RECEIVE GetAlarmSummary-Request
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.6.2 Updating Alarm Summary Information with GetAlarmSummary

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetAlarmSummary service.

Configuration Requirements: The TD shall be configured to not support execution of GetEventInformation nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetAlarmSummary-Request
3. TRANSMIT GetAlarmSummary-Ack  
'List of Alarm Summaries' = (any valid list)
4. CHECK (that the IUT presents or updates the presentation of the alarm summary information and that the presentation is consistent with the information received in step 3)

Notes to Tester: The value presented by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

If the IUT has not already determined that the TD does not support execution of GetEventInformation and/or GetEnrollmentSummary, the IUT may initiate a GetEventInformation and/or GetEnrollmentSummary. If this occurs, the IUT shall only pass the test if it automatically falls back to using GetAlarmSummary upon receipt of the correct BACnetError-PDU from the TD, indicating that the alternate service is not supported.

## 8.7 GetEnrollmentSummary Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating GetEnrollmentSummary service requests.

### 8.7.1 Acknowledgment Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying an Acknowledgment Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = (any valid Acknowledgment Filter enumeration)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.7.2 Enrollment Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying an Enrollment Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Enrollment Filter' = (any valid BACnetRecipientProcess)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.7.3 Event State Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying an Event State Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Event State Filter' = (any valid Event State Filter enumeration)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.7.4 Event Type Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying an Event Type Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Event Type Filter' = (any valid BACnetEventType enumeration)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.7.5 Priority Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying a Priority Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Priority Filter' = (any valid priority range)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

### 8.7.6 Notification Class Filter

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying a Notification Class Filter.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Notification Class Filter' = (any valid Notification Class)

2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

#### 8.7.7 Multiple Filters

Purpose: To verify that the IUT can initiate GetEnrollmentSummary service requests conveying multiple Filters.

Test Steps:

1. RECEIVE GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = (any valid Acknowledgment Filter enumeration)  
'Enrollment Filter' = (any valid BACnetRecipientProcess)  
'Event State Filter' = (any valid Event State Filter enumeration)  
'Event Type Filter' = (any valid BACnetEventType enumeration)  
'Priority Filter' = (any valid priority range)  
'Notification Class Filter' = (any valid Notification Class)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Alarm Summaries' = (an empty list)

#### 8.8 GetEventInformation Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating GetEventInformation service requests.

##### 8.8.1 Without Chaining

Purpose: To verify that the IUT can initiate a simple GetEventInformation service request.

Test Steps:

1. RECEIVE GetEventInformation-Request,  
'Last Received Object Identifier' = (none)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Event Summaries' = (any valid list)  
'More Events' = FALSE
3. CHECK(that the IUT exhibits the vendor defined results)

##### 8.8.2 With Chaining

Purpose: To verify that the IUT can respond to the “more events” flag by initiating a GetEventInformation service request with the chaining parameter “last received object identifier” present.

Test Steps:

1. RECEIVE GetEventInformation-Request,  
'Last Received Object Identifier' = (none)
2. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Event Summaries' = (a non-empty list)  
'More Events' = TRUE
3. BEFORE the tester's patience is exceeded  
RECEIVE GetEventInformation-Request,  
'Last Received Object Identifier' = (last object identifier sent in step 2)
4. TRANSMIT BACnet-ComplexACK-PDU,  
'List of Event Summaries' = (a non-empty list)  
'More Events' = FALSE
5. CHECK(that the IUT exhibits the vendor defined results)

##### 8.8.3 Updating Alarm Summary Information with GetEventInformation Without Chaining

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetEventInformation service.

Configuration: The TD shall be configured to not support execution of GetAlarmSummary nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary. The number of

objects that will be reported via GetEventInformation shall not exceed the number that can be reported in a single GetEventInformation-Ack response.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetEventInformation-Request
3. TRANSMIT GetEventInformation-Ack,  
     'List of Event Summaries' = (any valid list),  
     'More Events' = FALSE
4. CHECK (that the IUT presents or updates the alarm summary information presented to the user and that the presentation is consistent with the information received in step 3)

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

If the IUT has not already determined that the TD does not support execution of GetAlarmSummary and/or GetEnrollmentSummary, the IUT may initiate a GetAlarmSummary and/or GetEnrollmentSummary. If this occurs, the IUT shall pass the test only if it automatically falls back to using GetEventInformation upon receipt of the correct BACnetError-PDU from the TD, indicating that alternate service is not supported.

#### 8.8.4 Updating Alarm Summary Information with GetEventInformation With Chaining

Purpose: This test case verifies that the IUT is capable of updating or presenting alarm summary information using the GetEventInformation service.

Configuration: The TD shall be configured to not support execution of GetAlarmSummary nor GetEnrollmentSummary. The TD shall be configured with a set of event-generating objects in a variety of event states which the IUT can interrogate during the execution of the test and some of which will be reported to the IUT via GetAlarmSummary. The number of objects that will be reported via GetEventInformation shall exceed the number that can be reported in a single GetEventInformation-Ack response.

Test Steps:

1. MAKE (the IUT present or update the alarm summary presented to the user)
2. RECEIVE GetEventInformation-Request
3. TRANSMIT GetEventInformation-Ack,  
     'List of Event Summaries' = (any valid list that represents the state of event generating objects in the TD),  
     'More Events' = TRUE
4. RECEIVE GetEventInformation-Request,  
     'Last Received Object Identifier' = (last object identifier sent in step 3)
5. TRANSMIT GetEventInformation-Ack,  
     'List of Event Summaries' = (any valid list that represents the state of event generating objects in the TD),  
     'More Events' = FALSE
6. CHECK (that the IUT presents or updates the alarm summary information presented to the user and that the presentation is consistent with the information received in steps 3 and 5)

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

If the IUT has not already determined that the TD does not support execution of GetAlarmSummary and/or GetEnrollmentSummary, the IUT may initiate a GetAlarmSummary and/or GetEnrollmentSummary. If this occurs, the IUT shall pass the test only if it automatically falls back to using GetEventInformation upon receipt of the correct BACnetError-PDU from the TD, indicating that alternate service is not supported.

### 8.9 LifeSafetyOperation Service Initiation Tests

#### 8.9.1 LifeSafetyOperation Service Initiation Tests to an Object

This clause defines the tests necessary to demonstrate support for initiating LifeSafetyOperation service requests intended for a single Life Safety Point or Life Safety Zone object.



Test Steps:

Purpose: To verify that the IUT can correctly initiate a LifeSafetyOperation service request.

Test Steps:

1. RECEIVE LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid identifier),  
     'Requesting Source' = (any valid character string),  
     'Request' = (any valid LifeSafetyOperation request),  
     'Object Identifier' = (any valid Life Safety Point or Life Safety Zone object identifier)

### 8.9.2 LifeSafetyOperation Service Initiation Tests to all Objects in a Device

This clause defines the tests necessary to demonstrate support for initiating LifeSafetyOperation service requests intended for all Life Safety Point and Life Safety Zone objects in a device.

Purpose: To verify that the IUT can correctly initiate a LifeSafetyOperation service request to a Life Safety Object.

Test Steps:

1. RECEIVE LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid identifier),  
     'Requesting Source' = (any valid character string),  
     'Request' = (any valid LifeSafetyOperation request)

### 8.10 SubscribeCOV Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating SubscribeCOV service requests.

#### 8.10.1 Confirmed Notifications Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOV service request for confirmed notifications.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any identifier for a standard object type for which COV reporting is defined),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any non-zero value)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.10.2 Unconfirmed Notifications Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOV service request for unconfirmed notifications.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any identifier for a standard object type for which COV reporting is defined),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (any non-zero value)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.10.3 Canceling a Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOV service request to cancel a subscription.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),

'Monitored Object Identifier' = (any identifier for a standard object type for which COV reporting is defined)

## 2. TRANSMIT BACnet-SimpleACK-PDU

### 8.10.4 Requests 8 Hour Lifetimes

Purpose: To verify that the IUT correctly generates subscription requests with lifetimes less than or equal to 8 hours. Either confirmed or unconfirmed notifications may be used, but at least one of these options shall be supported by the IUT.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (any valid lifetime between 1 and 28800)
2. TRANSMIT BACnet-SimpleACK-PDU

### 8.11 SubscribeCOVProperty Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating SubscribeCOVProperty service requests.

#### 8.11.1 Confirmed Notifications Subscription

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

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

Test Steps:

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

#### 8.11.2 Unconfirmed Notifications Subscription

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

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

Test Steps:

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

### 8.11.3 Canceling a Subscription

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

Test Steps:

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

### 8.11.4 Change of Value Notification Tests

#### 8.11.4.1 Change of Value Notification

Purpose: To verify that the IUT can execute COVNotification requests from object types that provides a Property and Status\_Flags properties in COV notifications.

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

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value – optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - TRANSMIT ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the process identifier used in step 1),
      - 'Initiating Device Identifier' = TD,
      - 'Monitored Object Identifier' = X
      - 'Time Remaining' = (any value appropriate for the Lifetime selected),
      - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
    - RECEIVE BACnet-SimpleACK-PDU
  - ELSE
    - TRANSMIT UnconfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the process identifier used in step 1),
      - 'Initiating Device Identifier' = TD,
      - 'Monitored Object Identifier' = X
      - 'Time Remaining' = (any value appropriate for the Lifetime selected),
      - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 8.11.4.2 Change of Value Notifications with Invalid Process Identifier

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

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

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (a process identifier different from the one used in step 1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X
  - 'Time Remaining' = (any value appropriate for the Lifetime selected),
  - 'List of Values' = (values appropriate to the property Y subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN
  - RECEIVE
    - BACnet-Error-PDU,
    - 'Error Class' = SERVICES,
    - 'Error Code' = (UNKNOWN\_SUBSCRIPTION)
 (BACnet-SimpleACK-PDU)
  - ELSE
    - RECEIVE
      - BACnet-Error-PDU,
      - 'Error Class' = SERVICES,
      - 'Error Code' = (any valid error code for class SERVICES) |
 (BACnet-SimpleACK-PDU)

#### 8.11.4.3 Change of Value Notification Arrives after Subscription has Expired

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

Test Concept: A subscription for COV notifications is established and then cancelled or allowed to expire. A ConfirmedCOVNotification is then sent to the IUT to verify it returns the appropriate error or a Simple-Ack.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = X
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = L,
  - 'Monitored Property Identifier' = (the property Y to be monitored),
  - 'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. BEFORE **Notification Fail Time**
  - TRANSMIT ConfirmedCOVNotification-Request,
    - 'Subscriber Process Identifier' = (the process identifier used in step 1),
    - 'Initiating Device Identifier' = TD,
    - 'Monitored Object Identifier' = X
    - 'Time Remaining' = (any value appropriate for the Lifetime selected),
    - 'List of Values' = (values appropriate to the property Y subscribed to, and any other

- properties the IUT provides with it, such as Status\_Flags)  
 RECEIVE BACnet-SimpleACK-PDU
5. MAKE (the IUT stop resubscribing, if it resubscribes automatically)
  6. IF (the IUT can cancel the subscription) THEN  
 RECEIVE SubscribeCOVProperty – Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Monitored Object Identifier' = X  
     'Monitored Property Identifier' = Y  
     'COV Increment' = (Any REAL value –optional)  
 ELSE  
     WAIT (a value two times Lifetime)  
 7. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (values appropriate to the property Y subscribed to, and any other  
     properties the IUT provides with it, such as Status\_Flags)
  8. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN  
 RECEIVE  
 BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = (UNKNOWN\_SUBSCRIPTION) |  
 (BACnet-SimpleACK-PDU)  
 ELSE  
 RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = (any valid error code for class SERVICES) |  
 (BACnet-SimpleACK-PDU)

#### 8.11.4.4 Change of Value Notifications with Invalid Monitored Object Identifier

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L,  
     'Monitored Property Identifier' = (the property Y to be monitored),  
     'COV Increment' = (Any REAL value -- optional)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = (any object Y supporting COV notification except X),  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (any value)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN  
 RECEIVE  
 BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = (UNKNOWN\_SUBSCRIPTION) |  
 (BACnet-SimpleACK-PDU)

```

ELSE
 RECEIVE
 BACnet-Error-PDU,
 'Error Class' = SERVICES,
 'Error Code' = (any valid error code for class SERVICES) |
 (BACnet-SimpleACK-PDU)

```

#### 8.11.4.5 Change of Value Notifications with Invalid Monitored Property

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request
 

```

 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = X
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = L,
 'Monitored Property Identifier' = (the property Y to be monitored),
 'COV Increment' = (Any REAL value -- optional)

```
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotification-Request,
 

```

 'Subscriber Process Identifier' = (the process identifier used in step 1),
 'Initiating Device Identifier' = TD,
 'Monitored Object Identifier' = X
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (any property supporting COV notification except Y),

```
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN
 RECEIVE
 BACnet-Error-PDU,
 'Error Class' = SERVICES,
 'Error Code' = (UNKNOWN\_SUBSCRIPTION) |
 (BACnet-SimpleACK-PDU)
 ELSE
 RECEIVE
 BACnet-Error-PDU,
 'Error Class' = SERVICES,
 'Error Code' = (any valid error code for class SERVICES) |
 (BACnet-SimpleACK-PDU)

#### 8.11.5 Requests 8 Hour Lifetimes

Purpose: To verify that the IUT correctly generates subscription requests with lifetimes less than or equal to 8 hours. Either confirmed or unconfirmed notifications may be used, but at least one of these options shall be supported by the IUT.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVProperty-Request),
2. RECEIVE SubscribeCOVProperty-Request,
 

```

 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = X
 'Issue Confirmed Notifications' = TRUE | FALSE,
 'Lifetime' = (any valid lifetime between 1 and 28800),
 'Monitored Property Identifier' = (the property Y to be monitored),
 'COV Increment' = (Any REAL value -- optional)

```
3. TRANSMIT BACnet-SimpleACK-PDU



### 8.12 AtomicReadFile Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating AtomicReadFile Service requests. The BACnet file access services permit access to files on either a record basis or a stream basis. If the IUT supports stream-based file structures then the stream access test (8.12.1) shall be performed. If the IUT supports record-based file structures then the record access test (8.12.2) shall be performed.

#### 8.12.1 Stream Access

Purpose: To verify that the IUT can initiate an AtomicReadFile service request using the stream-oriented file access method.

Test Steps:

1. RECEIVE AtomicReadFile-Request,  
     'File Start Position' = (any value  $\geq 0$ ),  
     'Requested Octet Count' = (any value  $> 0$ )
2. TRANSMIT AtomicReadFile-ACK,  
     'End Of File' = TRUE,  
     'File Start Position' = (the start position indicated in step 1),  
     'File Data' = (any stream file data of size  $\leq$  'Requested Octet Count' from step 1)
3. CHECK (if the IUT displays the file data, verify that it is correct)

#### 8.12.2 Record Access

Purpose: To verify that the IUT can initiate an AtomicReadFile service request using the record-oriented file access method.

Test Steps:

1. RECEIVE AtomicReadFile-Request,  
     'File Start Record' = (any value  $\geq 0$ ),  
     'Requested Record Count' = (any value  $> 0$ )
2. TRANSMIT AtomicReadFile-ACK,  
     'End Of File' = TRUE,  
     'File Start Record' = (the start position indicated in step 1),  
     'Returned Record Count' = (any value  $\leq$  the 'Requested Record Count' from step 1),  
     'File Record Data' = (any record file data containing the indicated number of records)
3. CHECK (if the IUT displays the file data, verify that it is correct)

### 8.13 AtomicWriteFile Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating AtomicWriteFile service requests. The BACnet file access services permit access to files on either a record basis or a stream basis. If the IUT supports stream-based file structures then the stream access test (8.13.1) shall be performed. If the IUT supports record-based file structures then the record access test (8.13.2) shall be performed.

#### 8.13.1 Stream Access

Purpose: To verify that the IUT can initiate an AtomicWriteFile service request using the stream-oriented file access method.

Test Steps:

1. RECEIVE AtomicWriteFile-Request,  
     'File Start Position' = (any position  $\geq -1$ ),  
     'File Data' = (any stream file data containing at least one octet)
2. TRANSMIT AtomicWriteFile-ACK,  
     'File Start Position' = (0 if the 'File Start Position' was -1, otherwise the indicated start position)

#### 8.13.2 Record Access

Purpose: To verify that the IUT can initiate an AtomicWriteFile service request using the record-oriented file access method.

Test Steps:

1. RECEIVE AtomicWriteFile-Request,  
     'File Start Record' = (any record  $\geq -1$ ),  
     'Record Count' = (any value  $> 0$ ),  
     'File Record Data' = (any record file data containing the indicated number of records)
2. TRANSMIT AtomicWriteFile-ACK,  
     'File Start Record' = (0 if the 'File Start Position' was -1, otherwise the indicated start position)

#### 8.14 AddListElement Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating AddListElement service requests.

##### 8.14.1 Non-Array Properties

Purpose: To verify that the IUT can initiate an AddListElement service request that does not contain the 'Property Array Index' parameter.

Test Steps:

1. RECEIVE AddListElement-Request,  
     'Object Identifier' = (any object that contains a property having a list datatype),  
     'Property Identifier' = (any property having a list datatype),  
     'List of Elements' = (one or more elements with the correct datatype to add to the list)
2. TRANSMIT BACnet-SimpleACK-PDU

##### 8.14.2 Array Properties

Purpose: To verify that the IUT can initiate an AddListElement service request that contains the 'Property Array Index' parameter.

Test Steps:

1. RECEIVE AddListElement-Request,  
     'Object Identifier' = (any object that contains a property having a datatype that is an array of lists),  
     'Property Identifier' = (any property having a datatype that is an array of lists),  
     'Property Array Index' = (any value  $> 0$ ),  
     'List of Elements' = (one or more elements with the correct datatype to add to the list)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.15 RemoveListElement Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating RemoveListElement service requests.

##### 8.15.1 Non-Array Properties

Purpose: To verify that the IUT can initiate a RemoveListElement service request that does not contain the 'Property Array Index' parameter.

Test Steps:

1. RECEIVE RemoveListElement-Request,  
     'Object Identifier' = (any object that contains a property having a list datatype),  
     'Property Identifier' = (any property having a list datatype),  
     'List of Elements' = (one or more elements with the correct datatype to remove from the list)
2. TRANSMIT BACnet-SimpleACK-PDU

##### 8.15.2 Array Properties

Purpose: To verify that the IUT can initiate a RemoveListElement service request that contains the 'Property Array Index' parameter.

Test Steps:

1. RECEIVE RemoveListElement-Request,

- 'Object Identifier' = (any object that contains a property having a datatype that is an array of lists),
- 'Property Identifier' = (any property having a datatype that is an array of lists),
- 'Property Array Index' = (any value > 0),
- 'List of Elements' = (one or more elements with the correct datatype to remove from the list)

## 2. TRANSMIT BACnet-SimpleACK-PDU

### 8.16 CreateObject Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating CreateObject service requests.

#### 8.16.1 Creating Objects by Specifying the Object Identifier with no Initial Values

Purpose: To verify that the IUT can initiate a CreateObject service request that contains an object identifier in the 'Object Specifier' parameter and no initial property values.

Test Steps:

1. RECEIVE CreateObject-Request,  
'Object Specifier' = (any BACnetObjectIdentifier)
2. TRANSMIT CreateObject-ACK,  
'Object Identifier' = (the object identifier specified in step 1)

#### 8.16.2 Creating Objects by Specifying the Object Type with no Initial Values

Purpose: To verify that the IUT can initiate a CreateObject service request that contains an object type in the 'Object Specifier' parameter and no initial property values.

Test Steps:

1. RECEIVE CreateObject-Request,  
'Object Specifier' = (any BACnetObjectType),
2. TRANSMIT CreateObject-ACK,  
'Object Identifier' = (an object identifier consistent with the object type specified in step 1)

#### 8.16.3 Creating Objects by Specifying the Object Identifier and Providing Initial Values

Purpose: To verify that the IUT can initiate a CreateObject service request that contains an object identifier in the 'Object Specifier' parameter and a list of initial property values for the object to be created.

Test Steps:

1. RECEIVE CreateObject-Request,  
'Object Specifier' = (any BACnetObjectIdentifier)  
'List of Initial Values' = (a list of one or more properties and their initial values that the IUT will accept)
2. TRANSMIT CreateObject-ACK,  
'Object Identifier' = (the object identifier specified in step 1)

#### 8.16.4 Creating Objects by Specifying the Object Type and Providing Initial Values

Purpose: To verify that the IUT can initiate a CreateObject service request that contains an object type in the 'Object Specifier' parameter and a list of initial property values for the object to be created.

Test Steps:

1. RECEIVE CreateObject-Request,  
'Object Specifier' = (any BACnetObjectType),  
'List of Initial Values' = (a list of one or more properties and their initial values that the IUT will accept)
2. TRANSMIT CreateObject-ACK,  
'Object Identifier' = (an object identifier consistent with the object type specified in step 1)

### 8.17 DeleteObject Service Initiation Tests

This clause defines the test necessary to demonstrate support for initiating DeleteObject service requests.

Purpose: To verify that the IUT can initiate a DeleteObject service request.

Test Steps:

1. RECEIVE DeleteObject-Request,  
    'Object Identifier' = (any object identifier)
2. TRANSMIT BACnet-SimpleACK-PDU

### 8.18 ReadProperty Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ReadProperty service requests.

#### 8.18.1 Reading Non-Array Properties

Purpose: To verify that the IUT can initiate a ReadProperty service request that does not contain the 'Property Array Index' parameter and can correctly process the response.

Test Steps:

1. RECEIVE ReadProperty-Request,  
    'Object Identifier' = (any object),  
    'Property Identifier' = (any valid non-array property of the specified object)
2. TRANSMIT BACnet-ComplexACK-PDU,  
    'Object Identifier' = (object identifier from step 1),  
    'Property Identifier' = (property identifier from step 1),  
    'Property Value' = (any valid value for the property)
3. CHECK (that the IUT exhibits the vendor defined results)

#### 8.18.2 Reading an Array Element

Purpose: To verify that the IUT can initiate a ReadProperty service request that references a specific element of an array property and can correctly process the response.

Test Steps:

1. RECEIVE ReadProperty-Request,  
    'Object Identifier' = (any object),  
    'Property Identifier' = (any valid array property of the specified object),  
    'Property Array Index' = (any valid ARRAY INDEX for the specified property)
2. TRANSMIT BACnet-ComplexACK-PDU,  
    'Object Identifier' = (object identifier from step 1),  
    'Property Identifier' = (property identifier from step 1),  
    'Array Index' = (ARRAY INDEX from step 1),  
    'Property Value' = (any valid value for the property)
3. CHECK (that the IUT exhibits the vendor defined results)

#### 8.18.3 Reading and Presenting Properties

Purpose: This test case verifies that the IUT is capable of reading and presenting properties. It is a generic test used to test data presentation requirements.

Configuration: For this test, the tester shall choose a property, P1, from an object, O1. The TD shall be configured to not support the execution of ReadPropertyMultiple or the initiation of COV notifications.

Test Steps:

1. MAKE (the IUT read P1)
2. RECEIVE ReadProperty-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1
3. TRANSMIT BACnet-ComplexACK-PDU,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1,  
    'Property Value' = (any valid value for P1)
4. CHECK (that the IUT presents a value that is consistent with the value received in step 3)

Notes to Tester: The value presented by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the property being read and displayed is an array, the IUT may include an Array Index parameter in the ReadProperty-Request in step 2 and the TD will include it in the response in step 4.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of ReadPropertyMultiple, SubscribeCOV, and SubscribeCOVProperty, the IUT may initiate any of these services. If this occurs, the IUT shall pass the test only if it automatically falls back to using ReadProperty upon receipt of the correct BACnetReject-PDU(s) from the TD, indicating that other service(s) is not supported.

#### 8.18.4 Reading Whole Array Properties

Purpose: To verify that the IUT can initiate a ReadProperty service request that does not contain the 'Property Array Index' parameter for an array property and can correctly process the response.

Test Steps:

1. RECEIVE ReadProperty-Request,  
     'Object Identifier' = (any object),  
     'Property Identifier' = (any valid array property of the specified object)
2. TRANSMIT BACnet-ComplexACK-PDU,  
     'Object Identifier' = (object identifier from step 1),  
     'Property Identifier' = (property identifier from step 1),  
     'Property Value' = (any valid array of values for the property)
3. CHECK (that the IUT exhibits the vendor defined results)

#### 8.18.5 Reading an Array Size

Purpose: To verify that the IUT can initiate a ReadProperty service request for the size of an array property and can correctly process the response.

Test Steps:

1. RECEIVE ReadProperty-Request,  
     'Object Identifier' = (any object),  
     'Property Identifier' = (any valid array property of the specified object),  
     'Priority Array Index' = 0
2. TRANSMIT BACnet-ComplexACK-PDU,  
     'Object Identifier' = (object identifier from step 1),  
     'Property Identifier' = (property identifier from step 1),  
     Array Index' = 0,  
     'Property Value' = (any valid size for the array)
3. CHECK (that the IUT exhibits the vendor defined results)

#### 8.18.6 Reading and Presenting Large List Properties

Purpose: This test case verifies that the IUT is capable of reading and presenting large list properties using ReadRange. It is a generic test used to test data presentation requirements.

Configuration: For this test, the tester shall choose a list property, P1, from an object, O1. The TD shall be configured to not support segmentation. The value P1 shall be too large to read via ReadProperty or ReadPropertyMultiple.

Notes to Tester: The value presented by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the IUT has not already determined that the value cannot be read using ReadProperty or ReadPropertyMultiple, the IUT may initiate a ReadProperty or ReadPropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using ReadRange upon receipt of the correct BACnetReject-PDU from the TD, indicating that the response is too large.

Test Steps:

1. MAKE (the IUT read P1)
2. WHILE (the complete list has not been read)
  - RECEIVE ReadRange-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Range' = (any valid value for P1)
  - TRANSMIT BACnet-ComplexACK-PDU,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Result Flags' = (values consistent with the request),
  - 'Item Count' = (values consistent with the request),
  - 'Item Data' = (values consistent with the request)
3. CHECK (that the IUT presents a list of values that is consistent with the values received in step 2)

### 8.19 ReadPropertyConditional Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ReadPropertyConditional service requests.

#### 8.19.1 Reading Object Identifiers of Objects that Meet the Selection Criteria

Purpose: To verify that the IUT can initiate a ReadPropertyConditional service request in which objects are chosen based on some selection criteria and no 'List of Property References' is specified.

Test Steps:

1. RECEIVE ReadPropertyConditional-Request,
- 'Selection Logic' = AND,
- 'List of Selection Criteria' = ((any non-array property identifier, any relation specifier, any valid comparison value),  
(any array property identifier, any ARRAY INDEX, any relation specifier, any valid comparison value)  
)

#### 8.19.2 Reading Specific Properties of Objects that Meet the Selection Criteria

Purpose: To verify that the IUT can initiate a ReadPropertyConditional service request in which objects are chosen based on some selection criteria and a set of properties to be read is specified.

Test Steps:

1. RECEIVE ReadPropertyConditional-Request,
- 'Selection Logic' = OR,
- 'List of Selection Criteria' = ((any non-array property identifier, any relation specifier, any valid comparison value),  
(any array property identifier, any ARRAY INDEX, any relation specifier, any valid comparison value)  
)
- 'List of Property References' = (two or more properties that correspond to objects meeting the selection criteria)

### 8.20 ReadPropertyMultiple Service Initiation Tests

Purpose: This clause defines the tests necessary to demonstrate support for initiating ReadPropertyMultiple service requests. At least one of the combinations defined in 8.20.2 through 8.20.4 needs to be supported in order to claim the ability to initiate the ReadPropertyMultiple service.

#### 8.20.1 Reading a Single Property of a Single Object

Purpose: To verify that the IUT can correctly initiate a ReadPropertyMultiple service request containing a single property of a single object. If the IUT cannot be configured to initiate a ReadPropertyMultiple service request containing a single property of a single object, then this test shall be omitted.



Test Steps:

1. RECEIVE ReadPropertyMultiple-Request,  
     'Object Identifier' = (any valid object identifier),  
     'List of Property References' = (a single property of the specified object)

#### 8.20.2 Reading Multiple Properties of a Single Object

Purpose: To verify that the IUT can correctly initiate a ReadPropertyMultiple service request containing multiple properties of a single object. If the IUT cannot be configured to initiate a ReadPropertyMultiple service request containing multiple properties of a single object, then this test shall be omitted.

Test Steps:

1. RECEIVE ReadPropertyMultiple-Request,  
     'Object Identifier' = (any valid object identifier),  
     'List of Property References' = (two or more properties of the specified object)

#### 8.20.3 Reading Multiple Objects, One Property Each

Purpose: To verify that the IUT can correctly initiate a ReadPropertyMultiple service request containing multiple objects and a single property for each object. If the IUT cannot be configured to initiate a ReadPropertyMultiple service request containing multiple objects and a single property for each object, then this test shall be omitted.

Test Steps:

1. RECEIVE ReadPropertyMultiple-Request,  
     'Object Identifier' = (any valid object identifier),  
     'List of Property References' = (a single property of the specified object),  
     'Object Identifier' = (any valid object identifier not previously used),  
     'List of Property References' = (a single property of the specified object),  
     ... (Including additional objects and properties is acceptable.)

#### 8.20.4 Reading Multiple Objects, Multiple Properties for Each

Purpose: To verify that the IUT can correctly initiate a ReadPropertyMultiple service request containing multiple objects and multiple properties for each object. If the IUT cannot be configured to initiate a ReadPropertyMultiple service request containing multiple objects and multiple properties for each object, then this test shall be omitted.

Test Steps:

1. RECEIVE ReadPropertyMultiple-Request,  
     'Object Identifier' = (any valid object identifier),  
     'List of Property References' = (two or more properties of the specified object)  
     'Object Identifier' = (any valid object identifier not previously used),  
     'List of Property References' = (two or more properties of the specified object)  
     ... (Including additional objects and properties is acceptable.)

#### 8.20.5 Cases In Which ReadProperty Shall Be Used After ReadPropertyMultiple Fails

The tests defined in this clause are used to verify that an IUT which initiates ReadPropertyMultiple is able to obtain external property values via the ReadProperty service when interoperating with a device that does not support the ReadPropertyMultiple service.

##### 8.20.5.1 The IUT Determines the TD does not Support the ReadPropertyMultiple Service

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the IUT determines the TD does not support the ReadPropertyMultiple service.

Test Concept: The IUT is configured in a manner that would normally cause it to access one or more properties in the TD via the ReadPropertyMultiple service. Prior to sending a ReadPropertyMultiple request, however, the IUT determines that the TD does not support the ReadPropertyMultiple service. The IUT instead attempts to access the TD's property values via the ReadProperty service (it is assumed that the IUT will make this determination by reading the TD's Protocol\_Services\_Supported property, but this test specifically does not attempt to verify this behavior).

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it is accessing one or more properties of a single or multiple objects in the TD via the ReadProperty and ReadPropertyMultiple services.

Notes to Tester: In step 3, observing a ReadProperty of the properties from Step 1 is what is expected, but it is acceptable if fewer properties are read with ReadProperty, if those which are omitted are not essential to the operation being performed.

Test Steps:

1. MAKE (a condition in the IUT that would normally cause it to send a ReadPropertyMultiple request to the TD to access one or more property values)
2. WAIT (a time interval specified by the vendor as sufficient for the IUT to determine that the TD does not support the ReadPropertyMultiple service)
3. REPEAT X = (the properties that the IUT is to read) DO {
  - RECEIVE ReadProperty-Request,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X)
  - TRANSMIT ReadProperty-Ack,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X),
  - 'Property Value' = (any valid value)

#### 8.20.5.2 Fallback to ReadProperty on Reject - UNRECOGNIZED\_SERVICE Response

Purpose: Verifies the IUT's ability to automatically change its service choice from ReadPropertyMultiple to ReadProperty when the TD returns a Reject-PDU and a Reject Reason of UNRECOGNIZED\_SERVICE.

Test Concept: The IUT is configured to send the TD a ReadPropertyMultiple request to access one or more properties of a single object. The TD responds with a Reject-PDU and a Reject Reason of UNRECOGNIZED\_SERVICE. With no additional configuration, the IUT sends one or more ReadProperty requests to the TD, where each ReadProperty request specifies an individual property from the original ReadPropertyMultiple request.

Configuration Requirements: The TD is configured so that it does not support the ReadPropertyMultiple service. The IUT is configured such that it attempts to acquire values from the TD using the ReadPropertyMultiple service without first interrogating the TD's Protocol\_Services\_Supported property. If the IUT cannot be configured in this way then this test shall be omitted.

Notes to Tester: In step 4, observing a ReadProperty of the properties from Step 1 is what is expected, but it is acceptable if fewer properties are read with ReadProperty, if those which are omitted are not essential to the operation being performed.

Test Steps:

1. MAKE (the IUT send a ReadPropertyMultiple-Request for a list of properties L)
2. RECEIVE ReadPropertyMultiple-Request,
  - the following is repeated for each object, O referenced in L
  - 'Object Identifier' = O,
  - 'List of Property References' = (one or more properties of O as specified in L)
2. TRANSMIT BACnet-Reject-PDU,
  - 'Reject Reason' = UNRECOGNIZED\_SERVICE
3. REPEAT X = (the properties from L in the order the IUT chooses) DO {
  - RECEIVE ReadProperty-Request,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X)
  - TRANSMIT ReadProperty-Ack,
  - 'Object Identifier' = (object identifier referenced by X),
  - 'Property Identifier' = (property identifier referenced by X),

'Property Value' = (any valid value)  
}

### 8.21 ReadRange Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ReadRange service requests.

#### 8.21.1 Reading Values with no Specified Range

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that does not specify any range of values to be returned.

Test Steps:

1. RECEIVE ReadRange-Request,  
'Object Identifier' = (O, any object),  
'Property Identifier' = (P, any list property the IUT can read)
2. TRANSMIT ReadRange-ACK  
'Object Identifier' = O,  
'Property Identifier' = P,  
'Result Flags' = (TRUE, (bLast), (NOT bLast)),  
'Item Count' = (C: any valid value)  
'Item Data' = (C valid records for the requested property)
3. CHECK(that the IUT performs the vendor specified action)

#### 8.21.2 Reading Values with an Array Index

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies an ARRAY INDEX.

Test Steps:

1. RECEIVE ReadRange-Request,  
'Object Identifier' = (any object with a property that is an array of lists),  
'Property Identifier' = (any property that is an array of lists),  
'Priority Array Index' = (an value > 0)

#### 8.21.3 Reading a Range of Values by Position

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies the range of values to be returned by position.

Test Steps:

1. RECEIVE ReadRange-Request,  
'Object Identifier' = (O, any object),  
'Property Identifier' = (P, any list property),  
'Reference Index' = (any Unsigned value),  
'Count' = (C1, any INTEGER value)
2. TRANSMIT ReadRange-ACK  
'Object Identifier' = O,  
'Property Identifier' = P,  
'Result Flags' = ((TRUE if the first was requested, FALSE otherwise), ?, ?),  
'Item Count' = (C2: any valid value <= |C|)  
'Item Data' = (C2 valid records for the requested property)
3. CHECK(that the IUT performs the vendor specified action)

#### 8.21.4 Reading a Range of Values by Time

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies the range of values to be returned by time.

Test Steps:

1. RECEIVE ReadRange-Request,

'Object Identifier' = (any Trend Log object),  
 'Property Identifier' = Log\_Buffer,  
 'Reference Time' = (any BACnetDateTime value),  
 'Count' = (any INTEGER value)

#### 8.21.5 Reading a Range of Values by Time Range

Purpose: To verify that the IUT can correctly initiate a ReadRange service request that specifies the range of values to be returned by time range.

Test Steps:

1. RECEIVE ReadRange-Request,  
 'Object Identifier' = (any Trend Log object),  
 'Property Identifier' = Log\_Buffer,  
 'Beginning Time' = (any BACnetDateTime value),  
 'Ending Time' = (any BACnetDateTime value)

#### 8.21.6 Reading a Range of Items Using Any Valid Range in Response to ConfirmedEventNotifications of the BUFFER\_READY Event Type

Purpose: To verify that the IUT can initiate a series of ReadRange service requests that access a contiguous set of log records by responding to ConfirmedEventNotification messages with an 'Event Type' of BUFFER\_READY.

Test Concept: The TD contains a log object that has two logical sets of log records, S1 and S2. The log continues to accumulate log records until logical record sets S3 and S4 are added to the buffer. The TD sends BUFFER\_READY event notifications to the IUT about the addition of S2 and S3. The IUT accesses the records in S2 and S3 by sending a series of ReadRange requests, using any valid range. The test then verifies that the IUT has accessed a contiguous set of records in S2, S3, and possibly some portion of S4. The size of sets S1, S2, S3, and S4 shall each be small enough that none of them causes the reference server to send an event notification.

The test verifies that the IUT does not simply read the entire buffer each time, nor miss samples in the Log\_Buffer.

Configuration Requirements: The TD contains a log object, L1, that has a Buffer\_Size property of sufficient number that it can contain the set of records S1, S2, S3, and S4, where the size of each set is large enough that the TD cannot return the entire set with a single ReadRange response (this behavior can be achieved by disabling the TD's ability to transmit segmented messages or by limiting the number of segments the TD can transmit). The TD is configured such that it adds entries to its Log\_Buffer for the duration of the test and will send a ConfirmedEventNotification of type BUFFER\_READY to the IUT after the addition of S2 and S3.

Test Steps:

1. MAKE (L1 contain record sets S1 + S2)
2. TRANSMIT ConfirmedEventNotification-Request,  
 'Subscriber Process Identifier' = (any valid value selected for the IUT),  
 'Initiating Device Identifier' = TD,  
 'Event Object Identifier' = (the object detecting the event),  
 'Time Stamp' = (any valid value),  
 'Notification Class' = (any valid notification class),  
 'Priority' = (any valid value),  
 'Event Type' = BUFFER\_READY,  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = NORMAL,  
 'Event Values' = (L1, Log\_Buffer),  
 (any Unsigned32 value that indicates the end of S1),  
 (any Unsigned32 value that indicates the addition of S2)
3. RECEIVE BACnet-SimpleACK-PDU
4. BEFORE (a time interval specified by the vendor)  
 WHILE (not all records from S2 have been returned by TD (see notes to tester))

- RECEIVE ReadRange-Request,  
 'Object Identifier' = (L1),  
 'Property Identifier' = (Log\_Buffer),  
 'Range' = (any valid range)
- TRANSMIT ReadRange-Ack,  
 'Object Identifier' = (L1),  
 'Property Identifier' = (Log\_Buffer),  
 'ItemData' = (a set of records appropriate for this response)
5. MAKE (L1 contain records S1 + S2 + S3)
6. TRANSMIT ConfirmedEventNotification-Request,  
 'Subscriber Process Identifier' = (any valid value selected for the IUT),  
 'Initiating Device Identifier' = TD,  
 'Event Object Identifier' = (the object detecting the event),  
 'Time Stamp' = (any valid value),  
 'Notification Class' = (any valid notification class),  
 'Priority' = (any valid value),  
 'Event Type' = BUFFER\_READY,  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = NORMAL,  
 'To State' = NORMAL,  
 'Event Values' = (L1, Log\_Buffer),  
 (any Unsigned32 value that indicates the end of S2),  
 (any Unsigned32 value that indicates the addition of S3)
7. RECEIVE BACnet-SimpleACK-PDU
8. MAKE (L1 contain records S1 + S2 + S3 + S4 (see note to tester))
9. BEFORE (a time interval specified by the vendor)  
 WHILE (not all records from S3 have been returned by DI)  
 RECEIVE ReadRange-Request,  
 'Object Identifier' = (L1),  
 'Property Identifier' = (Log\_Buffer),  
 'Range' = (any valid range)
- TRANSMIT ReadRange-Ack,  
 'Object Identifier' = (L1),  
 'Property Identifier' = (Log\_Buffer),  
 'ItemData' = (a set of records appropriate for this response)
10. CHECK (the records from all steps form a contiguous set, including all records in S2, S3, and possibly some portion of S4)

Note to Tester: Before step 4, the IUT may have pre-read some of S2. In such cases, the IUT shall not be failed for not reading those samples during step 4.

Note to Tester: The addition of S4 in step 8 can occur anytime after step 6 but before step 10.

#### 8.21.7 Reading a Range of Items Using Any Valid Range in Response to UnconfirmedEventNotifications of the BUFFER\_READY Event Type

Purpose: To verify that the IUT can initiate a series of ReadRange service requests that access a contiguous set of log records by responding to UnconfirmedEventNotification messages with an 'Event Type' of BUFFER\_READY.

The test concept, configuration requirements, and test steps are identical to those in 8.21.6, except UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests, and the IUT does not acknowledge receiving the notifications.

#### 8.21.8 Reading a Range of Items Using Any Valid Range

Purpose: To verify that the IUT can initiate one or more ReadRange requests that access a tester-specified portion of log records, using any valid range.

Test Concept: The TD contains a Trend Log object that has a logical set of log records, S1. The tester selects a portion of S1 to be returned, and causes the IUT to request those records, using any valid range. The test then verifies that the IUT can display the records in a manner consistent with those that the TD returns.

Configuration Requirements: The TD contains a log object, L1, which has a set of records, S1. The IUT is configured to display the returned set of log records.

Test Steps:

1. MAKE (L1 contain the set of records S1)
2. MAKE (the IUT request a range of samples from L1)
3. WHILE (not all records from the tester-selected range have been requested)
  - RECEIVE ReadRange-Request,
    - 'Object Identifier' = (L1),
    - 'Property Identifier' = (Log\_Buffer),
    - 'Range' = (any valid range)
  - TRANSMIT ReadRange-Ack,
    - 'Object Identifier' = (L1),
    - 'Property Identifier' = (Log\_Buffer),
    - 'ItemData' = (a set of records appropriate for this response)
4. CHECK (the records returned in step 3 include the tester-selected range)
5. MAKE (the IUT display the tester-selected range)
6. CHECK (the records in step 5 are consistent with the records returned in step 3)

Note to Tester: The IUT is not required to display records containing log-status values.

#### 8.21.9 Presents Log Records

Purpose: To verify that the IUT can initiate one or more ReadRange requests that access and present a tester-specified portion of log records. It is a generic test used to test data presentation requirements.

Test Concept: Run test in Clause 8.21.8 and verify that the data presentation meets the criteria specified by the BIBB being tested.

Notes to Tester: The values presented by the IUT may differ from the values transmitted on the wire due to rounding, truncation, formatting, language, conversion, etc.

Notes to Tester: The IUT is not required to display records containing log-status values.

### 8.22 WriteProperty Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating WriteProperty service requests.

#### 8.22.1 Writing Non-Array Properties

Purpose: To verify that the IUT can initiate WriteProperty service requests that do not contain the 'Property Array Index' parameter.

Test Steps:

1. RECEIVE WriteProperty-Request,
  - 'Object Identifier' = (any valid object identifier),
  - 'Property Identifier' = (any valid non-array property of the specified object),
  - 'Property Value' = (any value appropriate to the specified property)

Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

#### 8.22.2 Writing Array Properties

Purpose: To verify that the IUT can initiate WriteProperty service requests that reference a specific element of an array property.

Test Steps:

1. RECEIVE WriteProperty-Request,
  - 'Object Identifier' = (any valid object identifier),



'Property Identifier' = (any valid array property of the specified object),  
 'Priority Array Index' = (any valid ARRAY INDEX for the specified property),  
 'Property Value' = (any value appropriate to the specified property)

Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

### 8.22.3 Writing Commandable Properties

Purpose: To verify that the IUT can initiate WriteProperty service requests that convey a write priority.

Test Steps:

1. RECEIVE WriteProperty-Request,  
     'Object Identifier' = (any valid object identifier for an object with a commandable property),  
     'Property Identifier' = (the commandable property of the specified object),  
     'Property Value' = (any value appropriate to the specified property),  
     'Priority' = (any unsigned value X:  $1 \leq X \leq 16$ )

### 8.22.4 Accepting Input and Modifying Properties

Purpose: This test case verifies that the IUT is capable of accepting user input and using it to modify properties. It is a generic test used to test data-input requirements.

Configuration: For this test, the tester shall choose a property, P1, from an object, O1. The TD shall be configured to not support execution of WritePropertyMultiple.

Test Steps:

1. MAKE (the IUT accept a new value for P1 from the user)
2. RECEIVE WriteProperty-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1  
     'Property Value' = (the value provided to the IUT for P1)
3. TRANSMIT BACnet-SimpleACK-PDU

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the property being modified is an array element, the IUT may include an Array Index parameter in the WriteProperty-Request in step 2.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of WritePropertyMultiple, the IUT may initiate a WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using WriteProperty upon receipt of the correct BACnetReject-PDU from the TD, indicating that WritePropertyMultiple is not supported.

### 8.22.5 Accepting Input and Commanding/Relinquishing Properties

Purpose: This test case verifies that the IUT is capable of accepting user input and using it to modify a commandable property at a specific priority. It also tests that the IUT is capable of relinquishing at that same priority.

Configuration: For this test, the tester shall choose a commandable property, P1, from an object, O1. The TD shall be configured to not support execution of WritePropertyMultiple. PR1 is the specific priority that will be tested.

Test Steps:

1. MAKE (the IUT accept a new value for P1 from the user, to be commanded at priority PR1)
2. RECEIVE WriteProperty-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Priority' = PR1,  
     'Property Value' = (the value provided to the IUT for P1)

3. TRANSMIT BACnet-SimpleACK-PDU
4. MAKE (the IUT relinquish P1 at priority PR1)
5. RECEIVE WriteProperty-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Priority' = PR1,
  - 'Property Value' = NULL
6. TRANSMIT BACnet-SimpleACK-PDU

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of WritePropertyMultiple, the IUT may initiate a WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using WriteProperty upon receipt of the correct BACnetReject-PDU from the TD, indicating that WritePropertyMultiple is not supported.

#### 8.22.6 Writing An Array Size

Purpose: To verify that the IUT can initiate WriteProperty service requests that modify the size of an array.

Test Steps:

1. RECEIVE WriteProperty-Request,
  - 'Object Identifier' = (any valid object identifier),
  - 'Property Identifier' = (any valid array property of the specified object),
  - 'Priority Array Index' = 0
  - 'Property Value' = (any unsigned value)

Note to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

#### 8.22.7 Accepting Input and Modifying Large List Properties

Purpose: This test case verifies that the IUT is capable of accepting user input and using it to modify large list properties where AddListElement and RemoveListElement will be required. It is a generic test used to test data-input requirements.

Configuration: For this test, the tester shall choose a list property, P1, from an object, O1. The TD shall be configured to not support segmentation. The list property shall be configured with a value such that it cannot be read or written without the use of ReadRange and AddListElement/RemoveListElement.

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion, etc.

Notes to Tester: If the IUT has not already determined that the value cannot be transmitted using WriteProperty or WritePropertyMultiple, the IUT may initiate a WriteProperty or WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using AddListElement and RemoveListElement upon receipt of the correct BACnetReject-PDU from the TD, indicating that the write request is too large.

Test Steps:

-- test adding elements into the list

1. MAKE (the IUT accept 1 or more new entries, {E1..En} for P1 from the user)
2. RECEIVE AddListElement-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1
  - 'List Of Elements' = (E1, ..., En)
3. TRANSMIT BACnet-SimpleACK-PDU

-- test removing elements from the list

4. MAKE (the IUT delete 1 or more entries, {E1..Ex} for P1 from the user)

5. RECEIVE RemoveListElement-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1  
     'List Of Elements' = (E1, ..., Ex)
6. TRANSMIT BACnet-SimpleACK-PDU

#### 8.22.8 Writing Array Properties as a Whole Array

Purpose: This test verifies that the IUT is writing the entire array to the TD without the use of the array index.

Configuration Requirements: For this test, the tester shall choose a property, P1, from an object, O1. The TD shall be configured to not support execution of WritePropertyMultiple.

The WriteProperty request initiated by IUT shall contain array of elements in P1, which shall fit in the APDU and segment limitations of the IUT.

Notes to Tester: The value accepted by the IUT may differ from the value transmitted on the wire due to rounding, truncation, formatting, language conversion etc.

Notes to Tester: If the IUT has not already determined that the TD does not support execution of WritePropertyMultiple, the IUT may initiate a WritePropertyMultiple. If this occurs, the IUT shall pass the test only if it automatically falls back to using WriteProperty upon receipt of the correct BACnetReject-PDU from the TD, indicating that WritePropertyMultiple is not supported.

Notes to Tester: Any WriteProperty request generated by the IUT may have a Priority parameter. If included, it shall be in the range 1-16, excluding 6.

Test Steps:

1. MAKE (the IUT accept a new value for P1 including all elements of the array from the user)
2. RECEIVE WriteProperty-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (the value provided to the IUT for P1)
3. TRANSMIT BACnet-SimpleACK-PDU

#### 8.23 WritePropertyMultiple Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating WritePropertyMultiple service requests. At least one of the combinations defined in 8.23.2 through 8.23.4 needs to be supported in order to claim the ability to initiate the WritePropertyMultiple service.

##### 8.23.1 Writing a Single Property of a Single Object

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing a single property of a single object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing a single property of a single object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,  
     'Object Identifier' = (any valid object identifier),  
     'Property Identifier' = (any valid property of the specified object),  
     'Property Value' = (any value appropriate to the specified property)

Note to Tester: Any WritePropertyMultiple request generated by the IUT may have a Priority parameter for any of the properties written. If included, the Priority parameter shall be in the range 1-16, excluding 6.

Notes to Tester: This test may also be used to test an IUT's ability to write a single array element by checking for the inclusion of the 'Priority Array Index' parameter.

### 8.23.2 Writing Multiple Properties of a Single Object

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple properties of a single object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple properties of a single object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,
  - 'Object Identifier' = (any valid object identifier),
  - 'Property Identifier' = (any valid non-array property of the specified object),
  - 'Property Value' = (any value appropriate to the specified property),
  - 'Property Identifier' = (any valid property of the specified object that was not previously used),
  - 'Property Value' = (any value appropriate to the specified property)
- ... (Any number of properties  $\geq 2$  is acceptable.)

Note to Tester: Any WritePropertyMultiple request generated by the IUT may have a Priority parameter for any of the properties written. If included, the Priority parameter shall be in the range 1-16, excluding 6.

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Priority Array Index' parameter for one or more of the properties.

### 8.23.3 Writing Multiple Objects, One Property Each

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple objects and a single property for each object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple objects and a single property for each object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,
  - 'Object Identifier' = (any valid object identifier),
  - 'Property Identifier' = (any valid non-array property of the specified object),
  - 'Property Value' = (any value appropriate to the specified property),
  - 'Object Identifier' = (any valid object identifier not previously used),
  - 'Property Identifier' = (any valid property of the specified object),
  - 'Property Value' = (any value appropriate to the specified property)
- ... (Any number of (object, property, value) tuples  $\geq 2$  is acceptable.)

Note to Tester: Any WritePropertyMultiple request generated by the IUT may have a Priority parameter for any of the properties written. If included, the Priority parameter shall be in the range 1-16, excluding 6.

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Priority Array Index' parameter for one or more of the properties.

### 8.23.4 Writing Multiple Objects, Multiple Properties for Each

Purpose: To verify that the IUT can correctly initiate a WritePropertyMultiple service request containing multiple objects and multiple properties for each object. If the IUT cannot be configured to initiate a WritePropertyMultiple service request containing multiple objects and multiple properties for each object, then this test shall be omitted.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,
  - 'Object Identifier' = (any valid object identifier),
  - 'Property Identifier' = (any valid non-array property of the specified object),
  - 'Property Value' = (any value appropriate to the specified property),
  - 'Property Identifier' = (any valid property of the specified object that was not previously used),
  - 'Property Value' = (any value appropriate to the specified property),
- ... (Additional properties may be included here.)
- 'Object Identifier' = (any valid object identifier not previously used),
- 'Property Identifier' = (any valid non-array property of the specified object),

'Property Value' = (any value appropriate to the specified property),  
 'Property Identifier' = (any valid property of the specified object that was not previously used),  
 'Property Value' = (any value appropriate to the specified property)  
 -- ... (Additional properties may be included here.)  
 -- ... (Additional objects and properties may be included here.)

Note to Tester: Any WritePropertyMultiple request generated by the IUT may have a Priority parameter for any of the properties written. If included, the Priority parameter shall be in the range 1-16, excluding 6.

Notes to Tester: This test may also be used to test an IUT's ability to write one or more array elements by checking for the inclusion of the 'Priority Array Index' parameter for one or more of the properties.

### 8.23.5 Writing Array Properties

Purpose: To verify that the IUT can initiate WritePropertyMultiple service requests that reference a specific element of an array property.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,  
 'Object Identifier' = (any valid object identifier),  
 'Property Identifier' = (any valid array property of the specified object),  
 'Priority Array Index' = (any valid ARRAY INDEX for the specified property),  
 'Property Value' = (any value appropriate to the specified property),  
 'Property Identifier' = (any valid non-array property of the specified object),  
 'Property Value' = (any value appropriate to the specified property)  
 -- ... (Including additional properties, with or without array indexes, is acceptable.)

Notes to Tester: Only one of the properties being written needs to be a specific element of an array, but writing to multiple specific elements as individual operations is acceptable.

Note to Tester: Any WritePropertyMultiple request generated by the IUT may have a Priority parameter for any of the properties written. If included, the Priority parameter shall be in the range 1-16, excluding 6.

### 8.23.6 Writing Commandable Properties

Purpose: To verify that the IUT can initiate WritePropertyMultiple service requests that convey a write priority.

Test Steps:

1. RECEIVE WritePropertyMultiple-Request,  
 'Object Identifier' = (any valid object identifier for an object with a commandable property),  
 'Property Identifier' = (the commandable property of the specified object),  
 'Property Value' = (any value appropriate to the specified property),  
 'Priority' = (any unsigned value X: 1 ≤ X ≤ 16),  
 'Property Identifier' = (any valid non-array property of the specified object),  
 'Property Value' = (any value appropriate to the specified property)

Notes to Tester: Only one of the properties being written needs to be commandable, but writing to multiple commandable properties is acceptable.

## 8.24 DeviceCommunicationControl Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating DeviceCommunicationControl service requests.

### 8.24.1 Deleted Clause

This test has been removed.

### 8.24.2 Indefinite Duration, Disable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE,  
    'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.24.3 Time Duration, Disable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Time Duration' = (any unsigned value > 0),  
    'Enable/Disable' = DISABLE,  
    'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.24.4 Enable, Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Enable/Disable' = ENABLE,  
    'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.24.5 Enable, No Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should resume and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Enable/Disable' = ENABLE,
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.24.6 Time Duration, Disable, No Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and do not convey a password. If the IUT does not support the “no password” option, this test shall not be performed.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
    'Time Duration' = (any unsigned value > 0),  
    'Enable/Disable' = DISABLE
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.24.7 Deleted Clause

This test has been removed.

### 8.25 ConfirmedPrivateTransfer Service Initiation Test

Purpose: To verify that the IUT can initiate the ConfirmedPrivateTransfer Service.



Test Concept: Since the private transfer services by definition convey non-standard service requests, the service parameters and service procedure is not tested. The test simply verifies that the parameters required by BACnet are correctly conveyed.

Test Steps:

1. RECEIVE ConfirmedPrivateTransfer-Request,  
'Vendor ID' = (any valid vendor identifier),  
'Service Number' = (any unsigned value)

#### 8.26 UnconfirmedPrivateTransfer Service Initiation Test

Purpose: To verify that the IUT can initiate the UnconfirmedPrivateTransfer Service.

Test Concept: Since the private transfer services by definition convey non-standard service requests, the service parameters and service procedure is not tested. The test simply verifies that the parameters required by BACnet are correctly conveyed.

Test Steps:

1. RECEIVE UnconfirmedPrivateTransfer-Request,  
'Vendor ID' = (any valid vendor identifier),  
'Service Number' = (any unsigned value)

#### 8.27 ReinitializeDevice Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ReinitializeDevice service requests.

##### 8.27.1 COLDSTART with no Password

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a COLDSTART should be performed and do not convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART
2. TRANSMIT BACnet-SimpleACK-PDU

##### 8.27.2 COLDSTART with a Password

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a COLDSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,  
'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

##### 8.27.3 WARMSTART with no Password

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a WARMSTART should be performed and do not convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART
2. TRANSMIT BACnet-SimpleACK-PDU

##### 8.27.4 WARMSTART with a Password

Purpose: To verify that the IUT can initiate ReinitializeDevice service requests that indicate a WARMSTART should be performed and convey a password.

Test Steps:

1. RECEIVE ReinitializeDevice-Request,  
    'Reinitialized State of Device' = WARMSTART,  
    'Password' = (a password of up to 20 characters)
2. TRANSMIT BACnet-SimpleACK-PDU

### 8.28 ConfirmedTextMessage Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ConfirmedTextMessage service requests.

#### 8.28.1 Text Message with no Message Class

Purpose: To verify that the IUT can initiate ConfirmedTextMessage service requests that do not contain a 'Message Class' parameter.

Test Steps:

1. RECEIVE ConfirmedTextMessage-Request,  
    'Text Message Source Device' = IUT,  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.28.2 Text Message with an Unsigned Message Class

Purpose: To verify that the IUT can initiate ConfirmedTextMessage service requests that contain an Unsigned 'Message Class' parameter.

Test Steps:

1. RECEIVE ConfirmedTextMessage-Request,  
    'Text Message Source Device' = IUT,  
    'Message Class' = (any Unsigned value),  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.28.3 Text Message with a CharacterString Message Class

Purpose: To verify that the IUT can initiate ConfirmedTextMessage service requests that contain a CharacterString 'Message Class' parameter.

Test Steps:

1. RECEIVE ConfirmedTextMessage-Request,  
    'Text Message Source Device' = IUT,  
    'Message Class' = (any CharacterString value),  
    'Message Priority' = NORMAL,  
    'Message' = (any CharacterString)
2. TRANSMIT BACnet-SimpleACK-PDU

#### 8.28.4 Text Message with an Urgent Priority

Purpose: To verify that the IUT can initiate ConfirmedTextMessage service requests that convey an urgent priority.

Test Steps:

1. RECEIVE ConfirmedTextMessage-Request,  
    'Text Message Source Device' = IUT,  
    'Message Priority' = URGENT,  
    'Message' = (any CharacterString)
2. TRANSMIT BACnet-SimpleACK-PDU

**8.29 UnconfirmedTextMessage Service Initiation Tests**

This clause defines the tests necessary to demonstrate support for initiating UnconfirmedTextMessage service requests.

**8.29.1 Text Message with no Message Class**

Purpose: To verify that the IUT can initiate UnconfirmedTextMessage service requests that do not contain a 'Message Class' parameter.

Test Steps:

1. RECEIVE UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = IUT,  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)

**8.29.2 Text Message with an Unsigned Message Class**

Purpose: To verify that the IUT can initiate UnconfirmedTextMessage service requests that contain an Unsigned 'Message Class' parameter.

Test Steps:

1. RECEIVE UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = IUT,  
     'Message Class' = (any Unsigned value),  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)

**8.29.3 Text Message with a CharacterString Message Class**

Purpose: To verify that the IUT can initiate UnconfirmedTextMessage service requests that contain a CharacterString 'Message Class' parameter.

Test Steps:

1. RECEIVE UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = IUT,  
     'Message Class' = (any CharacterString value),  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)

**8.29.4 Text Message with an Urgent Priority**

Purpose: To verify that the IUT can initiate UnconfirmedTextMessage service requests that convey an urgent priority.

Test Steps:

1. RECEIVE UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = IUT,  
     'Message Priority' = URGENT,  
     'Message' = (any CharacterString)

**8.30 TimeSynchronization Service Initiation Tests**

Purpose: To verify that the IUT can initiate TimeSynchronization service requests.

Test Steps:

1. RECEIVE TimeSynchronization-Request,  
     'Time' = (the current local date and time)

**8.31 UTCTimeSynchronization Service Initiation Tests**

Purpose: To verify that the IUT can initiate UTCTimeSynchronization service requests.

Test Steps:

1. RECEIVE UTCTimeSynchronization-Request,  
'Time' = (the current UTC date and time)

### 8.32 Who-Has Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating Who-Has service requests. At least one of the forms of the service defined in 8.32.1 through 8.32.4 needs to be supported in order to claim the ability to initiate the Who-Has service.

#### 8.32.1 Object Identifier Selection with no Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with no device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Object Identifier' = Object1
2. TRANSMIT  
DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
SOURCE = TD,  
I-Have-Request,  
'Device Identifier' = (the TD's Device object)  
'Object Identifier' = Object1
3. CHECK (for any vendor-defined observable actions)

#### 8.32.2 Object Name Selection with no Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with no device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Notes to Tester: If there is no vendor-defined observable action, then test step 3 can be skipped.

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Object Name' = V1
2. TRANSMIT  
DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
SOURCE = TD,  
I-Have-Request,  
'Device Identifier' = (the TD's Device object)  
'Object Name' = V1
3. CHECK (for any vendor-defined observable actions)

#### 8.32.3 Object Identifier Selection with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object identifier form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.

Test Steps:

1. RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
 SOURCE = IUT,  
 Who-Has-Request,  
 'Device Instance Range Low Limit' = (any integer X:  $0 \leq X \leq$  'Device Instance Range High Limit'),  
 'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq$  4,194,303),  
 'Object Identifier' = Object1
2. TRANSMIT  
 DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
 SOURCE = TD,  
 I-Have-Request,  
 'Device Identifier' = (the TD's Device object)  
 'Object Identifier' = Object1
3. CHECK (for any vendor-defined observable actions)

#### 8.32.4 Object Name Selection with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Notes to Tester: Device instance range should be selected to cover TD's device object identifier. If there is no vendor-defined observable action, then test step 3 can be skipped.

Test Steps:

1. RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
 SOURCE = IUT,  
 Who-Has-Request,  
 'Device Instance Range Low Limit' = (any integer X:  $0 \leq X \leq$  'Device Instance Range High Limit'),  
 'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq$  4,194,303),  
 'Object Name' = V1
2. TRANSMIT  
 DESTINATION = IUT | LOCAL BROADCAST | GLOBAL BROADCAST  
 SOURCE = TD,  
 I-Have-Request,  
 'Device Identifier' = (the TD's Device object)  
 'Object Name' = V1
3. CHECK (for any vendor-defined observable actions)

#### 8.33 I-Have Service Initiation Tests

Verification of the ability to initiate I-Have service requests is covered by the Who-Has service execution tests in 9.32.

#### 8.34 Who-Is Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating Who-Is service requests. At least one of the forms of the service defined in 8.34.1 and 8.34.2 needs to be supported in order to claim the ability to initiate the Who-Is service.

##### 8.34.1 Who-Is Request with no Device Instance Range

Purpose: To verify that the IUT can initiate Who-Is service requests with no device instance range. If the IUT cannot be caused to issue a Who-Is request of this form, then this test shall be omitted.

Test Steps:

1. RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
 SOURCE = IUT,  
 Who-Is-Request

**8.34.2 Who-Is Request with a Device Instance Range**

Purpose: To verify that the IUT can initiate Who-Is service requests with a device instance range. If the IUT cannot be caused to issue a Who-Is request of this form, then this test shall be omitted.

Test Steps:

1. RECEIVE  
     DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
     SOURCE = IUT,  
     Who-Is-Request,  
     'Device Instance Range Low Limit' = (any integer X:  $0 \leq X \leq$  'Device Instance Range High Limit'),  
     'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit'  $\leq Y \leq$  4,194,303)

**8.34.3 Who-Is Request with no Device Instance Range**

Purpose: To verify that the IUT can initiate unicast Who-Is service requests with no device instance range. If the IUT cannot be caused to issue a Who-Is request of this form, then this test shall be omitted.

Test Steps:

1. RECEIVE  
     DESTINATION = TD,  
     SOURCE = IUT,  
     Who-Is-Request

**8.35 I-Am Service Initiation Tests**

Verification of the ability to initiate I-Am service requests is covered by the Who-Is service execution tests in 9.30.

**8.36 VT-Open Service Initiation Tests**

This clause defines the tests necessary to demonstrate support for initiating VT-Open service requests.

**8.36.1 Default Terminal VT-class**

Purpose: To verify that the IUT can initiate VT-Open service requests for the DEFAULT\_TERMINAL VT-class and that the open VT session is reflected in the Active\_VT\_Sessions property of the IUT's Device object.

Configuration Requirements: The IUT shall be configured so that there are no active VT sessions.

Test Steps:

1. RECEIVE VT-Open-Request,  
     'VT-class' = DEFAULT\_TERMINAL,  
     'Local VT Session Identifier' = (any valid session identifier,  $S_{IUT}$ )
2. TRANSMIT VT-Open-ACK,  
     'Remote VT Session Identifier' = (any valid session identifier,  $S_{TD}$ )
3. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (the newly established session)

If the IUT is capable of having more than one active VT session the tester may optionally decide to leave the session created by this test active while proceeding with the tests in 8.36.2. Otherwise the VT session should be closed. This may be accomplished as part of the test for verifying the initiation of the VT-Close service request (8.37) or execution of the VT-Close service (9.35.1).

**8.36.2 Other VT-classes**

Purpose: To verify that the IUT can initiate VT-Open service requests for all supported optional VT-classes and that the open VT sessions are reflected in the Active\_VT\_Sessions property of the IUT's Device object. If the IUT does not support VT classes other than DEFAULT\_TERMINAL then this test shall be omitted.

Test Concept: An attempt is made to open a new VT session for each supported VT-class. The previously opened sessions are left open until all supported VT-classes are tested or until the maximum number of open sessions supported is reached.



If the maximum number of open sessions is reached before all of the VT-Classes have been tested, a session is closed before moving on to the next VT-class. Either the IUT or the TD can initiate the closure of a session.

Test Steps:

1. REPEAT X = (all supported VT classes except DEFAULT\_TERMINAL) DO {
  - RECEIVE VT-Open-Request,
    - 'VT-class' = X,
    - 'Local VT Session Identifier' = (any valid unique session identifier, S<sub>IUT,i</sub>)
  - TRANSMIT VT-Open-ACK,
    - 'Remote VT Session Identifier' = (any valid unique session identifier, S<sub>TD,i</sub>)
  - VERIFY (the IUT's Device object),
    - Active\_VT\_Sessions = (a list including the newly established session and any sessions previously established as part of this test but not yet closed)
  - IF (the IUT cannot support additional active VT sessions) THEN
    - (TRANSMIT VT-Close-Request,
      - 'List of Remote VT Session Identifiers' = (the IUT's session identifier for the most recently opened session)
    - RECEIVE BACnet-SimpleACK-PDU
      - )
      - |
      - (RECEIVE VT-Close-Request,
        - 'List of Remote VT Session Identifiers' = (the TD's session identifier for the most recently opened session)
      - TRANSMIT BACnet-SimpleACK-PDU
        - )

### 8.37 VT-Close Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating VT-Close service requests. The IUT shall pass the test defined in 8.37.1. If the IUT supports more than one active VT session it shall also pass the tests defined in 8.37.2 and 8.37.3.

#### 8.37.1 Closing a Single Open VT Session

Purpose: To verify that the IUT can initiate a VT-Close service request conveying the session identifier for a single open VT session and that, when closed, the session is removed from the Active\_VT\_Sessions property of the IUT's Device object.

Test Concept: At the beginning of the test the IUT has one active VT session. The IUT attempts to close that active session by initiating a VT-Close service request. The TD acknowledges the VT-Close request and then reads the Active\_VT\_Sessions property to verify that it has been correctly updated.

Configuration Requirements: The IUT shall be configured with one active VT session. The IUT will be in this state if it has just completed test 8.36.1 and the session was not subsequently closed.

Test Steps:

1. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (a single valid session)
2. RECEIVE VT-Close-Request,
  - 'List of Remote VT Session Identifiers' = (a single session identifier, S<sub>TD</sub>, appropriate for the TD's view of the open session)
3. TRANSMIT BACnet-SimpleACK-PDU
4. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (an empty list)

#### 8.37.2 Closing One of Multiple Open VT Sessions

Purpose: To verify that the IUT can initiate a VT-Close service request conveying one session identifier from multiple open VT sessions and that, when closed, the session is removed from the Active\_VT\_Sessions property of the IUT's Device object.

**Test Concept:** At the beginning of the test the IUT has multiple active VT sessions. The IUT attempts to close one active session by initiating a VT-Close service request. The TD acknowledges the VT-Close request and then reads the Active\_VT\_Sessions property to verify that the intended session was closed but the others still remain.

**Configuration Requirements:** The IUT shall be configured with multiple active VT sessions. The IUT will be in this state if it has just completed test 8.36.2 and the open sessions were not subsequently closed.

**Test Steps:**

1. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (a list of two or more valid sessions)
2. RECEIVE VT-Close-Request,  
'List of Remote VT Session Identifiers' = (a single session identifier, S<sub>TD</sub>, appropriate for the TD's view of one of the open sessions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (the same list as in step 1 except that the closed session is no longer present)

### 8.37.3 Closing Multiple Open VT Sessions

**Purpose:** To verify that the IUT can initiate a VT-Close service request conveying more than one session identifier from multiple open VT sessions and that, when closed, the sessions are removed from the Active\_VT\_Sessions property of the IUT's Device object.

**Test Concept:** At the beginning of the test the IUT has multiple active VT sessions. The IUT attempts to close more than one of these active sessions by initiating a single VT-Close service request. The TD acknowledges the VT-Close request and then reads the Active\_VT\_Sessions property to verify that the intended sessions were closed but the others still remain.

**Configuration Requirements:** The IUT shall be configured with multiple active VT sessions. The IUT will be in this state if it has just completed test 8.36.2 and the open sessions were not subsequently closed.

**Test Steps:**

1. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (a list of two or more valid sessions)
2. RECEIVE VT-Close-Request,  
'List of Remote VT Session Identifiers' = (two or more session identifiers, S<sub>TD,i</sub> and S<sub>TD,j</sub>, appropriate for the TD's view of the open sessions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. VERIFY (the IUT's Device object), Active\_VT\_Sessions = (the same list as in step 1 except that the closed sessions are no longer present)

## 8.38 VT-Data Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating VT-Data service requests. The means shall be provided to cause the IUT to transmit a VT data stream sufficiently long that it can be used to verify proper sequencing of the 'VT-Data Flag'. The details of how this is to be done are a local matter. If the IUT is the virtual operator interface side of the virtual terminal session then test 8.38.1 applies. Otherwise, test 8.38.2 applies.

### 8.38.1 Virtual Operator Interface

**Purpose:** To verify that the IUT initializes the 'VT-data Flag' to zero and alternates the value between zero and one with each new VT-Data request for this session. It also verifies that the IUT correctly sequences the data if only a portion of the data is accepted.

**Test Concept:** A virtual terminal session is established with the TD. The session needs to be long enough to verify the sequencing of the VT-data Flag. At one point in the session the TD will accept only a portion of the data in order to verify that the IUT correctly resequences the data for the next transmission. When all of this is completed the TD will terminate the session.

Test Steps:

1. RECEIVE VT-Open-Request,  
     'VT-class' = DEFAULT\_TERMINAL,  
     'Local VT Session Identifier' = (any valid session identifier, S<sub>IUT</sub>)
2. TRANSMIT VT-Open-ACK,  
     'Remote VT Session Identifier' = (any valid session identifier, S<sub>TD</sub>)
3. MAKE (the IUT initiate a VT-Data-Request)
4. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (any valid DEFAULT\_TERMINAL data more than one character in length),  
     'VT-data Flag' = 0
5. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = FALSE,  
     'Accepted Octet Count' = 1
6. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (a continuation of the DEFAULT\_TERMINAL data beginning with the  
     second character in step 4),  
     'VT-data Flag' = 1
7. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = TRUE
8. MAKE (the IUT initiate a VT-Data-Request)
9. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (any valid DEFAULT\_TERMINAL data),  
     'VT-data Flag' = 0
10. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = TRUE
11. TRANSMIT VT-Close-Request,  
     'List of VT Session Identifiers' = S<sub>IUT</sub>
12. RECEIVE BACnet-SimpleACK-PDU

### 8.38.2 Virtual Terminal

Purpose: To verify that the IUT initializes the 'VT-data Flag' to zero and alternates the value between zero and one with each new VT-Data request for this session. It also verifies that the IUT correctly sequences the data if only a portion of the data is accepted.

Test Concept: A virtual terminal session is established with the TD. The session needs to be long enough to verify the sequencing of the VT-data Flag. At one point in the session the TD will accept only a portion of the data in order to verify that the IUT correctly resequences the data for the next transmission. When all of this is completed the TD will terminate the session.

Test Steps:

1. TRANSMIT VT-Open-Request,  
     'VT-class' = DEFAULT\_TERMINAL,  
     'Local VT Session Identifier' = (any valid session identifier, S<sub>TD</sub>)
2. RECEIVE VT-Open-ACK,  
     'Remote VT Session Identifier' = (any valid session identifier, S<sub>IUT</sub>)
3. TRANSMIT VT-Data-Request,  
     'VT-session Identifier' = S<sub>IUT</sub>,  
     'VT-new Data' = (any data that will trigger the IUT to transfer a VT data stream),  
     'VT-data Flag' = 0
4. RECEIVE VT-Data-ACK,  
     'All new Data Accepted' = TRUE
5. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (any valid DEFAULT\_TERMINAL data more than one character in length),

- 'VT-data Flag' = 0
6. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = FALSE,  
     'Accepted Octet Count' = 1
7. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (a continuation of the DEFAULT\_TERMINAL data beginning with the second character in step 5),  
     'VT-data Flag' = 1
8. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = TRUE
9. MAKE (the IUT initiate a VT-Data-Request)
10. RECEIVE VT-Data-Request,  
     'VT-session Identifier' = S<sub>TD</sub>,  
     'VT-new Data' = (any valid DEFAULT\_TERMINAL data),  
     'VT-data Flag' = 0
11. TRANSMIT VT-Data-ACK,  
     'All new Data Accepted' = TRUE
12. TRANSMIT VT-Close-Request,  
     'List of VT Session Identifiers' = S<sub>IUT</sub>
13. RECEIVE BACnet-SimpleACK-PDU

### 8.39 RequestKey Service Initiation Tests

This clause defines the tests necessary to demonstrate the ability to initiate RequestKey service requests.

The means shall be provided to cause the IUT to transmit a RequestKey service to the TD-as-key-server, and to use the session key (SK) thus acquired in communication with another device.

All PDUs in this test which are specified to be enciphered are first padded and then enciphered with the specified key.

Configuration Requirements: The IUT shall be configured with a private session key, PK, known to the TD. The IUT shall also be configured to know the maximum APDU length accepted by the TD, in order that its enciphered APDUs may be padded as required.

#### 8.39.1 Initial Test

Purpose: To verify that the IUT can issue a RequestKey service request to a key server, padded and enciphered with its PK.

Test Steps:

1. RECEIVE RequestKey-Request,  
     'Requesting Device Identifier' = IUT,  
     'Requesting Device Address' = (the BACnetAddress of the IUT),  
     'Remote Device Identifier' = (any device identifier known to the TD key server),  
     'Remote Device Address' = (the BACnetAddress corresponding to the remote device)

Note: The service request portion of this PDU shall be enciphered using PK<sub>IUT</sub>.

#### 8.39.2 Random Padding Test

Purpose: To verify that the IUT issues a RequestKey service request with random padding.

Test Steps: The test step for this test case is identical to the test step in 8.39.1.

Notes to Tester: The passing results for this test case are identical to the test steps in 8.39.1 except that the sequence of octets padding the APDU transmitted by the IUT shall be different from those observed in 8.39.1

### 8.40 Authenticate Service Initiation Tests

This clause defines the tests necessary to demonstrate the ability to initiate Authenticate service requests under the five conditions in which they may be initiated. These are Peer Authentication, Message Execution Authentication, Message

Initiation Authentication, Operator Authentication and Enciphered Session. Not all devices supporting the Authentication Service will support all modes of initiation; only the applicable test shall be performed.

All PDUs in this test that are specified to be enciphered are first padded and then enciphered with the specified key.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key known to the TD.

#### 8.40.1 Peer Authentication

Purpose: To verify the ability of a device to correctly initiate an Authenticate service request to implement peer authentication.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. RECEIVE Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number)
2. TRANSMIT Authenticate-Request-ACK,  
'Modified Random Number' = (the modified pseudo random number)

#### 8.40.2 Message Execution Authentication

Purpose: To verify the ability of a device to correctly initiate an Authenticate service request to implement message execution authentication.

Test Concept: A secure session between the TD and the IUT has been established. The IUT makes a ReadProperty request using the procedures for authenticated service execution.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. RECEIVE Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number),  
'Expected Invoke ID' = (any valid invoke ID)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
2. RECEIVE ReadProperty-Request,  
Invoke ID = (the 'Expected Invoke ID' used in step 1),  
'Object Identifier' = (any valid object identifier),  
'Property Identifier' = (any supported property of the specified object)
3. TRANSMIT Authenticate-Request-ACK,  
'Modified Random Number' = (the modified pseudo random number)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
4. TRANSMIT ReadProperty-ACK,  
'Object Identifier' = (the object identifier used in step 2),  
'Property Identifier' = (the property identifier used in step 2),  
'Property Value' = (any valid value for the specified property)

#### 8.40.3 Message Initiation Authentication

This clause defines the tests necessary to demonstrate support for executing the Authenticate service for the purpose of message initiation authentication.

##### 8.40.3.1 Message Initiation Authentication by a Key-Server

Purpose: To verify the ability to correctly initiate an Authenticate service request to implement message initiation authentication. If the IUT is not a key-server this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key that corresponds to the TD.

Test Steps:

1. TRANSMIT RequestKey-Request,
  - 'Requesting Device Identifier' = IUT,
  - 'Requesting Device Address' = (the BACnetAddress of the IUT),
  - 'Remote Device Identifier' = (any device identifier known to the TD key server),
  - 'Remote Device Address' = (the BACnetAddress corresponding to the remote device)

Note: The service request portion of this PDU shall be enciphered using  $PK_{TD}$ .

2. RECEIVE Authenticate-Request,
    - 'Pseudo Random Number' = (any valid pseudo random number),
- Note: The service request portion of this PDU shall be enciphered using  $PK_{TD}$ .

#### 8.40.3.2 Message Initiation Authentication, Peer-to-Peer

Purpose: To verify the ability to correctly initiate an Authenticate service request to implement message initiation authentication. If the IUT is a key server only this test shall be omitted.

Test Concept: A secure session between the TD and the IUT has been established. The TD initiates an application service request addressed to the ITT. The IUT then follows the procedures for authenticating the source of the request.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. TRANSMIT ReadProperty-Request,
    - 'Object Identifier' = (any object supported by the IUT),
    - 'Property Identifier' = (any property of the specified object)
  2. RECEIVE Authenticate-Request
    - 'Pseudo Random Number' = (any valid pseudo random number),
    - 'Expected Invoke ID' = (the invoke ID used in step 1)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
3. TRANSMIT Authenticate-Request-ACK,
    - 'Modified Random Number' = (the modified pseudo random number)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
4. RECEIVE ReadProperty-ACK,
    - 'Object Identifier' = (the object specified in step 1),
    - 'Property Identifier' = (the property specified in step 1),
    - 'Property Value' = (the value of the specified property as indicated by the EPICS)

#### 8.40.4 Operator Authentication

Purpose: To verify the ability to correctly initiate an Authenticate service request to implement operator authentication. If the IUT is a key server only or does not support operator authentication this test shall be omitted.

Test Concept: The TD performs the function of the key-server, containing the operator password lists. The IUT attempts to authenticate an operator and password.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key.

Test Steps:

1. RECEIVE Authenticate-Request
  - 'Pseudo Random Number' = (any valid pseudo random number),
  - 'Operator Name' = (any CharacterString indicating the name of the operator),
  - 'Operator Password' = (any CharacterString indicating the password)

Note: The service request portion of this PDU shall be enciphered using  $PK_{IUT}$ .

### 8.40.5 Enciphered Session

Purpose: To verify the ability of the IUT to correctly initiate an Authenticate service request to initiate and terminate an enciphered session.

Test Concept: The IUT attempts to initiate an enciphered session with the TD by transmitting an Authenticate request. The TD follows the procedure to authenticate the request and start the session. The IUT then reads a property from the Device object of the TD. The IUT then attempts to end the session by transmitting another Authenticate request. Note that the service request portion of all of the messages in this test are enciphered using  $SK_{TD,IUT}$ .

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. RECEIVE Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Start Enciphered Session' = TRUE
  2. TRANSMIT Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Expected Invoke ID' = (the invoke ID used in step 1)
  3. RECEIVE Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 2)
  4. TRANSMIT Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 1)
- Note: At this point the enciphered session is initiated.
5. MAKE (the IUT read a property from the Device object of the TD)
  6. RECEIVE ReadProperty-Request,  
    'Object Identifier' = (the Device object of the TD),  
    'Property Identifier' = (any supported property)
  7. TRANSMIT ReadProperty-ACK,  
    'Object Identifier' = (the object specified in step 6),  
    'Property Identifier' = (the property specified in step 6),  
    'Property Value' = (the value of the specified property)
  8. RECEIVE Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Start Enciphered Session' = FALSE
  9. TRANSMIT Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Expected Invoke ID' = (the invoke ID used in step 8)
  10. RECEIVE Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 9)
  11. TRANSMIT Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 8)

## 8.41 WriteGroup Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating WriteGroup service requests.

### 8.41.1 Broadcasting to a Group of Channel Objects

Purpose: Verify that the IUT can initiate a WriteGroup request to an arbitrary group with an arbitrary channel number.

Test Concept: Make the IUT perform a WriteGroup request to a tester selected group and channel. Verify that the request is generated.

Test Steps:

1. MAKE (the IUT initiate a WriteGroup request to group G and Channel C)
2. RECEIVE WriteGroup-Request  
    DESTINATION = GLOBAL\_BROADCAST | LOCAL\_BROADCAST |  
    REMOTE\_BROADCAST | TD,



'Group Number' = G,  
 'Write Priority' = (any valid value),  
 'Change List' = (a valid list of 1 or more changes which impact channel C),  
 'Inhibit Delay' = (absent or TRUE or FALSE)

#### 8.42 SubscribeCOVPropertyMultiple Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating SubscribeCOVPropertyMultiple service requests.

##### 8.42.1 Positive SubscribeCOVPropertyMultiple Service Initiation Tests

###### 8.42.1.1 Confirmed Notifications Subscription

Purpose: To verify the client can subscribe to confirmed notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: The IUT is made to subscribe for confirmed notifications.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L,  
     'Max Notification Delay' = (any valid notification delay),  
     'List of COV Subscription Specifications' = (any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleAck-PDU

###### 8.42.1.2 Unconfirmed Notifications Subscription

Purpose: To verify the client can subscribe to unconfirmed notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: The IUT is made to subscribe for unconfirmed notifications.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = L,  
     'Max Notification Delay' = (any valid notification delay),  
     'List of COV Subscription Specifications' = (any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleAck-PDU

###### 8.42.1.3 Requests 8 Hour Lifetimes

Purpose: To verify that the IUT is able to provide a lifetime which is less than or equal to 8 hours for any SubscribeCOVPropertyMultiple request it generates.

Test Concept: The tester selects any of the possible COVM subscriptions that the IUT is able to generate, and it is configured to use a lifetime less than or equal to 8 hours. The IUT is made to send the subscription, and the lifetime is verified to be less than or equal to 8 hours.

Test Steps:

1. MAKE (the IUT send the selected SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (any value <= 28800),  
     'Max Notification Delay' = (any valid delay between 1 and 3600),

'List of COV Subscription Specifications' = (a valid list of COV Specifications)

### 3. TRANSMIT BACnet-SimpleACK-PDU

#### 8.42.1.4 Subscribe to Timestamped Notifications

Purpose: To verify the client can subscribe to timestamped notifications using the SubscribeCOVPropertyMultiple service.

Test Concept: A subscription for timestamped COVM notifications is established with Lifetime L for property P1 of Object O1. L shall be less than 8 hours but large enough to complete the test.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (any valid list with at least 1 entry where 'Timestamped' is TRUE)
3. TRANSMIT BACnet-SimpleAck-PDU

#### 8.42.1.5 Subscribe to Two Properties in a Single Object

Purpose: To verify that the IUT can subscribe to 2 or more properties from a single object.

Test Concept: A subscription for COVM notifications is established for properties from a single object.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (a valid list of 2 or more properties from a single object)
3. TRANSMIT BACnet-SimpleAck-PDU

#### 8.42.1.6 Subscribe to Properties in Multiple Objects Using a Single Request

Purpose: To verify the client can subscribe to properties from multiple objects.

Test Concept: A subscription for notifications is established for properties from 2 or more objects.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (PID: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = L,
  - 'Max Notification Delay' = (any valid notification delay),
  - 'List of COV Subscription Specifications' = (PROPS: any valid list of properties from 2 or more objects)
4. TRANSMIT BACnet-SimpleAck-PDU

#### 8.42.1.7 Change of Value Multiple Notification

Purpose: To verify that the IUT accepts COVM notifications for properties which it subscribed to.

Test Concept: A subscription for COVM notifications is established, a notification is sent to the IUT, and the vendor defined actions are verified.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (ID1: any valid process identifier),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (L: any valid lifetime),  
     'Max Notification Delay' = (any valid delay between 1 and 3600),  
     'List of COV Subscription Specifications' = (PROPS: any valid list of subscriptions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. IF (the subscription was for confirmed notifications) THEN  
     TRANSMIT ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = TD,  
     'Time Remaining' = (any value  $\sim$  L),  
     'Timestamp' = (any valid value, or absent if subscribed to non-timestamped notifications),  
     'List of COV Notifications' = (values appropriate to each entry in PROPS)  
     RECEIVE BACnet-SimpleACK-PDU  
   ELSE  
     TRANSMIT UnconfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = TD,  
     'Time Remaining' = (any value  $\sim$  L),  
     'Timestamp' = (any valid value, or absent if subscribed to non-timestamped notifications),  
     'List of COV Notifications' = (values appropriate to each entry in PROPS)
5. CHECK (verify that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 8.42.1.8 Canceling a Subscription

Purpose: To verify the client can cancel a COVM subscription.

Test Concept: A subscription for COVM notifications is established with a lifetime L, which is long enough to complete the test. The client is made to cancel the subscription by sending a SubscribeCOVPropertyMultiple request with Lifetime, and Max Notification Delay absent.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = L,  
     'Max Notification Delay' = (any valid delay between 1 and 3600),  
     'List of COV Subscription Specifications' = (PROPS: a valid list of COV Subscription Specifications)
3. TRANSMIT BACnet-SimpleAck-PDU
4. IF confirmed notifications were subscribed for THEN  
     TRANSMIT ConfirmedCOVNotificationMultiple-Request  
     'Subscriber Process Identifier' = PID,  
     'Initiating Device Identifier' = (TD's device identifier),  
     'Time Remaining' = (a value  $\sim$  L),  
     'Timestamp' = (a valid value, or absent if Time Of Change was not requested in the subscription)  
     'List of COV Notifications' = (a valid list containing an entry for each entry in PROPS)  
     RECEIVE BACnet-SimpleAck-PDU  
   ELSE  
     TRANSMIT UnconfirmedCOVNotificationMultiple-Request  
     'Subscriber Process Identifier' = PID,  
     'Initiating Device Identifier' = (TD's device identifier),

'Time Remaining' = (a value  $\sim$  L),  
 'Timestamp' = (a valid value, or absent if Time Of Change was not requested  
 in the subscription)  
 'List of COV Notifications' = (a valid list containing an entry for each entry in PROPS)

5. MAKE (the IUT cancel the subscription)
6. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Issue Confirmed Notifications' = (the same value used in step 2),  
     -- 'Lifetime' = (absent)  
     -- 'Max Notification Delay' = (absent)  
     'List of COV Subscription Specifications' = (PROPS, or an empty list)
7. TRANSMIT BACnet-SimpleAck-PDU

#### 8.42.2 Negative SubscribeCOVPropertyMultiple Service Initiation Tests

##### 8.42.2.1 Change of Value Multiple Notification Arrives After Subscription Has Expired

Purpose: To verify that an appropriate error is returned if a COVM notification arrives after the subscription time period has expired.

Test Concept: A subscription for COVM notifications is established and then cancelled or allowed to expire. A ConfirmedCOVNotificationMultiple is then sent to the IUT to verify it returns either the appropriate error or a SimpleACK.

Test Steps:

1. MAKE (the IUT send a SubscribeCOVPropertyMultiple-Request),
2. RECEIVE SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = (ID1: any valid process identifier),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (L: any valid lifetime),  
     'Max Notification Delay' = (any valid notification delay),  
     'List of COV Subscription Specifications' = (PROPS: any valid list of COV subscriptions)
3. TRANSMIT BACnet-SimpleACK-PDU
4. TRANSMIT ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = TD,  
     'Time Remaining' = (a value  $\sim$  L),  
     'Timestamp' = (any appropriate value or absent if it is not a timestamped subscription)  
     'List of COV Notifications' = (values appropriate to the properties in PROPS)
5. RECEIVE BACnet-SimpleACK-PDU
6. IF (the IUT can cancel the subscription) THEN  
     MAKE (the IUT cancel the subscription),  
     RECEIVE SubscribeCOVPropertyMultiple-Request,  
         'Subscriber Process Identifier' = ID1,  
         'Issue Confirmed Notifications' = TRUE,  
         'Lifetime' = (absent)  
         'Max Notification Delay' = (absent)  
         'List of COV Subscription Specifications' = (PROPS or an empty list)  
     ELSE  
         WAIT (2 \* L seconds)
7. TRANSMIT ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = TD,  
     'Time Remaining' = (a value  $\sim$  L),  
     'Timestamp' = (any appropriate value or absent if it is not a timestamped subscription)  
     'List of COV Notifications' = (values appropriate to the properties in PROPS)
8. RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = UNKNOWN\_SUBSCRIPTION |  
     (BACnet-SimpleAck-PDU)

**8.42.2.2 Unknown Subscription**

**Purpose:** To verify that an appropriate response is returned if a COVM notification arrives that contains arguments or parameters which do not match any current subscriptions.

**Test Concept:** The TD sends a ConfirmedCOVNotificationMultiple-Request which does not correspond to any existing subscriptions. Verify that the IUT responds with either an error message or a SimpleACK.

**Configuration Requirements:** At the start of the test, the IUT shall have no outstanding COVM subscriptions with TD using process identifier ID2.

**Test Steps:**

1. TRANSMIT ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID2,  
     'Initiating Device Identifier' = TD,  
     'Time Remaining' = (any valid value),  
     'List of COV Notifications' = (any valid list of property notifications)
2. RECEIVE  
     (BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = (UNKNOWN\_SUBSCRIPTION)) |  
     (BACnet-SimpleACK-PDU)

**8.43 AuditLogQuery Initiation Tests**

This clause defines the tests necessary to demonstrate support for initiating AuditLogQuery service requests.

**8.43.1 Reading a Range of Items Using Any Valid Query**

**Purpose:** To verify that the IUT can initiate one or more AuditLogQuery requests that access a tester-specified portion of an audit log, using any valid range.

**Test Concept:** The TD contains an Audit Log object that has a logical set of log records, S1. The tester selects a portion of S1 to be returned and causes the IUT to request those records using any valid range. The test then verifies that the IUT can display the records in a manner consistent with those that the TD returns.

**Configuration Requirements:** The TD contains an audit log object, L1, which has a set of records, S1. The IUT is configured to display the returned set of log records.

**Test Steps:**

1. MAKE (L1 contain the set of records S1)
2. MAKE (the IUT request a range of samples from L1)
3. WHILE (not all records from the tester-selected range have been returned) {  
     RECEIVE AuditLogQuery-Request,  
     'Audit Log' = (L1),  
     'Query Parameters' = (any valid query),  
     'Start at Sequence Number' = (an appropriate sequence number) -- or absent  
     'Requested Count' = (any valid range)  
     TRANSMIT AuditLogQuery-ACK,  
     'AuditLog' = (L1),  
     'Records' = (a set of records appropriate for this response),  
     'NoMoreItems' = (an appropriate value for this response)  
   }
4. CHECK (the records returned in step 3 include the tester-selected range)
5. MAKE (the IUT display the tester-selected range)
6. CHECK (the records displayed in step 5 are consistent with the records returned in step 3)

## 9. APPLICATION SERVICE EXECUTION TESTS

The test cases defined in this clause shall be used to verify that a BACnet device correctly implements the service procedure for the specified application service. BACnet devices shall be tested for the proper execution of each application service for which the PICS indicates execution is supported.

For each application service included in this clause several test cases are defined that collectively test the various options and features defined for the service in the BACnet standard. A test case is a sequence of one or more messages that are exchanged between the implementation under test (IUT) and the testing device (TD) in order to determine if a particular option or feature is correctly implemented. Multiple test cases that have a similar or related purpose are collected into test groups.

Under some circumstances an IUT may be unable to demonstrate conformance to a particular test case because the test applies to a feature that requires a particular BACnet object or optional property that is not supported in the IUT. For example, a device may support the File Access services but restrict files to stream access only. Such a device would have no way to demonstrate that it could implement the record access features of the File Access services. When this type of situation occurs the IUT shall be considered to be in conformance with BACnet provided the PICS documentation clearly indicates the restriction. Failure to document the restriction shall constitute nonconformance to the BACnet standard. All features and optional parameters for BACnet application services shall be supported unless a conflict arises because of unsupported objects or unsupported optional properties.

For each application service the tests are divided into two types, positive tests and negative tests. The positive tests verify that the IUT can correctly handle cases where the service is expected to be successfully completed. The negative tests verify correct handling for various error cases that may occur. Negative tests include inappropriate service parameters but they do not include cases with encoding errors or otherwise malformed PDUs. Tests to ensure that the IUT can handle malformed PDUs are defined in 13.4.

Many test cases allow flexibility in the value to be used in a service parameter. The tester is free to choose any value within the constraints defined in the test case. The IUT shall be able to respond correctly to any valid selection the tester might make. The EPICS is considered to be a definitive reference indicating the BACnet functionality supported and the configuration of the object database. Any discrepancies between the BACnet functionality defined in the EPICS and the functionality demonstrated by the device during testing shall constitute a failure. For example, it is considered a failure if a test step involves writing to a property and the EPICS indicates the property is writable, but the device returns an error indicating 'write access denied'.

### 9.1 AcknowledgeAlarm Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing AcknowledgeAlarm service requests.

BACnet devices that support initiation of ConfirmedEventNotification service requests shall pass the tests in 9.1.1.1 - 9.1.1.3 and 9.1.2.1 - 9.1.2.4. BACnet devices that support the initiation of UnconfirmedEventNotification service requests shall pass the tests in 9.1.1.4 - 9.1.1.6 and 9.1.2.5 - 9.1.2.7.

#### 9.1.1 Positive AcknowledgeAlarm Service Execution Tests

The purpose of this test group is to verify correct execution of the AcknowledgeAlarm service requests under circumstances where the service is expected to be successfully completed.

##### 9.1.1.1 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and all other recipients in the Recipient\_List. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of



the alarm notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 5. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, skip all steps related to receipt of the second notification.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (D1)
3. BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T1: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for this event),  
 'AckRequired' = TRUE,  
 'From State' = (any appropriate event state),  
 'To State' = (any appropriate event state),  
 'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE  
 DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number received in step 3),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for this event),  
 'AckRequired' = TRUE,  
 'From State' = (any appropriate event state),  
 'To State' = (any appropriate event state),  
 'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 'Time of Acknowledgment' = (the TD's current time using a Time format)
9. RECEIVE BACnet-SimpleACK-PDU
10. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedEventNotification-Request,



'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (the event type included in step 3),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (the 'To State' used in step 3)

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (the event type included in step 3),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

11. TRANSMIT BACnet-SimpleACK-PDU

12. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN

BEFORE **Notification Fail Time**

RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number received in step 10),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (the event type included in step 3),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (the 'To State' used in step 3)

ELSE

BEFORE **Notification Fail Time**

RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number received in step 10),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (the event type included in step 3),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

13. TRANSMIT BACnet-SimpleACK-PDU

14. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

### 9.1.1.2 Successful Alarm Acknowledgment of Confirmed Event Notifications using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.1. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_Class object, skip all steps related to receipt of the second notification.

### 9.1.1.3 Successful Alarm Acknowledgment of Confirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.1.1 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.1. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, skip all steps related to receipt of the second notification.

### 9.1.1.4 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Time Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Time form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 9.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (D1)
3. BEFORE **Notification Fail Time**  
RECEIVE,  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN  
RECEIVE  
DESTINATION = (a device other than the TD),  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (the timestamp or sequence number received in step 3),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (any valid time stamp),  
'Acknowledgement Source' = (any valid value),  
'Time of Acknowledgment' = (the TD's current time using a Time format)
7. RECEIVE BACnet-SimpleACK-PDU
8. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
BEFORE **Notification Fail Time**  
RECEIVE

DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (the 'To State' used in step 3 or 4)

ELSE

BEFORE **Notification Fail Time**

RECEIVE

DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the current time or sequence number),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION

9. IF (the notification in step 8 was not broadcast) THEN

IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN

RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number from the notification in step 8),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (the 'To State' used in step 3 or 4)

ELSE

RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (the timestamp or sequence number from the notification in step 8),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,

10. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

#### 9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Sequence Number form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall be a sequence number.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.4. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 9.

#### 9.1.1.6 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Date Time Form of the 'Time of Acknowledgment' Parameter

**Purpose:** To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status. The Date Time form of the 'Time of Acknowledgment' parameter is used.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.1.4 shall be followed except that the 'Time of Acknowledgment' parameter of the AcknowledgeAlarm service request shall convey the current time using a BACnetDateTime format.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.4. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 9.

#### 9.1.1.7 Successful Alarm Acknowledgment of any "Offnormal" Transitions Using an "Offnormal" 'To State'

**Purpose:** To verify the successful acknowledgment of an alarm that indicated an "offnormal" 'To State' other than OFFNORMAL.

**Test Concept:** An "offnormal" alarm is triggered in the IUT where the "offnormal" state is represented by an event-state other than OFFNORMAL (such as HIGH\_LIMIT or LOW\_LIMIT). The TD acknowledges the alarm with an 'Event State Acknowledged' of OFFNORMAL and verifies that the acknowledgment is accepted by the IUT.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and can enter "offnormal" states other than OFFNORMAL. The TD shall be a recipient of the alarm notification and the IUT shall be configured to send it unconfirmed.

**Test Steps:**

1. MAKE (a change that triggers the detection of an "offnormal" event other than OFFNORMAL in the IUT)

2. WAIT pTimeDelay
3. RECEIVE UnConfirmedEventNotification-Request,
 

|                                  |                                                                |
|----------------------------------|----------------------------------------------------------------|
| 'Process Identifier' =           | (the process identifier configured for this event),            |
| 'Initiating Device Identifier' = | IUT,                                                           |
| 'Event Object Identifier' =      | (the object detecting the alarm),                              |
| 'Time Stamp' =                   | (any valid time stamp),                                        |
| 'Notification Class' =           | (the notification class configured for this event),            |
| 'Priority' =                     | (the priority configured for this event),                      |
| 'Event Type' =                   | (E1: any valid event type),                                    |
| 'Message Text' =                 | (optional, any valid message text),                            |
| 'Notify Type' =                  | (any valid notify type),                                       |
| 'AckRequired' =                  | TRUE,                                                          |
| 'From State' =                   | (any valid event-state),                                       |
| 'To State' =                     | (S1: any "offnormal" event state other than OFFNORMAL itself), |
| 'Event Values' =                 | (the values appropriate to the event type)                     |
4. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = {0,?,?}
5. TRANSMIT AcknowledgeAlarm-Request,
 

|                                      |                                                                              |
|--------------------------------------|------------------------------------------------------------------------------|
| 'Acknowledging Process Identifier' = | (the value of the 'Process Identifier' parameter in the event notification), |
| 'Event Object Identifier' =          | (the 'Event Object Identifier' from the event notification),                 |
| 'Event State Acknowledged' =         | offnormal,                                                                   |
| 'Time Stamp' =                       | (the timestamp conveyed in the notification),                                |
| 'Acknowledgement Source' =           | (any valid value),                                                           |
| 'Time of Acknowledgment' =           | (any valid timestamp)                                                        |
6. RECEIVE BACnet-SimpleACK-PDU
7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN
 

|                                               |                                                     |
|-----------------------------------------------|-----------------------------------------------------|
| RECEIVE UnconfirmedEventNotification-Request, |                                                     |
| 'Process Identifier' =                        | (the process identifier configured for this event), |
| 'Initiating Device Identifier' =              | IUT,                                                |
| 'Event Object Identifier' =                   | (the object detecting the alarm),                   |
| 'Time Stamp' =                                | (any valid time stamp),                             |
| 'Notification Class' =                        | (the notification class configured for this event), |
| 'Priority' =                                  | (the priority configured for this event),           |
| 'Message Text' =                              | (optional, any valid message text),                 |
| 'Event Type' =                                | (E1),                                               |
| 'Notify Type' =                               | ACK_NOTIFICATION,                                   |
| 'To State' =                                  | (OFFNORMAL or S1)                                   |

 ELSE
 

|                                               |                                                     |
|-----------------------------------------------|-----------------------------------------------------|
| RECEIVE UnconfirmedEventNotification-Request, |                                                     |
| 'Process Identifier' =                        | (the process identifier configured for this event), |
| 'Initiating Device Identifier' =              | IUT,                                                |
| 'Event Object Identifier' =                   | (the object detecting the alarm),                   |
| 'Time Stamp' =                                | (any valid time stamp),                             |
| 'Notification Class' =                        | (the notification class configured for this event), |
| 'Priority' =                                  | (the priority configured for this event),           |
| 'Event Type' =                                | (E1),                                               |
| 'Message Text' =                              | (optional, any valid message text),                 |
| 'Notify Type' =                               | ACK_NOTIFICATION,                                   |
8. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = {1,?,?}

#### 9.1.1.8 Successful Alarm Acknowledgment of Confirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification when the acknowledgement contains a mismatched or unknown 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or an unknown 'Acknowledging Process Identifier' (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.



Configuration Requirements: The IUT shall be configured with at least one object, Object1, which can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification and shall use different Process Identifiers. D1 is either the pTimeDelay or pTimeDelayNormal parameter or 0 (for transitions to and from FAULT state) depending on the event transition.

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol\_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked\_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. WAIT D1
4. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (any Process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = Object1,  
'Time Stamp' = (any valid time stamp),  
'Notification Class' = (the Notification Class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ALARM or EVENT,  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (values appropriate to the event type)
5. TRANSMIT BACnet-SimpleACK-PDU
6. RECEIVE  
DESTINATION = (DST1: a device other than the TD),  
SOURCE = IUT,  
ConfirmedEventNotification-Request,  
'Process Identifier' = (any Process ID),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = Object1,  
'Time Stamp' = (any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ALARM | EVENT,  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (values appropriate to the event type)
7. TRANSMIT  
DESTINATION = IUT,  
SOURCE = (DST1),  
BACnet-SimpleACK-PDU
8. VERIFY (Object1), Acked\_Transitions = (one bit FALSE, the others TRUE)
9. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (Any mismatched or unknown value),  
'Event Object Identifier' = Object1,  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (the timestamp conveyed in the notification),  
'Acknowledgement Source' = (any valid value),

'Time of Acknowledgment' = (the current timestamp)

10. RECEIVE BACnet-SimpleACK-PDU

11. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any Process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = Object1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the Notification Class configured for this event),

'Priority' = (the priority configured for this event),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK\_NOTIFICATION,

'To State' = (any appropriate event state)

ELSE

BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (any Process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = Object1,

'Time Stamp' = (the current time or sequence number any valid time stamp),

'Notification Class' = (the Notification Class configured for this event),

'Priority' = (the priority configured for this event),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK\_NOTIFICATION,

12. TRANSMIT BACnet-SimpleACK-PDU

13. IF (Protocol\_Revision is present and Protocol\_Revision >= 1) THEN

RECEIVE

DESTINATION = (a device other than the TD),

SOURCE = IUT,

ConfirmedEventNotification-Request,

'Process Identifier' = (any Process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = Object1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for this event),

'Priority' = (the priority configured for this event),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK\_NOTIFICATION,

'To State' = (any appropriate event state)

ELSE

RECEIVE

DESTINATION = (a device other than the TD),

SOURCE = IUT,

ConfirmedEventNotification-Request,

'Process Identifier' = (any Process ID),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = Object1,

'Time Stamp' = (any valid time stamp),

'Notification Class' = (the notification class configured for this event),

'Priority' = (the priority configured for this event),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK\_NOTIFICATION

14. TRANSMIT

DESTINATION = IUT,

SOURCE = (DST1),

BACnet-SimpleACK-PDU

15. VERIFY (Object1), Acked\_Transitions = (TRUE,TRUE,TRUE)

Notes to Tester: The ConfirmedEventNotification-Request messages can be received in either order. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 6 and 7.

#### 9.1.1.9 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using an Unknown 'Acknowledging Process Identifier' Parameter

Purpose: To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification when the acknowledgement contains a mismatched or unknown 'Acknowledging Process Identifier' parameter.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm with a mismatched 'Acknowledging Process Identifier' (the Process Identifier associated with another recipient), or unknown (a Process Identifier not associated with any recipient), and verifies that the acknowledgment is properly noted by the IUT. This test should be performed twice, once with a mismatched Process Identifier and once with an unknown Process Identifier.

Configuration Requirements: The IUT shall be configured with at least one object, Object1, that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value (TRUE,TRUE,TRUE), indicating that all transitions have been acknowledged. The TD and at least one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification, configured to receive different Process Identifiers. D1 is either the pTimeDelay or pTimeDelayNormal parameter or 0 (for transitions to and from FAULT state) depending on the event transition.

This test is recommended for all BACnet devices that execute AcknowledgeAlarm but is required only for those that claim conformance to Protocol\_Revision 5 or greater.

Test Steps:

1. VERIFY (Object1), Acked\_Transitions = (TRUE,TRUE,TRUE)
2. MAKE (a change that triggers the detection of an alarm event in the IUT)
3. WAIT D1
4. BEFORE **Notification Fail Time**
  - RECEIVE UnconfirmedEventNotification-Request,
    - 'Process Identifier' = (any Process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = Object1,
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the Notification Class configured for this event),
    - 'Priority' = (the priority configured for this event),
    - 'Event Type' = (any valid event type),
    - 'Message Text' = (optional, any valid message text),
    - 'Notify Type' = ALARM or EVENT,
    - 'AckRequired' = TRUE,
    - 'From State' = (any appropriate event state),
    - 'To State' = (any appropriate event state),
    - 'Event Values' = (values appropriate to the event type)
5. RECEIVE
  - DESTINATION = (a device other than the TD),
  - SOURCE = IUT,
  - UnconfirmedEventNotification-Request,
    - 'Process Identifier' = (any Process ID),
    - 'Initiating Device Identifier' = IUT,
    - 'Event Object Identifier' = (the object detecting the alarm),
    - 'Time Stamp' = (any valid time stamp),
    - 'Notification Class' = (the notification class configured for this event),
    - 'Priority' = (the priority configured for this event),
    - 'Event Type' = (any valid event type),
    - 'Message Text' = (optional, any valid message text),

- 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE,  
 'From State' = (any appropriate event state),  
 'To State' = (any appropriate event state),  
 'Event Values' = (values appropriate to the event type)
6. VERIFY (Object1), Acked\_Transitions = (one bit FALSE, the others TRUE)
7. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (Any mismatched or unknown value),  
 'Event Object Identifier' = Object1,  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the timestamp conveyed in the notification),  
 'Acknowledgement Source' = (any valid value),  
 'Time of Acknowledgment' = (the current timestamp)
8. RECEIVE BACnet-SimpleACK-PDU
- 9 . BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any Process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = Object1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the Notification Class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (any appropriate event state)
10. RECEIVE  
 DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (any Process ID),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = Object1,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (any appropriate event state)
11. VERIFY (Object1), Acked\_Transitions = (TRUE,TRUE,TRUE)

Note to Tester: The UnconfirmedEventNotification-Request messages can be received in either order. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit step 5.

#### 9.1.1.10 Successful Alarm Re-Acknowledgment of Confirmed Event Notifications

Purpose: To verify the successful re-acknowledgment of an event signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status.

Test Concept: An event is triggered and the IUT notifies the TD and one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol\_Revision 7 and so this test shall only be performed if Protocol\_Revision is present (i.e., Protocol\_Revision >= 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends confirmed notifications. The Acked\_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the event notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 12, 13, 19, and 20.

Test Steps:

1. MAKE (a change that triggers the detection of an event in the IUT)
2. WAIT (D1)
3. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T1: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (E1: any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (S1: any appropriate offnormal event state),  
'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE  
DESTINATION = (a device other than the TD),  
SOURCE = IUT,  
ConfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T1),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for this event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (any valid time stamp),  
'Acknowledgment Source' = (any valid value)  
'Time of Acknowledgment' = (any of the forms specified for this parameter)
9. RECEIVE BACnet-SimpleACK-PDU
10. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T2: any valid time stamp),

'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

11. TRANSMIT BACnet-SimpleACK-PDU

12. RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

13. TRANSMIT BACnet-SimpleACK-PDU

14. VERIFY (the 'Event Object Identifier' from the event notification), Acked Transitions = (TRUE, TRUE, TRUE)

15. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),

'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (any valid time stamp),  
 'Acknowledgment Source' = (any valid value)  
 'Time of Acknowledgment' = (any of the forms specified for this parameter)

16. RECEIVE BACnet-SimpleACK-PDU

17. BEFORE **Notification Fail Time**

RECEIVE ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T3: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

18. TRANSMIT BACnet-SimpleACK-PDU

19. RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 ConfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T3),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)



20. TRANSMIT BACnet-SimpleACK-PDU

21. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

#### 9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications

Purpose: To verify the successful re-acknowledgment of an event signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status.

Test Concept: An event is triggered and the IUT notifies the TD and one other device. The TD acknowledges the event and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all recipients that the event has been acknowledged. The TD then acknowledges the event again, and the IUT again notifies all recipients. This behavior was not defined before Protocol\_Revision 7 and so this test shall only be performed if Protocol\_Revision is present (i.e., Protocol\_Revision >= 7).

Configuration Requirements: The IUT shall be configured with at least one event-initiating object that generates offnormal transitions and sends unconfirmed notifications. The Acked\_Transitions property shall have the value B'111', indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the event notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4, 9, and 14.

Test Steps:

1. MAKE (a change that triggers the detection of an offnormal event in the IUT)
2. WAIT (D1)
3. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T1: any valid time stamp),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1: any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for this event),  
 'AckRequired' = TRUE,  
 'From State' = NORMAL,  
 'To State' = (S1: any appropriate offnormal event state),  
 'Event Values' = (the values appropriate to the event type)

4. RECEIVE

DESTINATION = (at least one device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T1),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = (the notify type configured for this event),  
 'AckRequired' = TRUE,  
 'From State' = (any appropriate event state),  
 'To State' = (any appropriate event state),  
 'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (any valid time stamp),  
'Acknowledgment Source' = (any valid value)  
'Time of Acknowledgment' = (any of the forms specified for this parameter)
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**  
RECEIVE UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T2: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (E1),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,  
'To State' = (S1)
9. RECEIVE  
DESTINATION = (a device other than the TD),  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T2),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (E1),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,  
'To State' = (S1)
10. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)
11. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (the time stamp conveyed in the notification),  
'Acknowledgment Source' = (any valid value)  
'Time of Acknowledgment' = (any of the forms specified for this parameter)
12. RECEIVE BACnet-SimpleACK-PDU
13. BEFORE **Notification Fail Time**  
RECEIVE UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the event-initiating object),  
'Time Stamp' = (T3: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event),  
'Event Type' = (E1),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,  
'To State' = (S1)

## 14. RECEIVE

DESTINATION = (a device other than the TD),  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the event-initiating object),  
 'Time Stamp' = (T3),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event),  
 'Event Type' = (E1),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1)

15. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

#### 9.1.1.12 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is an Offnormal State other than OFFNORMAL

**Purpose:** To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status, when the 'To State' parameter is an offnormal state other than OFFNORMAL (e.g. HIGH\_LIMIT) and the 'Event State Acknowledged' parameter is OFFNORMAL.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and at least one other device with an offnormal 'To State' other than OFFNORMAL. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of OFFNORMAL and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in Clause 9.1.1.1 shall be followed except that the 'To State' parameter shall be an offnormal state other than OFFNORMAL. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of OFFNORMAL.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.1.

#### 9.1.1.13 Successful Alarm Acknowledgment of Unconfirmed Event Notifications when 'To State' is an Offnormal State other than OFFNORMAL

**Purpose:** To verify the successful acknowledgment of an alarm signaled by an UnconfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status, when the 'To State' parameter is an offnormal state other than OFFNORMAL (e.g. HIGH\_LIMIT) and the 'Event State Acknowledged' parameter is OFFNORMAL.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and at least one other device with an offnormal 'To State' other than OFFNORMAL. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of OFFNORMAL and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in Clause 9.1.1.4 shall be followed except that the 'To State' parameter shall be an offnormal state other than OFFNORMAL. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of OFFNORMAL.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in Clause 9.1.1.4.

#### 9.1.1.14 Successful Alarm Acknowledgment of Confirmed Event Notifications when 'To State' is either High-Limit or Low-Limit, Revision 5 and higher only

**Purpose:** To verify the successful acknowledgment of an alarm signaled by a ConfirmedEventNotification, including notification of other workstations and updating of the Acked\_Transitions status when the 'To State' parameter is either High-Limit or Low-Limit and the 'Event State Acknowledged' parameter is Off-Normal.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device with an 'To State' event of either High-Limit or Low-Limit. The TD acknowledges the alarm using all of the correct parameters and using an 'Event State Acknowledged' parameter of 'Off-Normal' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.1.1 shall be followed except that the 'To State' parameter shall be either High-Limit or Low-Limit. When acknowledging the alarm, the TD shall use an 'Event State Acknowledged' parameter of Off-Normal.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.1.1. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, skip all steps related to receipt of the second notification.

#### 9.1.1.15 Unsupported Message Text Character Set AcknowledgeAlarm Test

**Purpose:** To verify that the IUT does not fail to process an AcknowledgeAlarm request because the Acknowledgment Source parameter is of a character set that the IUT does not support.

**Test Concept:** Cause an event-initiating object, O1, in the IUT to transition to Event\_State ES1. Acknowledge the transition and, in the AcknowledgeAlarm service, provide an 'Acknowledgment Source' parameter, AS1, which has a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Acknowledgment Source' uses a character set that the IUT does not support, and that the IUT accepts and applies that Acknowledgment request, irrespective of the 'Acknowledgment Source'.

**Configuration Requirements:** Configure an event-initiating object, O1, which references a Notification Class object, N1. Configure O1 such that it needs an acknowledgment when it transitions out of its current state. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition. AS1 shall be a character string short enough for the IUT to receive and encoded in a character set that the IUT does not support. If the IUT supports all character sets, this test shall be skipped.

**Test Steps:**

1. MAKE (a condition exist that will cause O1 to create a transition)
2. WAIT D1
3. BEFORE **Notification Fail Time**

RECEIVE UnconfirmedEventNotification-Request

|                                  |                                                |
|----------------------------------|------------------------------------------------|
| 'Process Identifier' =           | (any valid process identifier),                |
| 'Initiating Device Identifier' = | IUT,                                           |
| 'Event Object Identifier' =      | O1,                                            |
| 'Time Stamp' =                   | (TS1: any valid timestamp),                    |
| 'Notification Class' =           | (N1: the Notification_Class configured in O1), |
| 'Priority' =                     | (any valid priority),                          |
| 'Event Type' =                   | (any standard event type),                     |
| 'Message Text' =                 | (optional, any valid message text),            |
| 'Notify Type' =                  | ALARM   EVENT,                                 |
| 'AckRequired' =                  | TRUE,                                          |
| 'From State' =                   | (any valid event state),                       |
| 'To State' =                     | (ES1: any valid event state),                  |
| 'Event Values' =                 | (any values appropriate to the event type)     |

4. IF (ES1 = NORMAL) THEN  
     VERIFY Acked\_Transitions = (?, ?, F)  
   ELSE IF (ES1 = FAULT) THEN  
     VERIFY Acked\_Transitions = (?, F, ?)  
   ELSE  
     VERIFY Acked\_Transitions = (F, ?, ?)
5. TRANSMIT AcknowledgeAlarm-Request  
     'Acknowledging Process Identifier' = (any valid value),  
     'Event Object Identifier' = O1,  
     'Event State Acknowledged' = ES1,  
     'Time Stamp' = TS1,  
     'Acknowledgment Source' = AS1,  
     'Time of Acknowledgment' = (any valid timestamp)
6. RECEIVE BACnet-SimpleACK-PDU
7. BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedEventNotification-Request  
     'Process Identifier' = (any valid process identifier),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = O1,  
     'Time Stamp' = TS1  
     Notification Class' = (N1: the Notification\_Class configured in O1),  
     'Priority' = (any valid priority),  
     'Event Type' = (any standard event type),  
     'Message Text' = (optional, any valid message text),  
     'Notify Type' = ACK\_NOTIFICATION,  
     'To State' = ES1
8. IF (ES1 = NORMAL) THEN  
     VERIFY Acked\_Transitions = (?, ?, T)  
   ELSE IF (ES1 = FAULT) THEN  
     VERIFY Acked\_Transitions = (?, T, ?)  
   ELSE  
     VERIFY Acked\_Transitions = (T, ?, ?)

Notes to Tester: The use of UnconfirmedEventNotification is specified in this test, solely to simplify the expression of the test. The behavior being tested applies to the ConfirmedEventNotification service as well.

### 9.1.2 Negative AcknowledgeAlarm Service Execution Tests

The purpose of this test group is to verify correct execution of the AcknowledgeAlarm service requests under circumstances where the service is expected to fail. All of the test cases represent examples of ways in which the AcknowledgeAlarm service parameters may be inconsistent with the current alarm status of the object.

#### 9.1.2.1 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (D1)
3. BEFORE **Notification Fail Time**  
RECEIVE ConfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T1: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (E1: any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for the event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (S1: any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE  
DESTINATION = (a device other than the TD),  
SOURCE = IUT,  
ConfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T1),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (E2: any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for the event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (S2: any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (any valid time stamp older than T1),  
'Acknowledgment Source' = (any valid value),  
'Time of Acknowledgment' = (the current time using a Time format)
9. RECEIVE BACnet-Error-PDU  
Error Class = SERVICES,  
Error Code = INVALID\_TIME\_STAMP
10. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
11. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the process identifier configured for this event),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),



'Time Stamp' = T1,  
 'Time of Acknowledgment' = (the current time using a Time format)

12. RECEIVE BACnet-SimpleACK-PDU

13. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     BEFORE **Notification Fail Time**  
     RECEIVE  
         ConfirmedEventNotification-Request,  
         'Process Identifier' = (the process identifier configured for this event),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object detecting the alarm),  
         'Time Stamp' = (T2: any valid time stamp),  
         'Notification Class' = (the notification class configured for this event),  
         'Priority' = (the priority configured for this event type),  
         'Event Type' = (any valid event type),  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = ACK\_NOTIFICATION,  
         'To State' = (S1 or S2)

ELSE  
     BEFORE **Notification Fail Time**  
     RECEIVE  
         ConfirmedEventNotification-Request,  
         'Process Identifier' = (the process identifier configured for this event),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object detecting the alarm),  
         'Time Stamp' = (T2: any valid time stamp),  
         'Notification Class' = (the notification class configured for this event),  
         'Priority' = (the priority configured for this event type),  
         'Event Type' = (any valid event type),  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = ACK\_NOTIFICATION

14. TRANSMIT BACnet-SimpleACK-PDU

15. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
     RECEIVE  
         DESTINATION = (a device other than the TD),  
         SOURCE = IUT,  
         ConfirmedEventNotification-Request,  
         'Process Identifier' = (the process identifier configured for this event),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object detecting the alarm),  
         'Time Stamp' = (T2),  
         'Notification Class' = (the notification class configured for this event),  
         'Priority' = (the priority configured for this event type),  
         'Event Type' = (any valid event type),  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = ACK\_NOTIFICATION,  
         'To State' = (S1 or S2)

ELSE  
     RECEIVE  
         DESTINATION = (a device other than the TD),  
         SOURCE = IUT,  
         ConfirmedEventNotification-Request,  
         'Process Identifier' = (the process identifier configured for this event),  
         'Initiating Device Identifier' = IUT,  
         'Event Object Identifier' = (the object detecting the alarm),  
         'Time Stamp' = (T2),  
         'Notification Class' = (the notification class configured for this event),  
         'Priority' = (the priority configured for this event type),  
         'Event Type' = any valid event type),  
         'Message Text' = (optional, any valid message text),  
         'Notify Type' = ACK\_NOTIFICATION

16. TRANSMIT BACnet-SimpleACK-PDU

17. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

#### 9.1.2.2 Deleted Clause

This test has been removed.

#### 9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event Object Identifier' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist or is not consistent with the other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an improper 'Event Object Identifier' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event Object Identifier' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall specify an object that does not support or is not configured for alarming, or which does not exist.

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the Error Class and Error Code in step 9 shall be OBJECT and UNKNOWN\_OBJECT if the object referenced by 'Event Object Identifier' does not exist or OBJECT and NO\_ALARM\_CONFIGURED if the object exists but does not support or is not configured for alarming. For devices claiming a Protocol Revision less than 5, an Error Class and Error Code of SERVICES and INCONSISTENT\_PARAMETERS or Error Class of OBJECT and Error Code of OTHER shall also be allowed. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.

#### 9.1.2.4 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the Event\_State that was provided in the notification which is being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an invalid event state and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event state and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.1 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification, the 'To State' in the notification shall be any offnormal transition and the 'Event State Acknowledged' shall have an offnormal value that is different from the 'To State' in the event notification and shall not be OFFNORMAL.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.1 except that the Error Code in step 9 shall be INVALID\_EVENT\_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of INCONSISTENT\_PARAMETERS shall also be allowed. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.

### 9.1.2.5 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Time Stamp' is Too Old

Purpose: To verify that an alarm remains unacknowledged if the time stamp in the acknowledgment does not match the most recent transition to the current alarm state.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Aacked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification. D1 is either the pTimeDelay, or pTimeDelayNormal parameter, or 0 (for transitions to and from FAULT state) depending on the event transition.

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT D1
3. BEFORE **Notification Fail Time**  
RECEIVE UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T1: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for the event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (S1: any appropriate event state),  
'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN  
RECEIVE  
DESTINATION = (at least one device other than the TD),  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T1),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = (the notify type configured for the event),  
'AckRequired' = TRUE,  
'From State' = (any appropriate event state),  
'To State' = (S2: any appropriate event state),  
'Event Values' = (the values appropriate to the event type)

5. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (a time stamp older than the one conveyed in the notification),  
'Acknowledgement Source' = (any valid value),  
'Time of Acknowledgment' = (the TD's current time using a Time format)
7. RECEIVE BACnet-Error-PDU  
Error Class = SERVICES,  
Error Code = INVALID\_TIME\_STAMP
8. VERIFY (the 'Event Object Identifier' from the event notification),  
Acked\_Transitions = (appropriate bit FALSE, the others TRUE)
9. TRANSMIT AcknowledgeAlarm-Request,  
'Acknowledging Process Identifier' = (the process identifier configured for this event),  
'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
'Time Stamp' = (the time stamp conveyed in the notification),  
'Acknowledgement Source' = (any valid value),  
'Time of Acknowledgment' = (the TD's current time using a Time format)
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
BEFORE **Notification Fail Time**  
RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T2: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION,  
'To State' = (S1 or S2)
- ELSE  
BEFORE **Notification Fail Time**  
RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,  
'Process Identifier' = (the process identifier configured for this event),  
'Initiating Device Identifier' = IUT,  
'Event Object Identifier' = (the object detecting the alarm),  
'Time Stamp' = (T2: any valid time stamp),  
'Notification Class' = (the notification class configured for this event),  
'Priority' = (the priority configured for this event type),  
'Event Type' = (any valid event type),  
'Message Text' = (optional, any valid message text),  
'Notify Type' = ACK\_NOTIFICATION
12. IF (the notification in step 11 was not broadcast) THEN  
IF (Protocol\_Revision is present AND Protocol\_Revision >= 1) THEN  
RECEIVE  
DESTINATION = (at least one device other than the TD),  
SOURCE = IUT,  
UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (T2),  
 'Notification Class' = (the notification class configured for this event),  
 'Priority' = (the priority configured for this event type),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ACK\_NOTIFICATION,  
 'To State' = (S1 or S2)

ELSE

RECEIVE

DESTINATION = (at least one device other than the TD),

SOURCE = IUT,

UnconfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),

'Initiating Device Identifier' = IUT,

'Event Object Identifier' = (the object detecting the alarm),

'Time Stamp' = (T2),

'Notification Class' = (the notification class configured for this event),

'Priority' = (the priority configured for this event type),

'Event Type' = (any valid event type),

'Message Text' = (optional, any valid message text),

'Notify Type' = ACK\_NOTIFICATION

13. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE,TRUE,TRUE)

#### 9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event Object Identifier' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist or is not consistent with the other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an improper event object identifier and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event object identifier and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall specify an object that does not support or is not configured for alarming, or which does not exist.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Class and Error Code in step 7 shall be OBJECT and UNKNOWN\_OBJECT if the object referenced by 'Event Object Identifier' does not exist or OBJECT and NO\_ALARM\_CONFIGURED if the object exists but does not support or is not configured for alarming. For devices claiming a Protocol Revision less than 5, an Error Class and Error Code of SERVICES and INCONSISTENT\_PARAMETERS or Error Class of OBJECT and Error Code of OTHER shall also be allowed. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.

#### 9.1.2.7 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the 'Event State Acknowledged' is Invalid

Purpose: To verify that an alarm remains unacknowledged if the 'Event State Acknowledged' is inconsistent with the Event\_State that was provided in the notification which is being acknowledged.

**Test Concept:** An alarm is triggered that causes the IUT to notify the TD and one other device. The TD acknowledges the alarm using an invalid 'Event State Acknowledged' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event State Acknowledged' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

**Configuration Requirements:** The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked\_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and one other BACnet device, if the IUT supports multiple recipients, shall be recipients of the alarm notification.

**Test Steps:** The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification, the 'To State' in the notification shall be any offnormal transition and the 'Event State Acknowledged' shall have an offnormal value that is different from the 'To State' in the event notification and shall not be OFFNORMAL.

**Notes to Tester:** A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Code in step 7 shall be INVALID\_EVENT\_STATE. For devices claiming a Protocol Revision less than 5, an Error Code of INCONSISTENT\_PARAMETERS shall also be allowed. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.

## 9.2 ConfirmedCOVNotification Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ConfirmedCOVNotification service requests. The ConfirmedCOVNotification tests are specific to a particular object type that provides intrinsic COV reporting capabilities. The IUT shall pass all of the tests for each standard BACnet object type that has optional or required intrinsic COV reporting capability.

### 9.2.1 Positive ConfirmedCOVNotification Service Execution Tests

The purpose of this test group is to verify correct execution of the ConfirmedCOVNotification service requests under circumstances where the service is expected to be successfully completed.

#### 9.2.1.1 Change of Value Notifications

**Purpose:** To verify that the IUT can execute ConfirmedCOVNotification requests from object types that provide the Present\_Value and Status\_Flags properties in COV notifications.

**Test Concept:** The IUT is made to subscribe for COV from an object of the type being tested. The TD then sends a COV notification to the IUT and verifies that the IUT exhibits any actions identified by the vendor.

**Test Steps:**

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, and additional properties appropriate to object type X)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 9.2.1.2 Change of Value Notification from Loop Objects

**Purpose:** To verify that the IUT can execute ConfirmedCOVNotification requests from loop objects.



Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any Loop object, X),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining in the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, Setpoint, and Controlled\_Variable\_Value appropriate to object X)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.2.1.3 Change of Value Notification from Pulse Converter Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Pulse Converter objects.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any Pulse Converter object, X),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining for the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, and Update)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.2.1.4 Change of Value Notification from Load Control Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Load Control objects.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any Load Control object, X),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining for the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, Requested\_Shed\_Level, Start\_Time, Shed\_Duration, and Duty\_Window appropriate to object X)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

**9.2.1.5 ConfirmedCOVNotification from Access Door Object**

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier value > 0),
  - 'Monitored Object Identifier' = (any Access Door object, X),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the process identifier used in step 1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (the time remaining in the subscription),
  - 'List of Values' = (the initial Present\_Value, initial Status\_Flags, and Door\_Alarm\_State if X has a Door\_Alarm\_State property)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

**9.2.1.6 ConfirmedCOVNotification from Access Point Object**

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Point objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),
  - 'Monitored Object Identifier' = X: any Access Point object),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = PI,
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (the time remaining in the subscription),
  - 'List of Values' = (any valid set of values)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

**9.2.1.7 ConfirmedCOVNotification from Credential Data Input Object**

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Credential Data Input objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),
  - 'Monitored Object Identifier' = (X: any Credential Data Input object),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = PI,
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (the time remaining in the subscription),
  - 'List of Values' = (any valid set of values)

4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.2.2 Negative ConfirmedCOVNotification Service Execution Tests

The purpose of this test group is to verify correct execution of the ConfirmedCOVNotification service requests under circumstances where the service is expected to fail. All of the test cases represent examples of ways in which the ConfirmedCOVNotification service parameters may be inconsistent with the current status of the subscription from the perspective of the IUT.

#### 9.2.2.1 Change of Value Notification Arrives after Subscription has Expired

Purpose: To verify that an appropriate error is returned if a COV notification arrives after the subscription time period has expired.

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier, P1),
  - 'Monitored Object Identifier' = (any object X of a type that supports COV notification),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (any valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (P1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (any amount of time greater than 0),
  - 'List of Values' = (a list of values appropriate to object X)
4. IF (the IUT can cancel the subscription) THEN
  - RECEIVE SubscribeCOV – Request,
    - 'Subscriber Process Identifier' = (PI),
    - 'Monitored Object Identifier' = X
  - ELSE
    - MAKE (the IUT stop resubscribing, if it resubscribes automatically)
5. WAIT (at least Lifetime, but sufficient to ensure the subscription has expired)
6. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (P1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (any amount of time greater than 0),
  - 'List of Values' = (a list of values appropriate to object X)
7. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN
  - RECEIVE BACnet-Error-PDU,
    - 'Error Class' = SERVICES,
    - 'Error Code' = UNKNOWN\_SUBSCRIPTION |
  - (BACnet-SimpleACK-PDU)
  - ELSE
    - RECEIVE BACnet-Error-PDU,
      - 'Error Class' = SERVICES,
      - 'Error Code' = (any valid error code for class SERVICES) |
    - (BACnet-SimpleACK-PDU)

#### 9.2.2.2 Change of Value Notifications with Invalid Process Identifier

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a process identifier that does not match any current subscriptions.

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (P1, any valid process identifier),  
     'Monitored Object Identifier' = (any object X of a type that supports COV notification),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (any process identifier different from P1),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any amount of time greater than 0),  
     'List of Values' = (a list of values appropriate to object X)
4. IF (Protocol\_Revision is present AND Protocol\_Revision >= 10) THEN  
     RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (UNKNOWN\_SUBSCRIPTION) |  
         (BACnet-SimpleACK-PDU)  
     ELSE  
         RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (any valid error code for class SERVICES) |  
         (BACnet-SimpleACK-PDU)

### 9.2.2.3 Change of Value Notifications with Invalid Initiating Device Identifier

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains an initiating device identifier that does not match any current subscriptions.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object X of a type that supports COV notification),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value no greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier different used in step 2),  
     'Initiating Device Identifier' = (any valid Device object except TD),  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any amount of time greater than 0),  
     'List of Values' = (a list of values appropriate to object X)
4. IF (Protocol\_Revision is present AND Protocol\_Revision >= 10) THEN  
     RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (UNKNOWN\_SUBSCRIPTION) |  
         (BACnet-SimpleACK-PDU)  
     ELSE  
         RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (any valid error code for class SERVICES) |  
         (BACnet-SimpleACK-PDU)

**9.2.2.4 Change of Value Notifications with Invalid Monitored Object Identifier**

Purpose: To verify that an appropriate error is returned if a COV notification arrives that contains a monitored object identifier that does not match any current subscriptions.

Configuration Requirements: If the IUT does not support initiation of SubscribeCOV-Request with 'Issue Confirmed Notifications' equal to TRUE, then this test shall be skipped.

Notes to Tester: If possible, select an object Y for which IUT supports COV Subscription.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object X of a type that supports COV notification),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any valid Lifetime)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = (any object Y in the IUT except X, and for which IUT  
         does not already have an active subscription),  
     'Time Remaining' = (any amount of time greater than 0),  
     'List of Values' = (a list of values appropriate to object Y)
4. IF (Protocol\_Revision is present AND Protocol\_Revision >= 10) THEN  
     RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (UNKNOWN\_SUBSCRIPTION) |  
         (BACnet-SimpleACK-PDU)  
     ELSE  
         RECEIVE  
         BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (any valid error code for class SERVICES) |  
         (BACnet-SimpleACK-PDU)

**9.2.2.5 Deleted Clause**

This clause has been removed.

**9.3 UnconfirmedCOVNotification Service Execution Tests**

The purpose of this test group is to verify correct execution of the UnconfirmedCOVNotification service requests under circumstances where the service is expected to be successfully completed.

**9.3.1 Device Restart Notifications**

Purpose: To verify that the IUT executes UnconfirmedCOVNotification service requests that convey restart notifications.

Test Concept: The TD sends a restart notification and the tester verifies that the IUT processes the notification in a vendor-specified manner.

Configuration Requirements: A valid BACnetDeviceStatus and a valid Last\_Restart\_Reason pair, DS<sub>1</sub> and LRR<sub>1</sub>, shall be selected for which the IUT will take some action.

Device restart notifications differ from subscribed COV notifications that use the UnconfirmedCOVNotification service in two respects. First, subscription is made through the Restart\_Notification\_Recipients property instead of SubscribeCOV. Second, the 'Subscriber Process Identifier' parameter always has a value of zero.

Test Steps:

1. TRANSMIT UnconfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = 0,
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = (the TD Device Identifier),
  - 'Time Remaining' = 0,
  - 'List of Values' = (System\_Status = DS<sub>1</sub>,  
Time\_Of\_Device\_Restart = (any valid time),  
Last\_Restart\_Reason = LRR<sub>1</sub>)
2. CHECK(that the IUT processes the notification as described by the vendor)

### 9.3.2 Positive UnconfirmedCOVNotification Service Execution Tests

The purpose of this test group is to verify correct execution of the UnconfirmedCOVNotification service requests under circumstances where the service is expected to be successfully completed.

#### 9.3.2.1 Change of Value Notifications

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from objects that provide the Present\_Value and Status\_Flags properties in COV notifications.

Test Concept: The IUT is made to subscribes for COV from an object of the type being tested. The TD then sends a COV notification to the IUT and verifies that the IUT exhibits any actions identified by the vendor.

Test Steps:

1. RECEIVE SubscribeCOV,
  - 'Subscriber Process Identifier' = (P1, any valid process identifier),
  - 'Monitored Object Identifier' = X,
  - 'Issue Confirmed Notifications' = FALSE,
  - 'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (P1),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (the time remaining for the subscription),
  - 'List of Values' = (Present\_Value, Status\_Flags, and additional properties appropriate to object type X)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 9.3.2.2 Change of Value Notification from Loop Objects

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Loop objects.

Test Steps:

1. RECEIVE SubscribeCOV,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any Loop object, X),
  - 'Issue Confirmed Notifications' = FALSE,
  - 'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the process identifier used in step 2),
  - 'Initiating Device Identifier' = TD,
  - 'Monitored Object Identifier' = X,
  - 'Time Remaining' = (the time remaining for the subscription),
  - 'List of Values' = (Present\_Value, Status\_Flags, Setpoint, and  
Controlled\_Variable\_Value appropriate to object X)



4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.3.2.3 Change of Value Notification from Pulse Converter Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Pulse Converter objects.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any Pulse Converter object, X),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining for the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, and Update\_Time appropriate to object X)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.3.2.4 Change of Value Notification from Load Control Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Load Control objects.

Test Steps:

1. RECEIVE SubscribeCOV,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any Load Control object, X),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than 1 minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 2),  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (the time remaining for the subscription),  
     'List of Values' = (Present\_Value, Status\_Flags, Requested\_Shed\_Level, Start\_Time, Shed\_Duration, and Duty\_Window appropriate to object X)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.3.2.5 UnconfirmedCOVNotification from Access Door Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier value > 0),  
     'Monitored Object Identifier' = (any Access Door object, X),  
     'Issue Confirmed Notifications' = FALSE,  
     'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the process identifier used in step 1),  
     'Initiating Device Identifier' = TD,

'Monitored Object Identifier' = X,  
 'Time Remaining' = the time remaining in the subscription),  
 'List of Values' = (the initial Present\_Value, initial Status\_Flags, and  
 Door\_Alarm\_State if X has a Door\_Alarm\_State property)

4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 9.3.2.6 UnconfirmedCOVNotification from Access Point Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Point objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
 'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
 'Monitored Object Identifier' = (X: any Access Door object),  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PI,  
 'Initiating Device Identifier' = TD,  
 'Monitored Object Identifier' = X,  
 'Time Remaining' = (the time remaining in the subscription),  
 'List of Values' = (any valid set of values)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

#### 9.3.2.7 UnconfirmedCOVNotification from Credential Data Input Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Credential Data Input objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,  
 'Subscriber Process Identifier' = (PI: any valid process identifier value > 0),  
 'Monitored Object Identifier' = (X: any Access Door object),  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PI,  
 'Initiating Device Identifier' = TD,  
 'Monitored Object Identifier' = X,  
 'Time Remaining' = (the time remaining in the subscription),  
 'List of Values' = (any valid set of values)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

### 9.4 ConfirmedEventNotification Service Execution Tests

Purpose: To verify that the IUT can execute the ConfirmedEventNotification service request.

Test Concept: BACnet does not define any action to be taken upon receipt of a ConfirmedEventNotification except to return an acknowledgement. Although returning an acknowledgment is sufficient to conform to the standard, a vendor may specify additional actions that can be observed. Any vendor-defined actions that do not occur during the tests shall be noted in the test report. No negative tests are included.

#### 9.4.1 ConfirmedEventNotification Using the Time Form of the 'Timestamp' Parameter and Conveying a Message Text

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,

|                                  |                                                       |
|----------------------------------|-------------------------------------------------------|
| 'Process Identifier' =           | (any valid process identifier),                       |
| 'Initiating Device Identifier' = | TD,                                                   |
| 'Event Object Identifier' =      | (any Event Enrollment object),                        |
| 'Time Stamp' =                   | (current time using the Time format),                 |
| 'Notification Class' =           | (any valid notification class),                       |
| 'Priority' =                     | (any valid priority),                                 |
| 'Event Type' =                   | (any standard event type),                            |
| 'Message Text' =                 | (any character string),                               |
| 'Notify Type' =                  | ALARM   EVENT,                                        |
| 'AckRequired' =                  | FALSE,                                                |
| 'From State' =                   | NORMAL,                                               |
| 'To State' =                     | (any non-normal state appropriate to the event type), |
| 'Event Values' =                 | (any values appropriate to the event type)            |

2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

#### 9.4.2 ConfirmedEventNotification Using the DateTime Form of the 'Timestamp' Parameter and no Message Text

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                       |
|----------------------------------|-------------------------------------------------------|
| 'Process Identifier' =           | (any valid process identifier),                       |
| 'Initiating Device Identifier' = | TD,                                                   |
| 'Event Object Identifier' =      | (any Event Enrollment object),                        |
| 'Time Stamp' =                   | (current time using the DateTime format),             |
| 'Notification Class' =           | (any valid notification class),                       |
| 'Priority' =                     | (any valid priority),                                 |
| 'Event Type' =                   | (any standard event type),                            |
| 'Notify Type' =                  | ALARM   EVENT,                                        |
| 'AckRequired' =                  | FALSE,                                                |
| 'From State' =                   | NORMAL,                                               |
| 'To State' =                     | (any non-normal state appropriate to the event type), |
| 'Event Values' =                 | (any values appropriate to the event type)            |
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

#### 9.4.3 ConfirmedEventNotification Using the Sequence Number Form of the 'Timestamp' Parameter and no Message Text

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                       |
|----------------------------------|-------------------------------------------------------|
| 'Process Identifier' =           | (any valid process identifier),                       |
| 'Initiating Device Identifier' = | TD,                                                   |
| 'Event Object Identifier' =      | (any Event Enrollment object),                        |
| 'Time Stamp' =                   | (current sequence number),                            |
| 'Notification Class' =           | (any valid notification class),                       |
| 'Priority' =                     | (any valid priority),                                 |
| 'Event Type' =                   | (any standard event type),                            |
| 'Notify Type' =                  | ALARM   EVENT,                                        |
| 'AckRequired' =                  | FALSE,                                                |
| 'From State' =                   | NORMAL,                                               |
| 'To State' =                     | (any non-normal state appropriate to the event type), |
| 'Event Values' =                 | (any values appropriate to the event type)            |
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

#### 9.4.4 Deleted Clause

This test has been removed.

**9.4.5 ConfirmedEventNotification Simple Presentation**

Purpose: This test case verifies that the IUT is capable of minimally displaying ConfirmedEventNotifications.

Configuration: For this test, the tester shall choose one event-generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                                     |
|----------------------------------|---------------------------------------------------------------------|
| 'Process Identifier' =           | (a valid process identifier specified by the IUT vendor),           |
| 'Initiating Device Identifier' = | TD,                                                                 |
| 'Event Object Identifier' =      | O1,                                                                 |
| 'Time Stamp' =                   | (any valid time stamp),                                             |
| 'Notification Class' =           | (any valid notification class),                                     |
| 'Priority' =                     | (any valid priority),                                               |
| 'Event Type' =                   | (any standard event type),                                          |
| 'Message Text' =                 | (any character string),                                             |
| 'Notify Type' =                  | ALARM   EVENT,                                                      |
| 'AckRequired' =                  | TRUE   FALSE,                                                       |
| 'From State' =                   | (state S1, any valid state for this event type),                    |
| 'To State' =                     | (state S2, any valid state for this event type that can follow S1), |
| 'Event Values' =                 | (any values appropriate to the event type),                         |
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, and the event Message Text)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

Note to Tester: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 32 characters. The IUT shall not truncate Message Text that is less than or equal to 32 characters in length. A device shall not fail to process an EventNotification service request containing a 'Message Text' parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.

**9.4.6 ConfirmedEventNotification Full Presentation**

Purpose: This test case verifies that the IUT is capable of displaying ConfirmedEventNotifications.

Configuration: For this test, the tester shall choose one event generating object, O1.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                                     |
|----------------------------------|---------------------------------------------------------------------|
| 'Process Identifier' =           | (a valid process identifier specified by the IUT vendor),           |
| 'Initiating Device Identifier' = | TD,                                                                 |
| 'Event Object Identifier' =      | O1,                                                                 |
| 'Time Stamp' =                   | (any valid time stamp),                                             |
| 'Notification Class' =           | (any valid notification class),                                     |
| 'Priority' =                     | (any valid priority),                                               |
| 'Event Type' =                   | (any standard event type),                                          |
| 'Message Text' =                 | (any character string),                                             |
| 'Notify Type' =                  | ALARM   EVENT,                                                      |
| 'AckRequired' =                  | TRUE   FALSE,                                                       |
| 'From State' =                   | (state S1, any valid state for this event type),                    |
| 'To State' =                     | (state S2, any valid state for this event type that can follow S2), |
| 'Event Values' =                 | (any values appropriate to the event type)                          |
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (that the IUT indicates the notification to the operator and that the indication includes identification of the event generating object or the monitored object, the event timestamp, the event Message Text, Notification Class, Priority, Notify Type, Ack Required, To State and Event Values)
4. CHECK (that all information indicated to the user is consistent with the information provided in step 1)

Note to Tester: The IUT shall truncate the message text if it is longer than the maximum length displayable by the IUT. The IUT is allowed to include characters in the displayed text that indicate the message has been truncated, even if the truncated message is then shorter than 255 characters. The IUT shall not truncate Message Text that is less than or equal to 255 characters in length. A device shall not fail to process an EventNotification service request containing a 'Message Text' parameter in an unsupported character set. It is a local matter whether the parameter is used as provided or whether a character string, in a supported character set, of length 0 is used in its place.

#### 9.4.7 Unsupported Message Text Character Set ConfirmedEventNotification Test

Purpose: To verify that the IUT correctly receives and processes ConfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT returns a Result(+) and performs the vendor specified actions.

Configuration Requirements: Configure the TD as though it has an event-initiating object, O1, which references a Notification Class object N1. Configure N1 to direct notifications to the IUT using a vendor specified Process Id, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps:

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                |
|----------------------------------|------------------------------------------------|
| 'Process Identifier' =           | PID1,                                          |
| 'Initiating Device Identifier' = | TD,                                            |
| 'Event Object Identifier' =      | O1,                                            |
| 'Time Stamp' =                   | (any valid timestamp),                         |
| 'Notification Class' =           | (N1: the Notification Class configured in O1), |
| 'Priority' =                     | (any valid priority),                          |
| 'Event Type' =                   | (the standard event type associated with O1),  |
| 'Message Text' =                 | T1,                                            |
| 'Notify Type' =                  | ALARM / EVENT,                                 |
| 'AckRequired' =                  | FALSE,                                         |
| 'From State' =                   | (any valid event state),                       |
| 'To State' =                     | (any valid event state),                       |
| 'Event Values' =                 | (any values appropriate to the event type)     |
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (for any vendor-defined observable actions)

#### 9.4.8 Decoding BACnetPropertyStates in 'Event Values'

Purpose: To verify that the IUT is correctly accepting 'Event Values', which contain BACnetPropertyStates values across the full value range of context tags, ensuring proper decoding of the various forms of the production.

Test Concept: Send 3 ConfirmedEventNotifications to the IUT conveying a CHANGE\_OF\_STATE event. After each notification verify that the IUT accepts and processes the notification. The first notification is sent with a new-state, NS1, having a context tag value in the range 0 .. 62. The second notification is sent with a new-state, NS2, having a context tag value in the range 64 .. 253 (a vendor proprietary discrete datatype). The third notification is sent with a new-state, NS3, having a context tag value 254 (a standard discrete datatype) or greater and encoded with a context tag of 63 (the extended-value choice) using the special encoding rules defined in the comment at the end of the BACnetPropertyStates production in Clause 21.

Test Steps:

-- new-state with a tag value in the range 15 .. 62

1. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                 |
|----------------------------------|---------------------------------|
| 'Process Identifier' =           | (any valid process identifier), |
| 'Initiating Device Identifier' = | TD,                             |
| 'Event Object Identifier' =      | (any valid value),              |
| 'Time Stamp' =                   | (any valid time stamp),         |
| 'Notification Class' =           | (any valid value),              |

- 'Priority' = (any valid value),  
 'Event Type' = CHANGE\_OF\_STATE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = (any valid normal or offnormal value),  
 'To State' = (any valid normal or offnormal value),  
 'Event Values' = (NS1, (T,F,?,?))
2. RECEIVE BACnet-SimpleACK-PDU
  3. CHECK (that the IUT has utilized the value conveyed, correctly decoded)
- new-state with a tag value in the range 64 .. 253
4. TRANSMIT ConfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = TD,  
 'Event Object Identifier' = (any valid value),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (any valid value),  
 'Priority' = (any valid value),  
 'Event Type' = CHANGE\_OF\_STATE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = (any valid normal or offnormal value),  
 'To State' = (any valid normal or offnormal value),  
 'Event Values' = (NS2, (T,F,?,?))
  5. RECEIVE BACnet-SimpleACK-PDU
  6. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are utilized the value conveyed, correctly decoded)
- new-state with a tag value greater than 254,
7. TRANSMIT ConfirmedEventNotification-Request,  
 'Process Identifier' = (any valid process identifier),  
 'Initiating Device Identifier' = TD,  
 'Event Object Identifier' = (any valid value),  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = (any valid value),  
 'Priority' = (any valid value),  
 'Event Type' = CHANGE\_OF\_STATE,  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = ALARM | EVENT,  
 'AckRequired' = TRUE | FALSE,  
 'From State' = (any valid normal or offnormal value),  
 'To State' = (any valid normal or offnormal value),  
 'Event Values' = (NS3, (T,F,?,?))
  8. RECEIVE BACnet-SimpleACK-PDU
  9. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

## 9.5 UnconfirmedEventNotification Service Execution Tests

### 9.5.1 UnconfirmedEventNotification Simple Presentation

This test is identical to the one in Clause 9.4.5 ConfirmedEventNotification Simple Presentation, except that an UnconfirmedEventNotification is sent in step 1 and that no BACnet-SimpleACK-PDU is returned by the IUT in step 2.

### 9.5.2 UnconfirmedEventNotification Full Presentation

This test is identical to the one in Clause 9.4.6 ConfirmedEventNotification Simple Presentation, except that an UnconfirmedEventNotification is sent in step 1 and that no BACnet-SimpleACK-PDU is returned by the IUT in step 2.



### 9.5.3 Unsupported Message Text Character Set UnconfirmedEventNotification Test

Purpose: To verify that the IUT correctly receives and processes UnconfirmedEventNotifications when the Message Text parameter is of a character set that the IUT does not support.

Test Concept: Send a notification to the IUT, from an event-initiating object, O1, which contains a Message Text parameter value, T1, which uses a character set that the IUT does not support. Verify that the IUT processes the request even if the 'Message Text' uses a character set that the IUT does not support, and that the IUT performs the vendor specified actions.

Configuration Requirements: Configure TD to direct notifications to the IUT using a vendor specified Process Identifier, PID1. If the IUT supports all character sets, this test shall be skipped.

Test Steps: The test steps for this test case are identical to the test steps in 9.4.7 except that the UnconfirmedEventNotification requests are used instead of ConfirmedEventNotification requests and the IUT does not acknowledge receiving the notifications.

### 9.5.4 Decoding BACnetPropertyStates in 'Event Values'

Purpose: To verify that the IUT is correctly accepting 'Event Values', which contain BACnetPropertyStates values across the full value range of context tags, ensuring proper decoding of the various forms of the production.

Test Concept: Send 3 UnconfirmedEventNotifications to the IUT conveying a CHANGE\_OF\_STATE event. After each notification verify that the IUT accepts and processes the notification. The first notification is sent with a new-state, NS1, having a context tag value in the range 0 .. 62. The second notification is sent with a new-state, NS2, having a context tag value in the range 64 .. 253 (a vendor proprietary discrete datatype). The third notification is sent with a new-state, NS3, having a context tag value 254 (a standard discrete datatype) or greater and encoded with a context tag of 63 (the extended-value choice) using the special encoding rules defined in the comment at the end of the BACnetPropertyStates production in Clause 21.

Test Steps:

-- new-state with a tag value in the range 15 .. 62

1. TRANSMIT UnconfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid value),
  - 'Time Stamp' = (any valid time stamp),
  - 'Notification Class' = (any valid value),
  - 'Priority' = (any valid value),
  - 'Event Type' = CHANGE\_OF\_STATE,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,
  - 'From State' = (any valid normal or offnormal value),
  - 'To State' = (any valid normal or offnormal value),
  - 'Event Values' = (NS1, (T,F,?,?))
2. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

-- new-state with a tag value in the range 64 .. 253

3. TRANSMIT UnconfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid value),
  - 'Time Stamp' = (any valid time stamp),
  - 'Notification Class' = (any valid value),
  - 'Priority' = (any valid value),
  - 'Event Type' = CHANGE\_OF\_STATE,
  - 'Message Text' = (optional, any valid message text),
  - 'Notify Type' = ALARM | EVENT,
  - 'AckRequired' = TRUE | FALSE,

'From State' = (any valid normal or offnormal value),

'To State' = (any valid normal or offnormal value),

'Event Values' = (NS2, (T,F,?,?))

4. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

-- new-state with a tag value greater than 254,

5. TRANSMIT UnconfirmedEventNotification-Request,

'Process Identifier' = (any valid process identifier),

'Initiating Device Identifier' = TD,

'Event Object Identifier' = (any valid value),

'Time Stamp' = (any valid time stamp),

'Notification Class' = (any valid value),

'Priority' = (any valid value),

'Event Type' = CHANGE\_OF\_STATE,

'Message Text' = (optional, any valid message text),

'Notify Type' = ALARM | EVENT,

'AckRequired' = TRUE | FALSE,

'From State' = (any valid normal or offnormal value),

'To State' = (any valid normal or offnormal value),

'Event Values' = (NS3, (T,F,?,?))

6. CHECK (that the IUT has correctly decoded the value by examining exposed actions related to the receipt of the event notification, if there are any)

## 9.6 GetAlarmSummary Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing GetAlarmSummary service requests.

### 9.6.1 Alarm Summaries with no Active Alarms

Purpose: To verify that the IUT can execute the GetAlarmSummary service request when there are no active alarms to report.

Configuration Requirements: The IUT shall be configured so that there are no active alarms.

Test Steps:

1. TRANSMIT GetAlarmSummary-Request
2. RECEIVE GetAlarmSummary-ACK,  
'List of Alarm Summaries' = (an empty list)

### 9.6.2 Alarm Summaries with One Active Alarm

Purpose: To verify that the IUT can execute the GetAlarmSummary service request when there is exactly one active alarm to report.

Configuration Requirements: The IUT shall be configured so that there is exactly one active alarm.

Test Steps:

1. TRANSMIT GetAlarmSummary-Request
2. RECEIVE GetAlarmSummary-ACK,  
'List of Alarm Summaries' = (a list with exactly one entry corresponding to the known active alarm)

### 9.6.3 Alarm Summaries with Multiple Active Alarms

Purpose: To verify that the IUT can execute the GetAlarmSummary service request when there are multiple active alarms to report. This test case shall be executed only for devices that contain more than one object that can detect alarms.

Configuration Requirements: The IUT shall be configured so that there is more than one active alarm.

Test Steps:

1. TRANSMIT GetAlarmSummary-Request

2. RECEIVE GetAlarmSummary-ACK,  
'List of Alarm Summaries' = (a list containing one entry for each known active alarm)

### 9.7 GetEnrollmentSummary Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing GetEnrollmentSummary service requests.

Test Concept: This service has 6 filters that can be applied to the selection of the event enrollments. Of the 6 filters, only the 'Acknowledgment Filter' is required. For each test case a particular configuration is required that will produce the expected result for the filters used. The test cases in 9.7.1 utilize only the required 'Acknowledgment Filter'. The IUT shall be configured for each of these test cases as described. The test cases in 9.7.2 utilize the optional filters. The intention is to configure the IUT with a rich database of event enrollments that have features appropriate to these filters. This may not be possible for some implementations. It is not necessary to support a configuration that provides event enrollments that match each of the filter criteria. However, it is a requirement that all of the tests be executed and a response returned that is appropriate based on the configuration of the IUT's database. Returning an empty list of enrollments is appropriate if it is not possible to configure the IUT in such a way that the filter criteria can be met. There are no negative tests.

#### 9.7.1 Required GetEnrollmentSummary Filters

Purpose: This test group is to verify the correct execution of the GetEnrollmentSummary service request under the circumstances where the service is expected to be successfully completed and only the required 'Acknowledgment Filter' is used.

##### 9.7.1.1 Enrollment Summary with Zero Summaries

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when there are no enrollments to report.

Configuration Requirements: The IUT shall be configured with no enrollments to report.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = NOT\_ACKED
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (an empty list)

Notes to Tester: If the IUT cannot be configured with no enrollments to report, then the GetEnrollmentSummary-Request shall be transmitted with a further constrained argument so that the resulting filtered enrollment summary yields zero summaries.

##### 9.7.1.2 ACKED

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the Acknowledgement Filter is set to ACKED

Configuration Requirements: The IUT shall be configured with at least two event enrollments, one for which all event transitions have been acknowledged, and one for which at least one event transition has not been acknowledged.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ACKED
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all events for which the event transitions have all been acknowledged)

##### 9.7.1.3 NOT-ACKED

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the Acknowledgement Filter is set to NOT-ACKED.

Configuration Requirements: The IUT shall be configured with at least two event enrollments, one for which all event transitions have been acknowledged, and one for which at least one event transition has not been acknowledged.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = NOT-ACKED
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all events for which there is at least one unacknowledged event transition)

#### 9.7.1.4 All

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the filter request in the Acknowledgement Filter is set to ALL.

Configuration Requirements: The IUT shall be configured with at least two event enrollments, one for which all event transitions have been acknowledged, and one for which at least one event transition has not been acknowledged.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (the union of the summaries provided in 9.7.1.2 and 9.7.1.3)

### 9.7.2 User Selectable GetEnrollmentSummary Filters

Purpose: This test group is to verify the correct execution of the GetEnrollmentSummary service request under the circumstances where the service is expected to be successfully completed and user selectable filters are used.

#### 9.7.2.1 Enrollment Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary when the 'Enrollment Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured so that there are at least two Enrollment Summaries to report with different (BACnetRecipient, Process Identifier) pairs. The TD will use one of these combinations in the GetEnrollmentSummary service request.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL,  
'Enrollment Filter' = (one of the (BACnetRecipient, Process Identifier) pairs configured for this test)
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all enrollments configured with this (BACnetRecipient, Process Identifier) pair)

#### 9.7.2.2 Event State Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Event State Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects that have an Event\_State property value of NORMAL, one or more with an Event\_State property value of FAULT, one or more with an Event\_State property value of OFFNORMAL, and one or more with an Event\_State property value that is not NORMAL, OFFNORMAL, or FAULT. If only a subset of these cases can be supported as many of them as possible shall be configured.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL,  
'Event State Filter' = NORMAL
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects with pCurrentState = NORMAL)

3. TRANSMIT GetEnrollmentSummary-Request,  
     'Acknowledgment Filter' = ALL,  
     'Event State Filter' = FAULT
4. RECEIVE GetEnrollmentSummary-ACK,  
     'List of Enrollment Summaries' = (all configured event-generating objects with pCurrentState = FAULT)
5. TRANSMIT GetEnrollmentSummary-Request,  
     'Acknowledgment Filter' = ALL,  
     'Event State Filter' = OFFNORMAL
6. RECEIVE GetEnrollmentSummary-ACK,  
     'List of Enrollment Summaries' = (all configured event-generating objects with pCurrentState = OFFNORMAL)
7. TRANSMIT GetEnrollmentSummary-Request,  
     'Acknowledgment Filter' = ALL,  
     'Event State Filter' = ACTIVE
8. RECEIVE GetEnrollmentSummary-ACK,  
     'List of Enrollment Summaries' = (all configured event-generating objects with pCurrentState = a value other than  
     NORMAL)
9. TRANSMIT GetEnrollmentSummary-Request,  
     'Acknowledgment Filter' = ALL,  
     'Event State Filter' = ALL
10. RECEIVE GetEnrollmentSummary-ACK,  
     'List of Enrollment Summaries' = (the union of all of the summaries returned in steps 1 - 8)

#### 9.7.2.3 Event Type Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Event Type Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured so that it has one or more event-generating objects for each of its supported event types. If the IUT cannot be configured in such a way all at once, then the test shall be repeated so that each of its supported event types is tested. If only a subset of these event types are supported, as many of them as possible shall be configured.

Test Steps:

```

REPEAT Y = (All the configurations that will be tested) DO {
 REPEAT X = (All the Event Types currently configured) DO {
 TRANSMIT GetEnrollmentSummary-Request,
 'Acknowledgment Filter' = ALL,
 'Event Type Filter' = X
 RECEIVE GetEnrollmentSummary-ACK,
 'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = X)
 }
}

```

#### 9.7.2.4 Priority Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Priority Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured with one or more event-generating objects at each of four different priority levels. For the purpose of this test, the priority of an event-generating object is the priority of the object's most recent transition. The priority levels shall be 0,  $X_{low}$ ,  $X_{high}$ , 255, where  $10 < X_{low} < 100$  and  $100 < X_{high} < 255$ . If only a subset of these priorities can be supported at one time as many of them as possible shall be configured.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
     'Acknowledgment Filter' = ALL,  
     'MinPriority' = 0,  
     'MaxPriority' = 0

2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects with a priority in the specified range)
3. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL,  
'MinPriority' = 0,  
'MaxPriority' = 255
4. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects with a priority in the specified range)
5. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL,  
'MinPriority' =  $X_{low}$ ,  
'MaxPriority' =  $X_{high}$
6. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects with a priority in the specified range)

#### 9.7.2.5 Notification Class Filter

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when the 'Notification Class Filter' is used.

Configuration Requirements: If possible, the IUT shall be configured with one or more event-generating objects using each of two notification classes.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL,  
'Notification Class Filter' = (any of the configured notification classes)
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects using the specified notification class)

#### 9.7.2.6 A Combination of Filters

Purpose: To verify that the IUT can execute the GetEnrollmentSummary request when a combination of user selectable filters is used.

Configuration Requirements: Any combination of event-generating object configurations defined in 9.7.2.1 – 9.7.2.5 is acceptable.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,  
'Acknowledgment Filter' = ALL | ACKED | NOT\_ACKED,  
(any combination of user selectable filters chosen by the TD with appropriate values),
2. RECEIVE GetEnrollmentSummary-ACK,  
'List of Enrollment Summaries' = (all configured event-generating objects matching all of the filter requirements)

### 9.8 GetEventInformation Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing GetEventInformation service requests.

#### 9.8.1 Event Information with no Active Events

Purpose: To verify that the IUT can execute the GetEventInformation service request when there are no active events to report.

Configuration Requirements: The IUT shall be configured so that there are no active event states.



Test Steps:

1. TRANSMIT GetEventInformation-Request
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (an empty list),  
'More Events' = FALSE

#### 9.8.2 Event Information with one Active Event

Purpose: To verify that the IUT can execute the GetEventInformation service request when there is exactly one active event to report.

Configuration Requirements: The IUT shall be configured so that there is exactly one active event state.

Test Steps:

1. TRANSMIT GetEventInformation-Request
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (a list with exactly one entry corresponding to the known active event),  
'More Events' = FALSE

#### 9.8.3 Event Information with Multiple Active Events

Purpose: To verify that the IUT can execute the GetEventInformation service request when there are multiple active event states to report. This test case shall be executed only for devices that contain more than one object that can detect alarms.

Configuration Requirements: The IUT shall be configured so that there are more than one active event states, but fewer than would require transmission with 'More Events' = TRUE. If the IUT cannot be configured to contain multiple active events which can be transmitted in a single GetEventInformation service response without 'More Events' = TRUE, then this test shall be skipped.

Test Steps:

1. TRANSMIT GetEventInformation-Request
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (a list containing one entry for each known active event state),  
'More Events' = FALSE

#### 9.8.4 Event Information Based on Event\_State

Purpose: To verify that the IUT can execute the GetEventInformation service request when the active event state is caused by the Event\_State property, and not by unacknowledged event transitions.

Configuration Requirements: The IUT shall be configured so that there is only one active event state, that state is caused by the Event\_State property being not NORMAL, and the Acknowledged\_Transitions property having all bits set to TRUE.

Test Steps:

1. TRANSMIT GetEventInformation-Request
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (a list with exactly one entry corresponding to the known active event state),  
'More Events' = FALSE

#### 9.8.5 Event Information Based on Acknowledged\_Transitions

Purpose: To verify that the IUT can execute the GetEventInformation service request when the active event state is caused by unacknowledged event transitions, and not by the Event\_State property.

Configuration Requirements: The IUT shall be configured so that there is only one active event state, and that state is caused by the Event\_State property being NORMAL, and the Acknowledged\_Transitions property having at least one bit set to FALSE.

Test Steps:

1. TRANSMIT GetEventInformation-Request
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (a list with exactly one entry corresponding to the known active event state),  
'More Events' = FALSE

#### 9.8.6 Chaining Test

Purpose: This test case exercises the chaining capabilities using multiple GetEventInformation messages.

Configuration Requirements: The IUT shall be configured so that there are more event states than can be conveyed in a single APDU of 128 bytes. The IUT shall be configured to contain enough events to trigger the chaining effect. If the IUT cannot be configured to contain enough active events to trigger chaining, then this test shall be skipped.

Test Concept: In steps 1-4, the test first tests proper chaining by requesting two lists from the IUT and verifying that the second list is properly distinct from the first. In steps 5-9, to test the “fixed object processing order” as defined in BACnet 13.12.1.1.1, it requests the first list again, and then, before requesting the second list, the tester makes the last object in the first list no longer have any active event states. When the TD requests the second list using the object identifier of the now-normal device, the IUT should respond with the same second list as it did before.

Test Steps:

1. TRANSMIT GetEventInformation-Request,  
'max-APDU-length-accepted' = B'0001',  
'segmented-response-accepted' = FALSE
2. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (an arbitrary list),  
'More Events' = TRUE
3. TRANSMIT GetEventInformation-Request,  
'Last Received Object Identifier' = the last object identifier of the list received in step 2)
4. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (a list of object identifiers not including any received in step 2)
5. TRANSMIT GetEventInformation-Request,  
'max-APDU-length-accepted' = B'0001',  
'segmented-response-accepted' = FALSE
6. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (an arbitrary list),  
'More Events' = TRUE
7. MAKE (the object identified by the last object identifier in the list received in step 6 have no active event states)
8. TRANSMIT GetEventInformation-Request,  
'Last Received Object Identifier' = (the last object identifier of the list received in step 6)
9. RECEIVE GetEventInformation-ACK,  
'List of Event Summaries' = (the same list received in step 4)

### 9.9 LifeSafetyOperation Service Execution Test

This clause defines the tests necessary to demonstrate support for executing LifeSafetyOperation service requests.

#### 9.9.1 Positive LifeSafetyOperation Execution Tests

##### 9.9.1.1 Reset Single Object Execution Tests

Purpose: To verify that the IUT correctly resets a latched life safety object when executing a LifeSafetyOperation service request directed at the life safety object.

Test Concept: A life safety object, which latches its Present\_Value, is toggled between a normal and non-normal BACnetLifeSafetyState. The object's Present\_Value should latch in the non-normal state until the LifeSafetyOperation service request is executed. This is repeated for each LifeSafetyOperation request type supported by the device.

Configuration Requirements: The Life-Safety object starts the test with its Present\_Value in a normal state as interpreted by the life safety object. This test shall be skipped if the device does not support latching.

Test Steps:

```

REPEAT X = (All supported enumerations that reset the object) DO {
1. MAKE (the selected object enters a latched non-normal state where enumeration X will reset the object)
2. VERIFY Present_Value = (S1: a non-NORMAL state)
3. VERIFY Tracking_Value = S1
4. MAKE (remove the non-normal condition)
5. VERIFY Present_Value = S1
6. VERIFY Tracking_Value = (S2: a NORMAL state)
7. TRANSMIT LifeSafetyOperation-Request,
 'Requesting Process Identifier' = (any valid identifier),
 'Requesting Source' = (any valid character string),
 'Request' = X,
 'Object Identifier' = (the selected object)
8. RECEIVE BACnet-SimpleACK-PDU
9. VERIFY (Object), Present_Value = S2
}

```

### 9.9.1.2 Reset Multiple Object Execution Tests

Purpose: To verify that the IUT correctly resets multiple latched life safety objects when executing a LifeSafetyOperation service request not directed at a specified life safety object.

Test Concept: Two Multiple Life Safety objects, O1 ... On, are toggled between a normal and non-normal BACnetLifeSafetyState. The objects' Present\_Value properties should latch in the non-normal state until the LifeSafetyOperation service request is executed. This is repeated for each LifeSafetyOperation request type supported by the device.

Configuration Requirements: The Life-Safety objects start the test with their Present\_Value properties in a normal state as interpreted by the life safety objects. This test shall be skipped if the device does not support latching Life Safety objects. If the IUT supports a single latching life safety object, apply this test to the single object.

Test Steps:

```

REPEAT X = (All supported enumerations that reset the objects) DO {
1. MAKE (the selected objects enter a latched non-normal states where enumeration X will reset the objects)
2. REPEAT Oi = (each selected life safety object) {
 VERIFY Oi, Present_Value = (Si: a non-normal value)
 VERIFY Oi, Tracking_Value = Si
 }
3. MAKE (remove the non-normal conditions for each selected life safety object)
4. REPEAT Oi = (each selected life safety object) DO {
 VERIFY Oi, Present_Value = Si
 VERIFY Oi, Tracking_Value = (Tvi, a normal value)
 }
5. TRANSMIT LifeSafetyOperation-Request,
 'Requesting Process Identifier' = (any valid identifier),
 'Requesting Source' = (any valid character string),
 'Request' = X,
6. RECEIVE BACnet-SimpleACK-PDU
7. REPEAT Oi = (each selected life safety object) DO {
 VERIFY Oi, Present_Value = TVi
 }
}

```

### 9.9.1.3 Silencing/Unsilencing Execution Tests

Purpose: To verify that the IUT can correctly execute a LifeSafetyOperation service request to silence and unsilence an alarming device.

Test Concept: An audible device and/or visual device is attached to the IUT and is sounding/flashing because a life safety object has entered a non-normal state and the property Silenced is UNSILENCED. A LifeSafetyOperation service request is transmitted to silence the sounder/strobe. Then, the life safety object remains in the non-normal state with Silenced equal to SILENCED. A LifeSafetyOperation service request is transmitted to unsilence the sounder/strobe (reactivate it), and it is verified that the object is unsilenced.

There are different allowable BACnetSilencedState values based on the silence operation performed and the setup of the IUT. In the below tables, N/A marks an operation that is inappropriate for the test with the corresponding IUT setup.

| Only Sounder Attached |                                             |          |                      |                         |          |
|-----------------------|---------------------------------------------|----------|----------------------|-------------------------|----------|
| Silence Operation     | Allowable State                             | Silenced | Unsilenced Operation | Allowable State         | Silenced |
| SILENCE               | ALL_SILENCED, AUDIBLE_SILENCED, proprietary |          | UNSILENCE            | UNSILENCED, proprietary |          |
| SILENCE_AUDIBLE       | ALL_SILENCED, AUDIBLE_SILENCED, proprietary |          | UNSILENCE_AUDIBLE    | UNSILENCED, proprietary |          |
| SILENCE_VISUAL        | N/A                                         |          | UNSILENCE_VISUAL     | N/A                     |          |

| Only Strobe Attached |                                            |          |                      |                         |          |
|----------------------|--------------------------------------------|----------|----------------------|-------------------------|----------|
| Silence Operation    | Allowable State                            | Silenced | Unsilenced Operation | Allowable State         | Silenced |
| SILENCE              | ALL_SILENCED, VISUAL_SILENCED, proprietary |          | UNSILENCE            | UNSILENCED, proprietary |          |
| SILENCE_AUDIBLE      | N/A                                        |          | UNSILENCE_AUDIBLE    | N/A                     |          |
| SILENCE_VISUAL       | ALL_SILENCED, VISUAL_SILENCED, proprietary |          | UNSILENCE_VISUAL     | UNSILENCED, proprietary |          |

| Sounder And Strobe Attached |                               |          |                                                     |                              |          |
|-----------------------------|-------------------------------|----------|-----------------------------------------------------|------------------------------|----------|
| Silence Operation           | Allowable State               | Silenced | Unsilenced Operation                                | Allowable State              | Silenced |
| SILENCE                     | ALL_SILENCED, proprietary     |          | UNSILENCE                                           | UNSILENCED, proprietary      |          |
| SILENCE_AUDIBLE             | AUDIBLE_SILENCED, proprietary |          | UNSILENCE_AUDIBLE (all silenced)                    | SILENCED_VISUAL, proprietary |          |
|                             |                               |          | UNSILENCE_AUDIBLE (audible silenced, visual active) | UNSILENCED, proprietary      |          |
|                             |                               |          | UNSILENCE_AUDIBLE (audible active, visual silenced) | N/A                          |          |

|                |                                 |                                                          |                                  |
|----------------|---------------------------------|----------------------------------------------------------|----------------------------------|
| SILENCE_VISUAL | VISUAL_SILENCED,<br>proprietary | UNSILENCE_VISUAL<br>(all silenced)                       | SILENCED_AUDIBLE,<br>proprietary |
|                |                                 | UNSILENCE_VISUAL<br>(audible silenced,<br>visual active) | N/A                              |
|                |                                 | UNSILENCE_VISUAL<br>(audible active, visual<br>silenced) | UNSILENCED,<br>proprietary       |

Configuration Requirements: The IUT must be fitted with needed audible and visual equipment.

Notes to Tester: Source object needs to get silence only for configured objects.

Test Steps:

REPEAT X = (All supported enumerations that silence the object) DO {

1. MAKE (the selected object enter a state where enumeration X will silence the sounder/strobe)
  2. VERIFY Silenced = (Unsilenced or a proprietary value with a similar semantic)
  3. TRANSMIT LifeSafetyOperation-Request,  
    'Requesting Process Identifier' = (any valid identifier),  
    'Requesting Source' = any valid character string,  
    'Request' = X,  
    'Object Identifier' = (absent or the selected object)
  4. RECEIVE BACnet-SimpleACK-PDU
  5. CHECK (that the sounder/strobe is inactive)
  6. VERIFY Silenced = (an allowable silenced state based on the IUT setup and operation request X)
  7. TRANSMIT LifeSafetyOperation-Request,  
    'Requesting Process Identifier' = (any valid identifier),  
    'Requesting Source' = (any valid character string),  
    'Request' = (any valid LifeSafetyOperation request),  
    'Object Identifier' = (the selected object)
  8. RECEIVE BACnet-SimpleACK-PDU
  9. CHECK (the sounder / strobe active again, as appropriate to the operation)
  10. VERIFY Silenced = (the appropriate state based on the operation and IUT condition)
- }

## 9.9.2 Negative LifeSafetyOperation Execution Tests

### 9.9.2.1 LifeSafetyOperation for an Object Which Does Not Exist

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation targets a non-existent object.

Test Concept: Send a LifeSafetyOperation request to the IUT targeting an object that does not exist in the IUT.

Test Steps:

1. TRANSMIT LifeSafetyOperation-Request,  
    'Requesting Process Identifier' = (any valid value),  
    'Requesting Source' = (any valid value),  
    'Request' = (any valid LifeSafetyOperation request supported by the device),  
    'Object Identifier' = (any object not contained in the IUT's database)
2. RECEIVE BACnet-Error PDU,  
    'Error Class' = OBJECT,  
    'Error Code' = UNKNOWN\_OBJECT

### 9.9.2.2 LifeSafetyOperation which is Invalid given the Object's Current State

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation's Request value is invalid given the object's current state.

Test Concept: Send a LifeSafetyOperation request, with a Request value that is not valid for the current state, to the IUT.

Configuration Requirements: The life safety object, O1, being tested is placed into a state where Request R, a valid LifeSafetyOperation value which the life safety object would accept in other states, is currently invalid. If there is no such state, request value pair that satisfies this requirement, this test shall be skipped.

Test Steps:

1. TRANSMIT LifeSafetyOperation-Request,
  - 'Requesting Process Identifier' = (any valid value),
  - 'Requesting Source' = (any valid value),
  - 'Request' = (R: a request value which is invalid given the life safety object's current state),
  - 'Object Identifier' = O1
2. RECEIVE BACnet-Error PDU,
  - 'Error Class' = OBJECT,
  - 'Error Code' = INVALID\_OPERATION\_IN\_THIS\_STATE

### 9.9.2.3 LifeSafetyOperation On An Object Which Does Not Support It

Purpose: To verify that the IUT correctly responds when a LifeSafetyOperation's is received that targets an object that does not support it.

Test Concept: Send a LifeSafetyOperation request, with an Object Identifier referencing an object in the IUT which does not support life safety operations.

Test Steps:

1. TRANSMIT LifeSafetyOperation-Request,
  - 'Requesting Process Identifier' = (any valid value),
  - 'Requesting Source' = (any valid value),
  - 'Request' = (any valid value normally supported by the IUT),
  - 'Object Identifier' = (an object in the IUT which does not support life safety operations)
2. RECEIVE BACnet-Error PDU,
  - 'Error Class' = OBJECT,
  - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

## 9.10 SubscribeCOV Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing SubscribeCOV service requests.

Configuration Requirements: The IUT shall be configured with at least one object that supports subscriptions for COV notifications.

### 9.10.1 Positive SubscribeCOV Service Execution Tests

The purpose of this test group is to verify the correct execution of the SubscribeCOV service request under circumstances where the service is expected to be successfully completed.

#### 9.10.1.1 Confirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription for confirmed COV notifications. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.10.1.2.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (any value > 0 if automatic cancellation is supported, otherwise 0)
2. RECEIVE BACnet-SimpleACK-PDU



**3. BEFORE Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
 'List of Values' = (values appropriate to the object type of the monitored object)

**4. TRANSMIT BACnet-SimpleACK-PDU****9.10.1.2 Unconfirmed COV Notifications**

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription for unconfirmed COV notifications. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.10.1.1.

Test Steps:

**1. TRANSMIT SubscribeCOV-Request,**

'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = (any value > 0 if automatic cancellation is supported, otherwise 0)

0)

**2. RECEIVE BACnet-SimpleACK-PDU****3. BEFORE Notification Fail Time**

RECEIVE UnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
 'List of Values' = (values appropriate to the object type of the monitored object)

**9.10.1.3 Explicit Indefinite Lifetime COV Subscriptions**

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with an indefinite lifetime (lifetime = 0). Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Steps:

**1. TRANSMIT SubscribeCOV-Request,**

'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = 0

**2. RECEIVE BACnet-SimpleACK-PDU****3. IF (the subscription was for confirmed notifications) THEN****BEFORE Notification Fail Time**

RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = 0,  
 'List of Values' = (values appropriate to the object type of the monitored object)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

**BEFORE Notification Fail Time**

RECEIVE UnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = 0,

'List of Values' = (values appropriate to the object type of the monitored object)

4. MAKE (a change to the monitored object that should cause a COV notification)
5. IF (the subscription was for confirmed notifications) THEN
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = (the same object used in the subscription),
      - 'Time Remaining' = 0,
      - 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - BEFORE **Notification Fail Time**
      - RECEIVE UnconfirmedCOVNotification-Request,
        - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
        - 'Initiating Device Identifier' = IUT,
        - 'Monitored Object Identifier' = (the same object used in the subscription),
        - 'Time Remaining' = 0,
        - 'List of Values' = (values appropriate to the object type of the monitored object including the changed value of that triggered the notification)

#### 9.10.1.4 Canceling COV Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to cancel a COV subscription. This test cancels the subscription made in 9.10.1.1.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (the process identifier used in test 9.10.1.1),
  - 'Monitored Object Identifier' = (the same object used in test 9.10.1.1)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Notification Fail Time**
4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
5. CHECK (verify that the IUT did not transmit a COV notification message)

#### 9.10.1.5 Canceling Expired or Non-Existing Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to cancel a subscription that no longer exists.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any unused process identifier or an identifier from a previously terminated subscription),
  - 'Monitored Object Identifier' = (any unused object or an object from a previously terminated subscription)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Notification Fail Time**
4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
5. CHECK (verify that the IUT did not transmit a COV notification message)

#### 9.10.1.6 Implied Indefinite Lifetime COV Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with an implied indefinite lifetime (lifetime parameter omitted). Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Steps: The test steps are identical to 9.10.1.1 except that the 'Lifetime' parameter in step 1 shall be omitted.

Notes to Tester: The passing result is identical to the results in 9.10.1.1.

### 9.10.1.7 Finite Lifetime Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with a temporary lifetime. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (a value between 60 seconds and 300 seconds)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = (the same object used in the subscription),  
     'Time Remaining' = (A value approximately equal to, but not greater than, the requested subscription lifetime),  
     'List of Values' = (values appropriate to the object type of the monitored object)  
     TRANSMIT BACnet-SimpleACK-PDU  
   ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = (the same object used in the subscription),  
     'Time Remaining' = (A value approximately equal to, but not greater than, the requested subscription lifetime),  
     'List of Values' = (values appropriate to the object type of the monitored object)
4. MAKE (a change to the monitored object that cause a COV notification)
5. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = (the same object used in the subscription),  
     'Time Remaining' = (TR, a value greater than 0 and less than the requested subscription lifetime),  
     'List of Values' = (values appropriate to the object type of the monitored object)  
     TRANSMIT BACnet-SimpleACK-PDU  
   ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = (the same object used in the subscription),  
     'Time Remaining' = (TR, a value greater than 0 and less than the requested subscription lifetime),  
     'List of Values' = (values appropriate to the object type of the monitored object including the changed value of that triggered the notification)
6. WAIT (a time that should change the 'Time Remaining' and which is less than the lifetime of the subscription)
7. MAKE (a change to the monitored object that causes a COV notification)
8. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the TR),  
 'List of Values' = (values appropriate to the object type of the monitored object)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

BEFORE **Notification Fail Time**

RECEIVE UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than TR),  
 'List of Values' = (values appropriate to the object type of the monitored object  
 including the changed value that triggered the notification)

9. WAIT (the lifetime of the subscription)
10. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
11. CHECK (verify that the IUT did not transmit a COV notification message)

#### 9.10.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to update the lifetime of a subscription. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Concept: A subscription for COV notifications is made for 60 seconds. Before that subscription has expired a second subscription is made for 300 seconds. When the notification is sent in response to the second subscription the lifetime is checked to verify that it is greater than 60 but less than 300 seconds.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
 'Subscriber Process Identifier' = (PID1, any valid process identifier),  
 'Monitored Object Identifier' = (O1, any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = 60
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (60 or less than 60),  
 'List of Values' = (values appropriate to the object type of the monitored object)  
 TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
 BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (~60, but not greater than 60),  
 'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT SubscribeCOV-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Monitored Object Identifier' = O1,  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = (T1, a value between 180 and 300 seconds)
5. RECEIVE BACnet-SimpleACK-PDU
6. IF (the subscription was for confirmed notifications) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedCOVNotification-Request,

```

'Subscriber Process Identifier' = PID1,
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O1,
'Time Remaining' = (~T1, but not greater than T1),
'List of Values' = (values appropriate to the object type of the monitored object)
TRANSMIT BACnet-SimpleACK-PDU
ELSE
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = PID1,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O1,
 'Time Remaining' = (~T1, but not greater than T1),
 'List of Values' = (values appropriate to the object type of the monitored object)

```

#### 9.10.1.9 Ensuring Subscription Lifetimes Are Not Affected By Time Changes

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription with a temporary lifetime and that the lifetime is not affected by TimeSynchronization or UTC-TimeSynchronization service requests.

Test Concept: The TD subscribes to an object that supports COV reporting in the IUT. The time is changed to D<sub>1</sub>, which is more than 300 seconds in the future or the past. The object is made to change such that a COV notification would be generated and it is verified that a COV notification is issued. The subscription is then allowed to expire and the object is changed again such that a COV notification would be generated, if a subscription were present. It is verified that no notification is generated.

Configuration Requirements: The IUT contains an object that supports COV reporting. If the IUT does not support TimeSynchronization or UTC-TimeSynchronization, then this test shall be omitted.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (a value between 60 seconds and 300 seconds)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - RECEIVE ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = (the same object used in the subscription),
      - 'Time Remaining' = (the requested subscription lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - RECEIVE UnconfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = (the same object used in the subscription),
      - 'Time Remaining' = (the requested subscription lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object)
4. MAKE (a change to the monitored object that should cause a COV notification)
5. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - RECEIVE ConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,

- 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object)  
 TRANSMIT BACnet-SimpleACK-PDU
- ELSE  
 RECEIVE UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object)
- object including the changed value of that triggered the notification)
6. TRANSMIT  
 DA = GLOBAL BROADCAST,  
 SA = TD,  
 BACnet-Unconfirmed-Request-PDU,  
 'Service Choice' = TimeSynchronization-Request,  
 date = D<sub>1</sub>,  
 time = D<sub>1</sub>
7. TRANSMIT  
 DA = GLOBAL BROADCAST,  
 SA = TD,  
 BACnet-Unconfirmed-Request-PDU,  
 'Service Choice' = UTC-TimeSynchronization-Request,  
 date = D<sub>1</sub>,  
 time = D<sub>1</sub>
8. MAKE (a change to the monitored object that should cause a COV notification)
9. BEFORE **Notification Fail Time**  
 IF (the subscription was for confirmed notifications) THEN  
 RECEIVE ConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object)  
 TRANSMIT BACnet-SimpleACK-PDU
- ELSE  
 RECEIVE UnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object)
- object including the changed value of that triggered the notification)
10. WAIT (the remaining lifetime of the subscription)
11. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
12. CHECK (verify that the IUT did not transmit a COV notification message)

#### 9.10.1.10 Accepts 8 Hour Lifetimes

Purpose: To verify that the IUT correctly accepts lifetimes of at least 8 hours. Either confirmed or unconfirmed notifications may be used, but at least one of these options shall be supported by the IUT.



Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = 28800
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     IF (the subscription was for confirmed notifications) THEN  
         RECEIVE ConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same identifier used in the subscription),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),  
         'Time Remaining' = (the requested subscription lifetime),  
         'List of Values' = (values appropriate to the object type of the monitored object)  
         TRANSMIT BACnet-SimpleACK-PDU  
     ELSE  
         RECEIVE UnconfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same identifier used in the subscription),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),  
         'Time Remaining' = (the requested subscription lifetime),  
         'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Monitored Object Identifier' = (the same object used in the subscription)

#### 9.10.1.11 Ensuring 5 Concurrent COV Subscribers

Purpose: This test case verifies that the IUT can support 5 concurrent subscriptions.

Test Concept: Have the TD subscribe with 5 different process identifiers, V1 through V5, and then check to ensure that 5 notifications are sent when the monitored object changes.

Notes to Tester: The notification in step 3 can be received in any order by the TD.

Test Steps

1. REPEAT (X=V1 to V5) DO {  
     TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = X,  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (any valid value that will allow the subscription to outlast the test)  
     RECEIVE BACnet-SimpleACK-PDU  
     IF (confirmed notifications were requested) THEN  
         BEFORE **Notification Fail Time**  
         RECEIVE ConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = X,  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),  
         'Time Remaining' = (any valid value),  
         'List of Values' = (the initial Present\_Value and initial Status\_Flags)  
         TRANSMIT BACnet-SimpleACK-PDU  
     ELSE  
         BEFORE **Notification Fail Time**  
         RECEIVE UnconfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = X,  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),

```

 'Time Remaining' = (any valid value),
 'List of Values' = (the initial Present_Value and initial Status_Flags)
 }
2. MAKE (Present_Value = any value that differs from "initial Present_Value" such that a COV notification is
generated)
3. REPEAT (X=V1 to V5) DO {
 IF (if confirmed notifications were requested) THEN
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = X,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = (the same object used in the subscription),
 'Time Remaining' = (any valid value),
 'List of Values' = (the new Present_Value and Status_Flags)
 TRANSMIT BACnet-SimpleACK-PDU
 ELSE
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = X,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = (the same object used in the subscription),
 'Time Remaining' = (any valid value),
 'List of Values' = (the new Present_Value and Status_Flags)
 }
}

```

### 9.10.2 Negative SubscribeCOV Service Execution Tests

The purpose of this test group is to verify the correct execution of the SubscribeCOV service request under circumstances where the service is expected to fail.

#### 9.10.2.1 The Monitored Object Does Not Support COV Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the monitored object does not support COV notifications.

Configuration Requirements: This test shall only be executed if IUT contains objects which will not accept a COV subscription. If every object in IUT will accept a COV subscription, then this test shall be skipped.

Test Steps:

```

1. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = (any object that does not support COV notifications),
 'Issue Confirmed Notifications' = TRUE,
 'Lifetime' = 60
2. IF (Protocol_Revision is present AND Protocol_Revision >= 10) THEN
 RECEIVE BACnet-Error PDU,
 'Error Class' = OBJECT,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
ELSE
 RECEIVE
 (BACnet-Error PDU,
 'Error Class' = OBJECT,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED) |
 (BACnet-Error PDU,
 Error Class = SERVICES,
 Error Code = SERVICE_REQUEST_DENIED | OTHER) |
 (BACnet-Error PDU,
 'Error Class' = PROPERTY,
 'Error Code' = NOT_COV_PROPERTY)

```

#### 9.10.2.2 The Monitored Object Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the monitored object does not exist.

Notes to Tester: If the IUT is able to support objects other than those that currently exist, and none of those objects that currently do not exist would support COV notification if they did, then the IUT may return an error code of OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED instead of UNKNOWN\_OBJECT.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object of a type that supports COV and an instance which does not exist in the IUT),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = 60
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN  
     RECEIVE BACnet-Error PDU,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT  
   ELSE  
     RECEIVE BACnet-Error PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = SERVICE\_REQUEST\_DENIED | OTHER  
   | (BACnet-Error PDU,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT)

#### 9.10.2.3 There Is No Space For A Subscription

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error. This test only applies to IUTs that claim a Protocol\_Revision of 10 or higher.

Test Conditionality: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test may be skipped.

Test Steps:

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until the IUT returns an Error-PDU) {

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = PID,  
     'Monitored Object Identifier' = (any object of that supports COV),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = 6000
  2. RECEIVE BACnet-SimpleACK-PDU |  
     (BACnet-Error-PDU,  
     'Error Class' = RESOURCES,  
     'Error Code' = NO\_SPACE\_TO\_ADD\_LIST\_ELEMENT)
  3. READ ACS = (Active\_COV\_Subscriptions)
  4. IF (a BACnet-SimpleACKBACnet-SimpleACK-PDU was received in step 2) THEN  
     CHECK (that the subscription is in ACS)  
   ELSE  
     CHECK (that the subscription is not in ACS)
- }

#### 9.10.2.4 The Lifetime Parameter is Out of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOV request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object in the IUT that supports COV),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (a value less than the maximum unsigned value supported by the IUT,  
                     but large enough to produce a Result(-) result by the IUT)
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 10) THEN  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE  
   ELSE  
     RECEIVE BACnet-Error-PDU,  
     'Error Class' = SERVICES,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE | SERVICE\_REQUEST\_DENIED | OTHER  
     | (RECEIVE BACnet-Reject-PDU,  
         Reject Reason = PARAMETER\_OUT\_OF\_RANGE)

### 9.10.3 Positive Unsubscribed COVNotification Execution Tests

#### 9.10.3.1 Unsubscribed COVNotification Execution Test

Purpose: To verify that the IUT executes UnconfirmedCOVNotification service requests, with 'Process Identifier' equal to 0.

Test Concept: Using any received and supported unsubscribed UnconfirmedCOVNotification, observe the effect of its execution.

Test Steps:

1. TRANSMIT UnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = 0,  
     'Initiating Device Identifier' = TD,  
     'Monitored Object Identifier' = (any object present in TD),  
     'Time Remaining' = 0,  
     'List of Values' = (any valid set of values)
2. CHECK (for any vendor-defined observable actions)

### 9.11 SubscribeCOVProperty Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing SubscribeCOVProperty service requests.

Configuration Requirements: The IUT shall be configured with at least one object that supports subscriptions for COV notifications.

#### 9.11.1 Positive SubscribeCOVProperty Service Execution Tests

The purpose of this test group is to verify the correct execution of the SubscribeCOVProperty service request under circumstances where the service is expected to be successfully completed.

##### 9.11.1.1 Confirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription for confirmed COV notifications.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any value > 0),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)

2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
 RECEIVE BACnetConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (any value > 0),  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
4. TRANSMIT BACnet-SimpleACK-PDU

#### 9.11.1.2 Unconfirmed COV Notifications

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription for Unconfirmed COV notifications.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
 'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE,  
 'Lifetime' = (any value > 0),  
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
 RECEIVE BACnetUnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (any value > 0),  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)

#### 9.11.1.3 Explicit Indefinite Lifetime COV Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription with an indefinite lifetime (lifetime = 0). Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
 'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = 0,  
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
 IF (the subscription was for confirmed notifications) THEN  
 RECEIVE BACnetConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = 0,  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)  
 TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
 RECEIVE BACnetUnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = 0,  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status\_Flags)

4. MAKE (a change to the monitored object that should cause a COV notification)

5. BEFORE **Notification Fail Time**

IF (the subscription was for confirmed notifications) THEN

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = 0,

'List of Values' = (values appropriate to the property subscribed to including the changed value that triggered the notification, and any other properties the IUT provides with it, such as Status\_Flags)

ELSE

RECEIVE BACnetUnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = 0,

'List of Values' = (values appropriate to the property subscribed to including the changed value that triggered the notification, and any other properties the IUT provides with it, such as Status\_Flags)

#### 9.11.1.4 Canceling COV Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to cancel a COV subscription. This test cancels the subscription made in 9.11.1.1.

Configuration Requirements: This test should be executed after test 9.11.1.1, while the subscription created in that test still exists in the IUT.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (the process identifier used in test 9.11.1.1),

'Monitored Object Identifier' = (the same object used in test 9.11.1.1),

'Monitored Property Identifier' = (the same property used in test 9.11.1.1)

2. RECEIVE BACnet-SimpleACK-PDU

3. WAIT **Notification Fail Time**

4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)

5. CHECK(the IUT does not transmit a COV notification)

#### 9.11.1.5 Canceling Expired or Non-Existing Subscriptions

Purpose: TO verify that the IUT correctly responds to a SubscribeCOVProperty request to cancel a subscription that no longer exists.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (any unused process identifier or an identifier from a previously terminated subscription),

'Monitored Object Identifier' = (any unused object or an object from a previously terminated subscription),

'Monitored Property Identifier' = (any unused property or a property from a previously terminated subscription)

2. RECEIVE BACnet-SimpleACK-PDU

3. WAIT **Notification Fail Time**

4. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)

5. CHECK(the IUT did not issue a COV notification)



**9.11.1.6 Implied Indefinite Lifetime COV Subscriptions**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription with an implied indefinite lifetime (lifetime parameter omitted). Either confirmed or unconfirmed notifications may be used, but at least one of these options must be supported by the IUT.

Test Steps: The test steps are identical to 9.11.1.1 except that the 'Lifetime' parameter in step 1 shall be omitted.

Notes to Tester: The passing result is identical to the results in 9.11.1.1.

**9.11.1.7 Finite Lifetime Subscriptions**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription with a temporary lifetime. Either confirmed or unconfirmed notifications may be used, but at least one of these options must be supported by the IUT.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (PID1, any valid process identifier),  
     'Monitored Object Identifier' = (O1, any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (T1, any value between 60 seconds and 300 seconds),  
     'Monitored Property Identifier' = (P1, any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (A value approximately equal to, but not greater than T1),  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     TRANSMIT BACnet-SimpleACK-PDU  
   ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetUnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (A value approximately equal to, but not greater than T1) ,  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
4. WAIT a period longer than the resolution of the IUT's COV subscription lifetime timer
5. MAKE (a change to the monitored object that causes a COV notification)
6. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (T2, a value greater than 0 and less than the requested subscription lifetime),  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     TRANSMIT BACnet-SimpleACK-PDU  
   ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetUnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1  
     'Initiating Device Identifier' = IUT,

- 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (T2, a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
7. WAIT a period longer than the resolution of the IUT's COV subscription lifetime timer
  8. MAKE (a change to the monitored object that causes a COV notification)
  9. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (a value greater than 0 and less than the T2),  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
  - ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetUnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (a value greater than 0 and less than the T2),  
     'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)
  10. WAIT (the lifetime of the subscription)
  11. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
  12. CHECK (verify that the IUT did not transmit a COV notification message)

#### 9.11.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to update the lifetime of a subscription. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Concept: A subscription for COV notifications is made for 60 seconds. Before that subscription has expired a second subscription is made for 300 seconds. When the notification is sent in response to the second subscription, the lifetime is checked to verify that it is greater than 60 but less than 300 seconds.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (PID1, any valid process identifier),  
     'Monitored Object Identifier' = (O1, any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = 60,  
     'Monitored Property Identifier' = (P1, any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (60 or less than 60),  
     'List of Values' = (values appropriate to the object type and subscribed to property)  
     TRANSMIT BACnet-SimpleACK-PDU
- ELSE  
     **BEFORE Notification Fail Time**  
     RECEIVE BACnetUnconfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = PID1,

'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (60 or less than 60),  
 'List of Values' = (values appropriate to the object type and subscribed to property)

4. TRANSMIT SubscribeCOVProperty-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Monitored Object Identifier' = O1,  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = (T1, a value between 180 and 300 seconds),  
 'Monitored Property Identifier' = P1

5. RECEIVE BACnet-SimpleACK-PDU

6. IF (the subscription was for confirmed notifications) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE BACnetConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (~T1, but not greater than T1),  
 'List of Values' = (values appropriate to the object type and subscribed to property)  
 TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
 BEFORE **Notification Fail Time**  
 RECEIVE BACnetUnconfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = PID1,  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = O1,  
 'Time Remaining' = (~T1, but not greater than T1),  
 'List of Values' = (values appropriate to the object type and subscribed to property)

#### 9.11.1.9 Client-Supplied COV Increment

Purpose: To verify that the IUT correctly generates COV notifications when the client supplies the COV increment in the SubscribeCOVProperty request. Either confirmed or unconfirmed notifications may be used but at least one of these options must be supported by the IUT.

Test Concept: A subscription for COV notification is made for a property of numeric datatype. The subscription request specifies a COV increment. The monitored property is changed by an amount less than the increment, and the TD waits to ensure that the IUT does not generate a notification. The monitored property is changed by an amount slightly more than is required to cause a COV notification, and the TD waits for the notification.

Configuration Requirements: If the property being subscribed to has a related COV\_Increment property in the object, then the value of the COV\_Increment property should be significantly different than the COV increment provided in the subscription service. In devices where the 'COV Increment' is always less than the minimal change that the monitored property can make, skip steps 4 and 5.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
 'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = (any value that will ensure no re-subscription is required to complete the test),  
 'Monitored Property Identifier' = (any valid property supporting COV notifications),  
 'COV Increment' = (any valid increment value)

2. RECEIVE BACnet-SimpleACK-PDU

3. BEFORE **Notification Fail Time**  
 IF (the subscription was for confirmed notifications) THEN  
 RECEIVE BACnetConfirmedCOVNotification-Request,  
 'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = (the requested lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object including  
 the value of monitored property)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

RECEIVE BACnetUnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = (the requested lifetime),

'List of Values' = (values appropriate to the object type of the monitored object  
 including the value of monitored property)

4. MAKE (the monitored property change by less than the COV increment)

5. CHECK (that the IUT did not transmit a notification message for the monitored property)

6. MAKE (the monitored property change by slightly more than COV Increment less the amount changed in step 5)

7. **BEFORE Notification Fail Time**

IF (the subscription was for confirmed notifications) THEN

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = ?,

'List of Values' = (values appropriate to the object type of the monitored object  
 including the changed value that triggered the notification)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

RECEIVE BACnetUnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' = ?,

'List of Values' = (values appropriate to the object type of the monitored object  
 including the changed value that triggered the notification)

#### 9.11.1.10 Accepts SubscribeCOVProperty-Requests with 8 Hour Lifetimes

Purpose: To verify that the IUT correctly accepts lifetimes of at least 8 hours.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (any valid process identifier),

'Monitored Object Identifier' = (any object supporting COV notifications),

'Issue Confirmed Notifications' = TRUE | FALSE,

'Lifetime' = 28800

'Monitored Property Identifier' = (any valid property supporting COV notifications)

2. RECEIVE BACnet-SimpleACK-PDU

3. IF (the subscription was for confirmed notifications) THEN

**BEFORE Notification Fail Time**

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,

'Monitored Object Identifier' = (the same object used in the subscription),

'Time Remaining' ~ (the requested lifetime),

'List of Values' = (values appropriate to the object type of the monitored object  
 including the value of monitored property)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

**BEFORE Notification Fail Time**

RECEIVE BACnetUnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),

'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' ~ = (the requested lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object  
 including the value of monitored property)

4. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (the same identifier used in Step 1),  
 'Monitored Object Identifier' = (the same identifier used in the subscription),  
 'Monitored Property Identifier' = (the same object used in the subscription)

5. RECEIVE BACnet-SimpleACK-PDU

#### 9.11.1.11 Confirmed Change of Value Notification from Property Value

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Property Value.

Test Concept: A property subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Value of the monitored Property is changed, and a notification shall be received. The subscribed property may be changed using the WriteProperty service or by another means. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (any valid process identifier),  
 'Monitored Object Identifier' = X  
 'Issue Confirmed Notifications' = TRUE,  
 'Lifetime' = L  
 'Monitored Property Identifier' = (Y, any valid property supporting COV notifications)

2. RECEIVE BACnet-SimpleACK-PDU

3. BEFORE **Notification Fail Time**

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = X  
 'Time Remaining' = (any value appropriate for the Lifetime selected),  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties  
 the IUT provides with it, such as Status-Flags)

4. TRANSMIT BACnet-SimpleACK-PDU

5. MAKE (a change to the monitored object PROPERTY that causes a COV notification)

6. BEFORE **Notification Fail Time**

RECEIVE BACnetConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = X  
 'Time Remaining' = (any value appropriate for the Lifetime selected),  
 'List of Values' = (values appropriate to the property subscribed to, and any other properties  
 the IUT provides with it, such as Status-Flags)

7. TRANSMIT BACnet-SimpleACK-PDU

8. TRANSMIT SubscribeCOVProperty-Request,

'Subscriber Process Identifier' = (the same identifier used in Step 1),  
 'Monitored Object Identifier' = X  
 'Monitored Property Identifier' = Y

9. RECEIVE BACnet-SimpleACK-PDU

#### 9.11.1.12 Unconfirmed Change of Value Notification from Property Value

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Property Value.

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.11, except that the SubscribeCOVProperty service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

#### 9.11.1.13 Confirmed Change of Value Notification from Status\_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status\_Flags Property.

Test Concept: A property subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then made to change and a notification shall be received. For implementations where it is not possible to change the Status\_Flags by any other means, this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = X  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L  
     'Monitored Property Identifier' = Y (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (values appropriate to the property subscribed to and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. BEFORE **Notification Fail Time**  
     RECEIVE BACnetConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same identifier used in the subscription),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (initial values appropriate to the property subscribed to and new Status\_Flags)
7. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (the same identifier used in Step 1),  
     'Monitored Object Identifier' = X  
     'Monitored Property Identifier' = Y
8. RECEIVE BACnet-SimpleACK-PDU

#### 9.11.1.14 Unconfirmed Change of Value Notification from Status\_Flags Property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags Property.

Test Steps: The steps for this test case are identical to the test steps in 9.11.1.13 except that the SubscribeCOVProperty service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients

#### 9.11.2 Negative SubscribeCOVProperty Service Execution Tests

The purpose of this test group is to verify the correct execution of the SubscribeCOVProperty service request under circumstances where the service is expected to fail.



### 9.11.2.1 The Monitored Object Does Not Support COV Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object does not support COV notifications.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object that does not support COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = 60,  
     'Monitored Property Identifier' = (any property in the object)
2. IF (Protocol Revision < 15) THEN  
     RECEIVE  
         (BACnet-Error PDU,  
             Error Class = SERVICES,  
             Error Code = SERVICE\_REQUEST\_DENIED | OTHER ) |  
         (BACnet-Error PDU,  
             Error Class = OBJECT  
             Error Code = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED ) |  
         (BACnet-Error-PDU,  
             'Error Class' = PROPERTY,  
             'Error Code' = NOT\_COV\_PROPERTY)
- ELSE  
     RECEIVE  
         BACnet-Error PDU,  
             Error Class = OBJECT  
             Error Code = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED |  
         (BACnet-Error-PDU,  
             'Error Class' = PROPERTY,  
             'Error Code' = NOT\_COV\_PROPERTY)

### 9.11.2.2 The Monitored Property Does Not Support COV Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object supports COV notifications but not on the requested property.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object that supports COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = 60,  
     'Monitored Property Identifier' = (any property of the chosen object that does not support COV notifications)
2. IF (Protocol Revision < 15) THEN  
     RECEIVE  
         (BACnet-Error PDU,  
             Error Class = SERVICES,  
             Error Code = SERVICE\_REQUEST\_DENIED | OTHER ) |  
         (BACnet-Error PDU,  
             Error Class = PROPERTY  
             Error Code = NOT\_COV\_PROPERTY )
- ELSE  
     RECEIVE  
         BACnet-Error PDU,  
             Error Class = PROPERTY  
             Error Code = NOT\_COV\_PROPERTY

**9.11.2.3 Monitored Object Does Not Exist**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored object does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object of a type that supports COV and an instance which does not exist in the IUT),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = 60
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = OBJECT,
  - 'Error Code' = UNKNOWN\_OBJECT

**9.11.2.4 Monitored Property Does Not Exist**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the monitored property does not exist.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = 60
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications which does not exist for specified object)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = UNKNOWN\_PROPERTY

**9.11.2.5 There Is No Space For Subscription**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error. This test only applies to IUTs that claim a Protocol\_Revision of 10 or higher.

Test Conditionality: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test may be skipped.

Test Steps:

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1, or until the IUT returns an Error-PDU) DO {

1. TRANSMIT SubscribeCOVProperty-Request,
    - 'Subscriber Process Identifier' = PID,
    - 'Monitored Object Identifier' = (object supporting COV notifications),
    - 'Issue Confirmed Notifications' = TRUE,
    - 'Lifetime' = 6000
    - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
  2. RECEIVE
    - BACnet-SimpleACK-PDU |
    - (BACnet-Error-PDU,
      - 'Error Class' = RESOURCES,
      - 'Error Code' = NO\_SPACE\_TO\_ADD\_LIST\_ELEMENT)
- }

**9.11.2.6 The Lifetime Parameter is Out of Range**

Purpose: To verify that the IUT correctly responds to a SubscribeCOVProperty request to establish a subscription when the Lifetime parameter is out of range.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (a value larger than that supported by the IUT),
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. IF (Protocol\_Revision is present and Protocol\_Revision => 15) THEN
  - RECEIVE BACnet-Error-PDU,
    - 'Error Class' = SERVICES,
    - 'Error Code' = VALUE\_OUT\_OF\_RANGE
  - ELSE
    - RECEIVE BACnet-Error-PDU,
      - 'Error Class' = SERVICES,
      - 'Error Code' = (VALUE\_OUT\_OF\_RANGE | SERVICE\_REQUEST\_DENIED | OTHER
        - | (RECEIVE BACnet-Reject-PDU),
        - | Reject Reason = PARAMETER\_OUT\_OF\_RANGE)

**9.12 AtomicReadFile Service Execution Tests**

This clause defines the tests necessary to demonstrate support for executing AtomicReadFile service requests.

Test Concept: The BACnet file access services permit access to files on either a record basis or a stream basis. If the IUT supports record-based file structures then the record access tests (9.12.1.1 and 9.12.2.1) shall be used. If the IUT supports stream-based file structures then the stream access tests (9.12.1.2 and 9.12.2.2) shall be used. If only one file access type is supported in the IUT this shall be explicitly documented in the PICS and only the tests in this clause that apply to that file type need to be executed. The tests consist of reading the contents of the file using the AtomicReadFile service in various ways and verifying that the appropriate known file data is returned.

Configuration Requirements: The AtomicReadFile service execution tests require that the TD have knowledge of the exact contents of a known file. The IUT shall be configured with a file that supports record access and one that supports stream access. In the test procedures "R" will designate the File object identifier for the record access test file and "S" will designate the File object identifier for the stream access test file. If the IUT does not support both record access and stream access then one of these files may be omitted. The minimum test file size is four octets for stream access and four records for record access files. These files can be configured into the IUT or the AtomicWriteFile service can be used to initialize the files to a known state. The test procedures assume that the IUT is already configured with the known file data provided by the manufacturer.

**9.12.1 Positive AtomicReadFile Service Execution Tests****9.12.1.1 Reading Record Based Files**

The purpose of this test group is to verify the correct execution of the AtomicReadFile service requests from record-based files under circumstances where the service is expected to be successfully completed.

**9.12.1.1.1 Reading an Entire File**

Purpose: To verify that the IUT correctly responds to a request to read an entire file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = 0,
  - 'Requested Record Count' = (the number of records in the test file)
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Record' = 0,

'Returned Record Count' = (the number of records in the test file),  
 'File Record Data' = (the known contents of the test file)

#### 9.12.1.1.2 Reading Data from the Beginning of a File

Purpose: To verify that the IUT correctly responds to a request to read data from the beginning of the file to an intermediate point before the end of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = 0,  
     'Requested Record Count' = (any number  $n$ :  $0 < n < \text{the number of records in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,  
     'End of File' = FALSE,  
     'File Start Record' = 0,  
     'Returned Record Count' =  $n$ ,  
     'File Record Data' = (the first  $n$  records of the test file)

#### 9.12.1.1.3 Reading Data from an Intermediate Point to the End of the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from an intermediate point and continuing to the end of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = (any number  $n$ :  $0 < n < \text{the number of records in the test file}$ ),  
     'Requested Record Count' = (the number of records in the test file –  $n$ )
2. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Record' =  $n$ ,  
     'Returned Record Count' = (the number of records in the test file –  $n$ ),  
     'File Record Data' = (the test file record data from position  $n$  to the end of the file)

#### 9.12.1.1.4 Reading Data Beginning from an Intermediate Point and Ending at Another Intermediate Point in the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from an intermediate point and continuing to another intermediate point in the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = (any number  $n$ :  $0 < n < (\text{the number of records in the test file} - 2)$ ),  
     'Requested Record Count' = (any number  $m$ :  $0 < m < \text{the number of records remaining in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,  
     'End of File' = FALSE,  
     'File Start Record' =  $n$ ,  
     'Returned Record Count' =  $m$ ,  
     'File Record Data' = (the specified test file record data)

#### 9.12.1.1.5 Reading A Data Block of Size Zero

Purpose: To verify that the IUT correctly responds to a request to read zero records of file data.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,

- 'File Start Record' = (any number n:  $0 \leq n < \text{the number of records in the test file}$ ),  
 'Requested Record Count' = 0
2. RECEIVE AtomicReadFile-ACK,  
 'End of File' = FALSE,  
 'File Start Record' = n,  
 'Returned Record Count' = 0,  
 'File Record Data' = (an empty list of records)

#### 9.12.1.1.6 Reading Data Past the End of the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from any point and continuing past the end of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
 'File Identifier' = R,  
 'File Start Record' = (any number n:  $0 \leq n < \text{the number of records in the test file}$ ),  
 'Requested Record Count' = (any number m:  $m > \text{the number of records remaining in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Record' = n,  
 'Returned Record Count' = (the number of records in the test file  $- n$ ),  
 'File Record Data' = (the test file records from position n to the end of the file)

#### 9.12.1.2 Reading Stream Based Files

The purpose of this test group is to verify the correct execution of the AtomicReadFile service requests from stream-based files under circumstances where the service is expected to be successfully completed.

##### 9.12.1.2.1 Reading an Entire Stream Based File

Purpose: To verify that the IUT correctly responds to a request to read an entire file.

Test Concept: The test consists of reading the contents of the file using a sequence of AtomicReadFile requests and verifying that the appropriate known file data is returned.

Configuration Requirements: The AtomicReadFile service execution tests require that the TD has knowledge of the exact contents of a known file F1. The test procedures assume that the IUT is already configured with the known file data provided by the manufacturer. In the test procedures "X" will designate the File object identifier and Z the 'File Start Position' initialized at "0". When performing the AtomicReadFile services, a Maximum Requested Octet Count (MROC) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max\_APDU\_Length\_Accepted and the IUT's maximum transmittable APDU length.

Test Steps:

1. VERIFY File\_Access\_Method = STREAM\_ACCESS
2. WHILE (the last read resulted in an Ack with 'End Of File' = FALSE) DO {  
 TRANSMIT AtomicReadFile-Request,  
 'Object Identifier' = X,  
 'File Start Position' = Z (the next unread octet),  
 'Requested Octet Count' = MROC  
 RECEIVE AtomicReadFile-ACK,  
 'End Of File' = TRUE | FALSE,  
 'File Start Position' = Z  
 'File Data' = (the known contents of the test file of length MROC if 'End Of File' is FALSE or of length MROC or less if 'End Of File' is TRUE)  
 }
3. CHECK(that the returned file data is F1)

##### 9.12.1.2.2 Reading Data from the Beginning of a File

Purpose: To verify that the IUT correctly responds to a request to read data from the beginning of the file to an intermediate point before the end of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = 0,
  - 'Requested Octet Count' = (any number  $n$ :  $0 < n < \text{the number of octets in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = FALSE,
  - 'File Start Position' = 0,
  - 'File Data' = (the first  $n$  octets of the test file)

#### 9.12.1.2.3 Reading Data from an Intermediate Point to the End of the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from an intermediate point and continuing to the end of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = (any number  $n$ :  $0 < n < \text{the number of octets in the test file}$ ),
  - 'Requested Octet Count' = (the number of octets in the test file  $- n$ )
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Position' =  $n$ ,
  - 'File Data' = (the test file data from position  $n$  to the end of the file)

#### 9.12.1.2.4 Reading Data Beginning from an Intermediate Point and Ending at Another Intermediate Point in the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from an intermediate point and continuing to another intermediate point in the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = (any number  $n$ :  $0 < n < (\text{the number of octets in the test file} - 2)$ ),
  - 'Requested Octet Count' = (any number  $m$ :  $0 < m < \text{the number of octets remaining in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = FALSE,
  - 'File Start Position' =  $n$ ,
  - 'File Data' = (the specified test file data)

#### 9.12.1.2.5 Reading A Data Block of Size Zero

Purpose: To verify that the IUT correctly responds to a request to read zero octets of file data.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = (any number  $n$ :  $0 \leq n < \text{the number of octets in the test file}$ ),
  - 'Requested Octet Count' = 0
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = FALSE,
  - 'File Start Position' =  $n$ ,
  - 'File Data' = (an octet string of length 0)

#### 9.12.1.2.6 Reading Data Past the End of the File

Purpose: To verify that the IUT correctly responds to a request to read data beginning from any point and continuing past the end of the file.



Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = (any number n:  $0 \leq n < \text{the number of octets in the test file}$ ),
  - 'Requested Octet Count' = (any number m:  $m > \text{the number of octets remaining in the test file}$ )
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Position' = n,
  - 'File Data' = (the test file octets from position n to the end of the file)

### 9.12.2 Negative AtomicReadFile Service Execution Tests

#### 9.12.2.1 Reading Record Based Files

The purpose of this test group is to verify the correct execution of the AtomicReadFile service requests from record-based files under circumstances where the service is expected to fail.

##### 9.12.2.1.1 Attempting to Read Data from a Range of Records Outside the File Boundaries

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified records are outside of the boundaries of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = (any number n:  $n \geq \text{the number of records in the test file}$ ),
  - 'Requested Record Count' = (any number  $> 0$ )
2. RECEIVE BACnet-Error PDU,
  - Error Class = SERVICES,
  - Error Code = INVALID\_FILE\_START\_POSITION

##### 9.12.2.1.2 Attempting to Read Data from a Nonexistent File

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified file does not exist.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = (any non existent file),
  - 'File Start Record' = (any number  $\geq 0$ ),
  - 'Requested Record Count' = (any number  $> 0$ )
2. RECEIVE BACnet-Error PDU,
  - Error Class = OBJECT,
  - Error Code = UNKNOWN\_OBJECT

##### 9.12.2.1.3 Attempting to Read Data Using the Wrong File Access Type

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the file access type is inappropriate for the specified file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Position' = 0,
  - 'Requested Octet Count' = 1
2. RECEIVE BACnet-Error PDU,
  - Error Class = SERVICES,
  - Error Code = INVALID\_FILE\_ACCESS\_METHOD

**9.12.2.1.4 Attempting to Read Data Beginning with a Record Number Less Than Zero**

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified record range is invalid.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
 

|                            |                        |
|----------------------------|------------------------|
| 'File Identifier' =        | R,                     |
| 'File Start Record' =      | (any number n: n < 0), |
| 'Requested Record Count' = | 1                      |
2. RECEIVE
 

|                     |                         |
|---------------------|-------------------------|
| (BACnet-Error PDU,  |                         |
| Error Class =       | SERVICES,               |
| Error Code =        | INVALID_START_POSITION) |
| (BACnet-Reject-PDU, |                         |
| Reject Reason =     | PARAMETER_OUT_OF_RANGE) |

**9.12.2.2 Reading Stream Based Files**

The purpose of this test group is to verify the correct execution of the AtomicReadFile service requests from stream-based files under circumstances where the service is expected to fail.

**9.12.2.2.1 Attempting to Read Data from a Range of Records Outside the File Boundaries**

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified octets are outside of the boundaries of the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
 

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| 'File Identifier' =       | S,                                                          |
| 'File Start Position' =   | (any number n: n >= the number of octets in the test file), |
| 'Requested Octet Count' = | (any number > 0)                                            |
2. RECEIVE BACnet-Error PDU,
 

|               |                             |
|---------------|-----------------------------|
| Error Class = | SERVICES,                   |
| Error Code =  | INVALID_FILE_START_POSITION |

**9.12.2.2.2 Attempting to Read Data from a Nonexistent File**

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified file does not exist.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
 

|                           |                          |
|---------------------------|--------------------------|
| 'File Identifier' =       | (any non existent file), |
| 'File Start Position' =   | (any number >= 0),       |
| 'Requested Octet Count' = | (any number > 0)         |
2. RECEIVE BACnet-Error PDU,
 

|               |                |
|---------------|----------------|
| Error Class = | OBJECT,        |
| Error Code =  | UNKNOWN_OBJECT |

**9.12.2.2.3 Attempting to Read Data Using the Wrong File Access Type**

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the file access type is inappropriate for the specified file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
 

|                            |    |
|----------------------------|----|
| 'File Identifier' =        | S, |
| 'File Start Record' =      | 0, |
| 'Requested Record Count' = | 1  |

2. RECEIVE BACnet-Error PDU,  
     Error Class = SERVICES,  
     Error Code = INVALID\_FILE\_ACCESS\_METHOD

#### 9.12.2.2.4 Attempting to Read Data Beginning with a Start Position Less Than Zero

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified record range is invalid.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = S,  
     'File Start Position' = (any number n: n < 0),  
     'Requested Octet Count' = 1
2. RECEIVE  
     (BACnet-Error PDU,  
         Error Class = SERVICES,  
         Error Code = INVALID\_START\_POSITION) |  
     (BACnet-Reject-PDU,  
         Reject Reason = PARAMETER\_OUT\_OF\_RANGE)

### 9.13 AtomicWriteFile Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing AtomicWriteFile service requests.

Test Concept: The BACnet file access services permit access to files on either a record basis or a stream basis. If the IUT supports write access to record-based files then the record access tests (9.13.1.1 and 9.13.2.1) shall be used. If the IUT supports stream-based file structures then the stream access tests (9.13.1.2 and 9.13.2.2) shall be used. If only one file access type is supported in the IUT this shall be explicitly documented in the PICS and only the tests in this clause that apply to that file type need to be executed. The tests consist of modifying the contents of the files using the AtomicWriteFile service in various ways and verifying that the appropriate changes to the file data took place.

Some implementations may have special restrictions on files. For example, files that represent the device's operational software may contain proprietary header information that is used to ensure the authenticity of the file. Any such special restrictions must be documented in the PICS.

Configuration Requirements: If write access to record-based files is supported the IUT shall be configured with a record-based file object that permits write access. The object identifier for this file will be designated "R" in the test descriptions. If write access to stream-based files is supported the IUT shall be configured with a stream-based file object that permits write access. The object identifier for this file will be designated "S" in the test descriptions. The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file objects shall be configured with initial data that differs from the test data.

#### 9.13.1 Positive AtomicWriteFile Service Execution Tests

##### 9.13.1.1 Writing to Record-Based Files

The purpose of this test group is to verify the correct execution of AtomicWriteFile service requests for record-based files under circumstances where the service is expected to be successfully completed.

##### 9.13.1.1.1 Writing an Entire File

Purpose: To verify that the IUT correctly responds to a request to write an entire file.

Configuration Requirements: The test data shall contain at least as many records as the initial data for the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = 0,  
     'Requested Record Count' = (any number >= the number of records in the test data)

2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Record' = 0,
  - 'Returned Record Count' = (any number  $\leq$  the number of records in the test data),
  - 'File Record Data' = (the initial data)
3. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = 0,
  - 'Record Count' = (the number of records in the test data),
  - 'File Record Data' = (the test data)
4. RECEIVE AtomicWriteFile-ACK,
  - 'File Start Record' = 0
5. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = 0,
  - 'Requested Record Count' = (any number  $>$  the number of records in the test data)
6. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Record' = 0,
  - 'Returned Record Count' = (the number of records in the test data),
  - 'File Record Data' = (the test data)
7. VERIFY (R), Modification\_Date = (the current date and time)
8. VERIFY (R), ARCHIVE = FALSE
9. VERIFY (R), Number\_Of\_Records = (the number of records in the test data)

#### 9.13.1.1.2 Overwriting a Portion of a File

Purpose: To verify that the IUT correctly responds to a request to write to a file beginning at any intermediate point. If the IUT does not support files that can not be modified except by replacing the entire file, and this restriction is clearly stated in the PICS, then this test may be ignored.

Configuration Requirements: The file object shall be configured with data that differs from the test data.

Test Steps:

1. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = R,
  - 'Property Identifier' = Number\_Of\_Records
2. RECEIVE ReadProperty-ACK,
  - 'Object Identifier' = R,
  - 'Property Identifier' = Number\_Of\_Records
  - 'Property Value' = (the current number of records, designated "InitialNumRecords" below)
3. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = 0,
  - 'Requested Record Count' = (any number  $>$  InitialNumRecords)
4. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Record' = 0,
  - 'Returned Record Count' = InitialNumRecords,
  - 'File Record Data' = (the initial data)
5. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = (any value  $n$ :  $0 < n < \text{InitialNumRecords}$ ),
  - 'Record Count' = (the number of records in the test data),
  - 'File Record Data' = (the test data)
6. RECEIVE AtomicWriteFile-ACK,
  - 'File Start Record' = (the 'File Start Record' used in step 4)
7. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = R,
  - 'File Start Record' = (the 'File Start Record' used in step 4),

- 'Requested Record Count' = (the number of records in the test data)
8. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Record' = (the 'File Start Record' used in step 4),  
 'Returned Record Count' = (the number of records in the test data),  
 'File Record Data' = (the test data)
9. VERIFY (R), Modification\_Date = (the current date and time)
10. VERIFY (R), ARCHIVE = FALSE
11. VERIFY (R), Number\_Of\_Records = (the number of records in the test data + the 'File Start Record' used in step 4)

#### 9.13.1.1.3 Appending Data to the End of a File

Purpose: To verify that the IUT correctly responds to a request to write to the end of a file. If the IUT does not support files that cannot be modified except by replacing the entire file, and this restriction is clearly stated in the PICS, then this test may be ignored.

Configuration Requirements: The file object shall be configured with data that differs from the test data.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = R,  
 'Property Identifier' = Number\_Of\_Records
2. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = R,  
 'Property Identifier' = Number\_Of\_Records  
 'Property Value' = (the current number of records, designated "InitialNumRecords" below)
3. TRANSMIT AtomicReadFile-Request,  
 'File Identifier' = R,  
 'File Start Record' = 0,  
 'Requested Record Count' = (any number > InitialNumRecords)
4. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Record' = 0,  
 'Returned Record Count' = InitialNumRecords,  
 'File Record Data' = (the initial data)
5. TRANSMIT AtomicWriteFile-Request,  
 'File Identifier' = R,  
 'File Start Record' = -1,  
 'Record Count' = (the number of records in the test data),  
 'File Record Data' = (the test data)
6. RECEIVE AtomicWriteFile-ACK,  
 'File Start Record' = InitialNumRecords,
7. TRANSMIT AtomicReadFile-Request,  
 'File Identifier' = R,  
 'File Start Record' = InitialNumRecords,  
 'Requested Record Count' = (the number of records in the test data)
8. RECEIVE AtomicReadFile-ACK,  
 'End of File' = TRUE,  
 'File Start Record' = InitialNumRecords,  
 'Returned Record Count' = (the number of records in the test data),  
 'File Record Data' = (the test data)
9. VERIFY (R), Modification\_Date = (the current date and time)
10. VERIFY (R), ARCHIVE = FALSE
11. VERIFY (R), Number\_Of\_Records = (the number of records in the test data + InitialNumRecords)

#### 9.13.1.1.4 Truncating a File

Purpose: To verify that the IUT correctly responds to a WriteProperty request to truncate a file. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 1.

Configuration Requirements: The manufacturer shall configure the IUT with a file object that permits write access and contains initial file data more than one record in length. A copy of the file data shall also be provided to the tester.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = R,  
     'Property Identifier' = Number\_Of\_Records
2. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = R,  
     'Property Identifier' = Number\_Of\_Records  
     'Property Value' = (the current number of records, designated "InitialNumRecords" below)
3. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = 0,  
     'Requested Record Count' = (any number > InitialNumRecords)
4. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Record' = 0,  
     'Returned Record Count' = InitialNumRecords,  
     'File Record Data' = (the initial data)
5. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = R,  
     'Property Identifier' = Number\_Of\_Records,  
     'Property Value' = (any value n: 0 < n < InitialNumRecords)
6. RECEIVE BACnet-SimpleACK-PDU
7. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = 0,  
     'Requested Record Count' = InitialNumRecords
8. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Record' = 0,  
     'Returned Record Count' = n,  
     'File Record Data' = (the initial data from records 0 – (n-1))

#### 9.13.1.1.5 Deleting a File

Purpose: To verify that the IUT correctly responds to a WriteProperty request to delete a file. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 1.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = R,  
     'Property Identifier' = Number\_Of\_Records
2. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = R,  
     'Property Identifier' = Number\_Of\_Records  
     'Property Value' = (the current number of records, designated "InitialNumRecords" below)
3. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = R,  
     'File Start Record' = 0,  
     'Requested Record Count' = (any number > InitialNumRecords)
4. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Record' = 0,  
     'Returned Record Count' = InitialNumRecords,  
     'File Record Data' = (the initial data for this file)



5. TRANSMIT WriteProperty-Request,
 

|                         |                    |
|-------------------------|--------------------|
| 'Object Identifier' =   | R,                 |
| 'Property Identifier' = | Number_Of_Records, |
| 'Property Value' =      | 0                  |
6. RECEIVE BACnet-SimpleACK-PDU
7. TRANSMIT AtomicReadFile-Request,
 

|                            |                   |
|----------------------------|-------------------|
| 'File Identifier' =        | R,                |
| 'File Start Record' =      | 0,                |
| 'Requested Record Count' = | InitialNumRecords |
8. RECEIVE AtomicReadFile-ACK,
 

|                           |                            |
|---------------------------|----------------------------|
| 'End of File' =           | TRUE,                      |
| 'File Start Record' =     | 0,                         |
| 'Returned Record Count' = | 0,                         |
| 'File Record Data' =      | (an empty list of records) |

#### 9.13.1.2 Writing to Stream-Based Files

The purpose of this test group is to verify the correct execution of AtomicWriteFile service requests for stream-based files under circumstances where the service is expected to be successfully completed.

##### 9.13.1.2.1 Writing an Entire Stream Based File

Purpose: To verify that the IUT correctly responds to a request to write an entire file.

Test Concept: The tests consist of modifying the contents of the files using the AtomicWriteFile service and verifying that the appropriate changes to the file data took place

Configuration Requirements: The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file objects shall be configured with initial data that differs from the test data. In the test procedures, "X" will designate the File object identifier and Z the File Start Position' initialized at "1" at the beginning. When performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max\_APDU\_Length\_Accepted.

Test Steps:

1. VERIFY Read\_Only = FALSE
2. WRITE Archive = TRUE
3. VERIFY File\_Access\_Method = STREAM ACCESS
4. IF (File\_Size is not equal to the size of the test file) THEN
 

|                   |   |
|-------------------|---|
| WRITE File_Size = | 0 |
|-------------------|---|
5. REPEAT Z = (0 through the file size, in increments of MWDL) DO {
 

|                                  |                                                                                    |
|----------------------------------|------------------------------------------------------------------------------------|
| TRANSMIT AtomicWriteFile-Request |                                                                                    |
| 'File Identifier' =              | X                                                                                  |
| 'File Start Position' =          | Z                                                                                  |
| 'File Data' =                    | (file contents, the number of octets being the lesser of (file size - Z) and MWDL) |
| RECEIVE AtomicWriteFile-ACK      |                                                                                    |
| 'File Start Position' =          | Z                                                                                  |
6. VERIFY File\_Size = (file size of the test data)
7. VERIFY Modification\_Date = (the current date and time)
8. VERIFY ARCHIVE = FALSE

##### 9.13.1.2.2 Overwriting a Portion of a File

Purpose: To verify that the IUT correctly responds to a request to write to a file beginning at an intermediate point. If the IUT does not support files that can not be modified except by replacing the entire file, and this restriction is clearly stated in the PICS, then this test may be ignored.

Configuration Requirements: The file object shall be configured with data that differs from the test data.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,  
    'Property Value' = (the current file size, designated "InitialNumOctets" below)
3. TRANSMIT AtomicReadFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = 0,  
    'Requested Octet Count' = (any number > InitialNumOctets)
4. RECEIVE AtomicReadFile-ACK,  
    'End of File' = TRUE,  
    'File Start Position' = 0,  
    'File Data' = (the initial data)
5. TRANSMIT AtomicWriteFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = (any value n:  $0 < n < \text{InitialNumOctets}$ ),  
    'File Record Data' = (the test data)
6. RECEIVE AtomicWriteFile-ACK,  
    'File Start Position' = (the 'File Start Position' used in step 4)
7. TRANSMIT AtomicReadFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = (the 'File Start Position' used in step 4),  
    'Requested Octet Count' = (the number of octets in the test data)
8. RECEIVE AtomicReadFile-ACK,  
    'End of File' = TRUE,  
    'File Start Position' = (the 'File Start Position' used in step 4),  
    'File Data' = (the test data)
9. VERIFY (R), Modification\_Date = (the current date and time)
10. VERIFY (R), ARCHIVE = FALSE
11. VERIFY (R), File\_Size = (the number of octets in the test data + the 'File Start Position' used in step 4)

#### 9.13.1.2.3 Appending Data to the End of a File

Purpose: To verify that the IUT correctly responds to a request to write to the end of a file.

Configuration Requirements: The manufacturer shall provide appropriate test data to write to these files or sufficient information to permit the tester to construct the test data. The file object shall be configured with initial data that differs from the test data. In the test procedures, "X" will designate the File object identifier and when performing the AtomicWriteFile services, a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max\_APDU\_Length\_Accepted.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,  
    'Property Value' = (the current size in octets, designated "InitialNumOctets" below)
3. TRANSMIT AtomicWriteFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = -1,  
    'File Data' = (the test data, the number of octets being the lesser of (file size - Z) and MWDL)

4. RECEIVE AtomicWriteFile-ACK,  
    'File Start Position' = InitialNumOctets,
5. VERIFY (R), Modification\_Date = (the current date and time)
6. VERIFY (R), ARCHIVE = FALSE
7. VERIFY (R), File\_Size = (the number of octets in the test data + InitialNumOctets)

#### 9.13.1.2.4 Truncating a File

Purpose: To verify that the IUT correctly responds to a WriteProperty request to truncate a file. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 1.

Configuration Requirements: The manufacturer shall configure the IUT with a file object that permits write access and contains initial file data more than one octet in length. A copy of the file data shall also be provided to the tester.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,  
    'Property Value' = (the current size in octets, designated "InitialNumOctets" below)
3. TRANSMIT AtomicReadFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = 0,  
    'Requested Octet Count' = (any number > InitialNumOctets)
4. RECEIVE AtomicReadFile-ACK,  
    'End of File' = TRUE,  
    'File Start Position' = 0,  
    'File Data' = (the initial data)
5. TRANSMIT WriteProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,  
    'Property Value' = (any value n: 0 < n < InitialNumOctets)
6. RECEIVE BACnet-SimpleACK-PDU
7. TRANSMIT AtomicReadFile-Request,  
    'File Identifier' = S,  
    'File Start Position' = 0,  
    'Requested Octet Count' = InitialNumOctets
8. RECEIVE AtomicReadFile-ACK,  
    'End of File' = TRUE,  
    'File Start Position' = 0,  
    'File Data' = (the test data for this file from octets 0 – (n-1))

#### 9.13.1.2.5 Deleting a File

Purpose: To verify that the IUT correctly responds to a WriteProperty request to delete a file. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 1.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = S,  
    'Property Identifier' = File\_Size,  
    'Property Value' = (the current size in octets, designated "InitialNumOctets" below)
3. TRANSMIT AtomicReadFile-Request,  
    'File Identifier' = S,

- |    |                                  |                                 |
|----|----------------------------------|---------------------------------|
|    | 'File Start Position' =          | 0,                              |
|    | 'Requested Octet Count' =        | (any number > InitialNumOctets) |
| 4. | RECEIVE AtomicReadFile-ACK,      |                                 |
|    | 'End of File' =                  | TRUE,                           |
|    | 'File Start Position' =          | 0,                              |
|    | 'File Data' =                    | (the initial data)              |
| 5. | TRANSMIT WriteProperty-Request,  |                                 |
|    | 'Object Identifier' =            | S,                              |
|    | 'Property Identifier' =          | File_Size,                      |
|    | 'Property Value' =               | 0                               |
| 6. | RECEIVE BACnet-SimpleACK-PDU     |                                 |
| 7. | TRANSMIT AtomicReadFile-Request, |                                 |
|    | 'File Identifier' =              | S,                              |
|    | 'File Start Position' =          | 0,                              |
|    | 'Requested Octet Count' =        | InitialNumOctets                |
| 8. | RECEIVE AtomicReadFile-ACK,      |                                 |
|    | 'End of File' =                  | TRUE,                           |
|    | 'File Start Position' =          | 0,                              |
|    | 'File Record Data' =             | (an octet string with length 0) |

### 9.13.2 Negative AtomicWriteFile Service Execution Tests

The purpose of this test group is to verify the correct execution of AtomicWriteFile service requests under circumstances where the service is expected to fail.

#### 9.13.2.1 Writing to Record Access Files

##### 9.13.2.1.1 Writing to a Record Access File using Stream Access

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a file using the wrong access method. This test case should only be executed if the IUT supports file objects that use record access.

Test Steps:

- TRANSMIT AtomicWriteFile-Request,
 

|                         |                        |
|-------------------------|------------------------|
| 'File Identifier' =     | R,                     |
| 'File Start Position' = | 0,                     |
| 'File Data' =           | (any stream file data) |
- RECEIVE BACnet-Error-PDU,
 

|               |                            |
|---------------|----------------------------|
| Error Class = | SERVICES,                  |
| Error Code =  | INVALID_FILE_ACCESS_METHOD |

##### 9.13.2.1.2 Writing to a File with an Invalid Starting Position

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a file using an invalid start position.

Test Steps:

- TRANSMIT AtomicWriteFile-Request,
 

|                       |                                                                          |
|-----------------------|--------------------------------------------------------------------------|
| 'File Identifier' =   | R,                                                                       |
| 'File Start Record' = | (any value n: $n < -1$   $n >$ the maximum supported number of records), |
| 'Record Count' =      | (any value > 0),                                                         |
| 'File Record Data' =  | (any record data)                                                        |
- RECEIVE BACnet-Error-PDU,
 

|               |                             |
|---------------|-----------------------------|
| Error Class = | SERVICES,                   |
| Error Code =  | INVALID_FILE_START_POSITION |

##### 9.13.2.1.3 Writing to a Read Only File

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a read only file.

Configuration Requirements: The IUT shall be configured with a file that does not permit write access. If it is not possible to configure such a file this test may be omitted.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = (any record access file that is read only),
  - 'File Start Record' = 0,
  - 'Record Count' = (any value > 0),
  - 'File Record Data' = (any record data)
2. RECEIVE
  - (BACnet-Error-PDU,
    - Error Class = PROPERTY,
    - Error Code = WRITE\_ACCESS\_DENIED) |
  - (BACnet-Error-PDU,
    - Error Class = SERVICES,
    - Error Code = FILE\_ACCESS\_DENIED)

#### 9.13.2.1.4 Writing to a Nonexistent File

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a nonexistent file.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = (any nonexistent file),
  - 'File Start Record' = 0,
  - 'Record Count' = (any value > 0),
  - 'File Record Data' = (any record data)
2. RECEIVE BACnet-Error-PDU,
  - Error Class = OBJECT,
  - Error Code = UNKNOWN\_OBJECT | NO\_OBJECTS\_OF\_SPECIFIED\_TYPE

#### 9.13.2.2 Writing to Stream Access Files

##### 9.13.2.2.1 Writing to a Stream Access File using Record Access

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a file using the wrong access method. This test case should only be executed if the IUT supports file objects that use stream access.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = S,
  - 'File Start Record' = 0,
  - 'Record Count' = (the number of records in the test data),
  - 'File Record Data' = (any record file data)
2. RECEIVE BACnet-Error-PDU,
  - Error Class = SERVICES,
  - Error Code = INVALID\_FILE\_ACCESS\_METHOD

##### 9.13.2.2.2 Writing to a File with an Invalid Starting Position

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a file using an invalid start position.

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a file using an invalid start position.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = (any value n:  $n < -1$  |  $n > \text{the maximum supported number of octets}$ ),
  - 'File Data' = (any record data)

2. RECEIVE BACnet-Error-PDU,
 

|               |                             |
|---------------|-----------------------------|
| Error Class = | SERVICES,                   |
| Error Code =  | INVALID_FILE_START_POSITION |

#### 9.13.2.2.3 Writing to a Read Only File

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a read only file.

Configuration Requirements: The IUT shall be configured with a file that does not permit write access. If it is not possible to configure such a file this test may be omitted.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
 

|                         |                                             |
|-------------------------|---------------------------------------------|
| 'File Identifier' =     | (any stream access file that is read only), |
| 'File Start Position' = | 0,                                          |
| 'File Data' =           | (any stream data)                           |
2. RECEIVE  
(BACnet-Error-PDU,
 

|               |                                           |
|---------------|-------------------------------------------|
| Error Class = | PROPERTY,                                 |
| Error Code =  | WRITE_ACCESS_DENIED)   (BACnet-Error-PDU, |
| Error Class = | SERVICES,                                 |
| Error Code =  | FILE_ACCESS_DENIED)                       |

#### 9.13.2.2.4 Writing to a Nonexistent File

Purpose: To verify that an appropriate error is returned if an attempt is made to write to a nonexistent file.

Test Steps:

1. TRANSMIT AtomicWriteFile-Request,
 

|                         |                         |
|-------------------------|-------------------------|
| 'File Identifier' =     | (any nonexistent file), |
| 'File Start Position' = | 0,                      |
| 'File Data' =           | (any stream data)       |
2. RECEIVE BACnet-Error-PDU,
 

|               |                                               |
|---------------|-----------------------------------------------|
| Error Class = | OBJECT,                                       |
| Error Code =  | UNKNOWN_OBJECT   NO_OBJECTS_OF_SPECIFIED_TYPE |

### 9.14 AddListElement Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing AddListElement service requests.

Configuration Requirements: The IUT shall be configured with at least one standard object containing a property whose datatype is a list. The designation "L" shall be used to represent the object identifier for this object in the test description. This list property, designated "ListProp" in the test description shall contain one or more data elements and be capable of storing additional data elements. The property value shall be changeable using the AddListElement service.

#### 9.14.1 Positive AddListElement Service Execution Test

The purpose of this test group is to verify the correct execution of AddListElement service requests under circumstances where the service is expected to be successfully completed.

##### 9.14.1.1 Adding a Single Element

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add a single element to a list.

Test Steps:

1. TRANSMIT ReadProperty-Request,
 

|                         |          |
|-------------------------|----------|
| 'Object Identifier' =   | L,       |
| 'Property Identifier' = | ListProp |
2. RECEIVE ReadProperty-ACK,
 

|                       |    |
|-----------------------|----|
| 'Object Identifier' = | L, |
|-----------------------|----|



- 'Property Identifier' = ListProp,  
 'Property Value' = (any valid value referred to as "InitialList" below)
3. TRANSMIT AddListElement-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp  
 'List of Elements' = (a single element of the correct datatype that is not in InitialList )
  4. RECEIVE BACnet-SimpleACK-PDU
  5. VERIFY (L), ListProp = (a list containing the elements of InitialList + the newly added element)

#### 9.14.1.2 Adding Multiple Elements

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add multiple elements to a list.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp
2. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp,  
 'Property Value' = (any valid value referred to as "InitialList" below)
3. TRANSMIT AddListElement-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp  
 'List of Elements' = (two elements of the correct datatype that are not in InitialList)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (L), ListProp = (a list containing the elements of InitialList + the newly added elements)

#### 9.14.1.3 Adding a Redundant Element

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add a redundant element to a list.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp
2. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp,  
 'Property Value' = (any valid value referred to as "InitialList" below)
3. TRANSMIT AddListElement-Request,  
 'Object Identifier' = L,  
 'Property Identifier' = ListProp  
 'List of Elements' = (a single element that is already contained in InitialList)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (L) ListProp = InitialList

#### 9.14.2 Negative AddListElement Service Execution Tests

The purpose of this test group is to verify the correct execution of AddListElement service requests under circumstances where the service is expected to fail.

##### 9.14.2.1 Adding a List Element to a Property That is Not a List

Purpose: To verify the ability of the IUT to correctly respond to an AddListElement service request to add an element to a property that is not a list.

Test Steps:

1. TRANSMIT AddListElement-Request,

- 'Object Identifier' = (any supported object),  
 'Property Identifier' = (any writable property that is not a list),  
 'List of Elements' = (a single element of the same datatype as the specified property)
2. RECEIVE AddListElement-Error,  
     Error Class = SERVICES,  
     Error Code = PROPERTY\_IS\_NOT\_A\_LIST,  
     'First Failed Element' = 0

#### 9.14.2.2 Adding a List Element With an Invalid Datatype

Purpose: To verify the ability of the IUT to correctly respond to an AddListElement service request to add an element with an invalid datatype to a list.

Notes to Tester: value selected for step 1 is 'inappropriate', not a value which is 'allowed', but not supported by this instance of the property, i.e., it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE\_NOT\_SUPPORTED is only correct when the datatype requested is supported, for example, in a CHOICE, by this property in this object type, but not supported by this instance of the property.

Test Steps:

1. TRANSMIT AddListElement-Request,  
     'Object Identifier' = L,  
     'Property Identifier' = ListProp,  
     'List of Elements' = (a single element with a datatype inappropriate for this property)
2. RECEIVE  
     (AddListElement-Error,  
         Error Class = PROPERTY,  
         Error Code = INVALID\_DATATYPE,  
         'First Failed Element' = 1) |  
     (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE) |  
     (BACnet-Reject-PDU  
         Reject Reason = INVALID\_TAG)

#### 9.14.2.3 An AddListElement Failure Part Way Through a List

Purpose: To verify the ability of the IUT to respond to an AddListElement service request to add multiple elements to a list where one of the elements cannot be added. Upon failure, the AddListElement service should leave the list unchanged.

Notes to Tester: The value selected for step 2 is 'inappropriate', not a value which is 'allowed', but not supported by this instance of the property, i.e., it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE\_NOT\_SUPPORTED is only correct when the datatype requested is supported, for example, in a CHOICE, by this property in this object type, but not supported by this instance of the property. Care must be taken that the inserted error is not likely to cause confusion in the IUT as to which element contains the error.

Test Steps:

1. READ InitialList = (L), ListProp
2. TRANSMIT AddListElement-Request,  
     'Object Identifier' = L,  
     'Property Identifier' = ListProp  
     'List of Elements' = (two or more elements to be added to the list with the second element  
         having the wrong datatype)
2. IF (Protocol\_Revision is present AND Protocol\_Revision >= 7) THEN  
     RECEIVE  
         AddListElement-Error,  
             Error Class = PROPERTY,  
             Error Code = INVALID\_DATATYPE  
             'First Failed Element' = 2  
     | (RECEIVE BACnet-Reject-PDU,  
         Reject Reason = INVALID\_TAG | INVALID\_PARAMETER\_DATA\_TYPE)

```

ELSE
 RECEIVE
 AddListElement-Error,
 Error Class = SERVICES,
 Error Code = INVALID_PARAMETER_DATATYPE
 'First Failed Element' = 2
 | (AddListElement-Error,
 'Error Class' = PROPERTY,
 'Error Code' = INVALID_DATATYPE,
 'First Failed Element' = 2)
 | (BACnet-Reject-PDU,
 Reject Reason = INVALID_TAG | INVALID_PARAMETER_DATA_TYPE)
3. VERIFY (L), ListProp = InitialList

```

### 9.15 RemoveListElement Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing RemoveListElement service requests.

Configuration Requirements: The IUT shall be configured with at least one standard object containing a property whose datatype is a list. The designation "L" shall be used to represent the object identifier for this object in the test description. This list property, designated "ListProp" in the test description shall contain two or more data elements. The property value shall be changeable using the RemoveListElement service.

#### 9.15.1 Positive RemoveListElement Service Execution Tests

The purpose of this test group is to verify the correct execution of RemoveListElement service requests under circumstances where the service is expected to be successfully completed.

##### 9.15.1.1 Removing a Single Element from a List

Purpose: To verify the ability of the IUT to respond to a RemoveListElement service request to remove a single element from a list.

Test Steps:

1. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp
2. RECEIVE ReadProperty-ACK,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp,
  - 'Property Value' = (any valid value referred to as "InitialList" below)
3. TRANSMIT RemoveListElement-Request,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp
  - 'List of Elements' = (a single element of InitialList)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (L), ListProp = (a list containing all of the elements of InitialList except the one removed in step 3)

##### 9.15.1.2 Removing Multiple Elements from a List

Purpose: To verify the ability of the IUT to respond to a RemoveListElement service request to remove multiple elements from a list.

Test Steps:

1. TRANSMIT ReadProperty-Request,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp
2. RECEIVE ReadProperty-ACK,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp,
  - 'Property Value' = (any valid value referred to as "InitialList" below)

3. TRANSMIT RemoveListElement-Request,  
     'Object Identifier' = L,  
     'Property Identifier' = ListProp  
     'List of Elements' = (two or more elements of InitialList)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (L), ListProp = (a list containing all of the elements of InitialList except the ones removed in step3)

### 9.15.2 Negative RemoveListElement Service Execution Tests

The purpose of this test group is to verify the correct execution of RemoveListElement service requests under circumstances where the service is expected to fail.

#### 9.15.2.1 Removing a List Element from a Property That is Not a List

Purpose: To verify the ability of the IUT to correctly respond to a RemoveListElement service request to remove an element from a property that is not a list.

Test Steps:

1. TRANSMIT RemoveListElement-Request,  
     'Object Identifier' = (any supported object),  
     'Property Identifier' = (any writable property that is not a list),  
     'List of Elements' = (a single element of the same datatype as the specified property)
2. RECEIVE RemoveListElement-Error,  
     Error Class = SERVICES,  
     Error Code = PROPERTY\_IS\_NOT\_A\_LIST,  
     'First Failed Element' = 0

#### 9.15.2.2 A RemoveListElement Failure Part Way Through a List

Purpose: To verify the ability of the IUT to respond to a RemoveListElement service request to remove multiple elements from a list where one of the elements cannot be removed. Upon failure, the RemoveListElement service should leave the list unchanged.

Test Steps:

1. READ InitialList = (L), ListProp
2. TRANSMIT RemoveListElement-Request,  
     'Object Identifier' = L,  
     'Property Identifier' = ListProp  
     'List of Elements' = (one element from InitialList, followed by an element of the correct datatype that is not in InitialList, followed by one or more elements from InitialList)
3. IF (Protocol\_Revision is present AND Protocol\_Revision >= 7) THEN  
     RECEIVE RemoveListElement-Error,  
         Error Class = SERVICES,  
         Error Code = LIST\_ELEMENT\_NOT\_FOUND  
         'First Failed Element' = 2  
     ELSE  
         RECEIVE RemoveListElement-Error,
4. VERIFY (L), ListProp = InitialList

### 9.16 CreateObject Service Execution Tests

In each of the tests defined in this clause there is a step where the Object\_List list property of the Device object is read in order to verify that the newly created object appears in the list. This procedure may not work for implementations that do not support segmentation if the Object\_List is long. Under those circumstances an acceptable alternative procedure is to read each element of the Object\_List array one element at a time until an entry is found that contains the newly created object.

#### 9.16.1 Positive CreateObject Service Execution Tests

Test Concept: This clause defines the tests necessary to demonstrate support for executing CreateObject service requests. BACnet does not specify which object types are to be dynamically creatable. Manufacturers are free to make any combination of object types creatable that they wish. The tests procedures described here are generic in the sense that

they can be applied to any object type, including proprietary object types. An IUT must demonstrate that the CreateObject service works correctly for every object type that the PICS claims is dynamically creatable by applying all of these tests to one object of each type.

#### 9.16.1.1 Creating Objects by Specifying the Object Type with No Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Type is used as the object specifier.

Test Steps:

1. READ X1 = Object\_List
2. TRANSMIT CreateObject-Request,  
    'Object Specifier' = (any creatable object type)
3. RECEIVE CreateObject-ACK,  
    'Object Identifier' = (O1, any object identifier of the specified type)
4. CHECK (X1 does not contain O1)
5. READ X2 = Object\_List
6. CHECK (X2 contains O1)
7. VERIFY (O1), (any required property of the specified object) =  
    (any value of the correct datatype for the specified property)

#### 9.16.1.2 Creating Objects by Specifying the Object Identifier with No Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier.

Test Steps:

1. TRANSMIT CreateObject-Request,  
    'Object Specifier' = (any unique object identifier of a type that is creatable and an instance number that is creatable)
2. RECEIVE CreateObject-ACK,  
    'Object Identifier' = (the object identifier specified in step 1)
3. VERIFY (the object identifier of the newly created object),  
    (any required property of the specified object) = (any value of the correct datatype for the specified property)
4. VERIFY (the IUT's Device object), Object\_List = (any object list containing the newly created object)

#### 9.16.1.3 Creating Objects by Specifying the Object Type and Providing Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Type is used as the object specifier and a list of initial property values is provided.

Test Steps:

1. READ X1 = Object\_List
2. TRANSMIT CreateObject-Request,  
    'Object Specifier' = (any creatable object type),  
    'List Of Initial Values' = (a list of one or more properties and their initial values that the IUT will accept)
3. RECEIVE CreateObject-ACK,  
    'Object Identifier' = (any unique object identifier of the specified type)
4. CHECK (X1 does not contain O1)
5. READ X2 = Object\_List
6. CHECK (X2 contains O1)
7. REPEAT X = (properties initialized in the CreateObject-Request) DO {  
    VERIFY (O1), X = (the value specified in the 'List Of Initial Values' parameter of the CreateObject-Request)  
}

#### 9.16.1.4 Creating Objects by Specifying the Object Identifier and Providing Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an Object Identifier is used as the object specifier and a list of initial property values is provided.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any unique object identifier of a type that is creatable and an instance number that is creatable)  
'List Of Initial Values' = (a list of one or more properties and their initial values that the IUT will accept)
2. RECEIVE CreateObject-ACK,  
'Object Identifier' = (the object identifier specified in step 1)
3. REPEAT X = (properties initialized in the CreateObject-Request) DO {  
    VERIFY (the object identifier for the newly created object),  
    X = (the value specified in the 'List Of Initial Values' parameter of the CreateObject-Request)  
}
4. VERIFY (the IUT's Device object), Object\_List = (any object list containing the newly created object)

### 9.16.2 Negative CreateObject Service Execution Tests

The purpose of this test group is to verify correct execution of the CreateObject service requests under circumstances where the service is expected to fail.

#### 9.16.2.1 Attempting to Create an Object That Does Not Have a Unique Object Identifier

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier that already exists in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any object identifier representing an object that already exists having an object type for which dynamic creation is supported)
2. RECEIVE CreateObject-Error,  
    Error Class = OBJECT,  
    Error Code = OBJECT\_IDENTIFIER\_ALREADY\_EXISTS  
'First Failed Element Number' = 0

#### 9.16.2.2 Attempting to Create an Object with an Object Type That is Not Creatable by Specifying the Object Type

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object type that is not dynamically creatable in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any supported object type for which dynamic creation is not supported)
2. RECEIVE CreateObject-Error,  
    Error Class = OBJECT,  
    Error Code = DYNAMIC\_CREATION\_NOT\_SUPPORTED  
'First Failed Element Number' = 0

#### 9.16.2.3 Attempting to Create an Object with an Object Identifier That is Not Creatable by Specifying the Object Identifier

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier for an object type that is not dynamically creatable in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any object identifier having a supported object type for which dynamic creation is not supported)
2. RECEIVE CreateObject-Error,  
    Error Class = OBJECT,  
    Error Code = DYNAMIC\_CREATION\_NOT\_SUPPORTED  
'First Failed Element Number' = 0



3. VERIFY (the IUT's Device object),  
     Object\_List = (any object list that does not contain the object specified in step 1)

Notes to tester: If the IUT limits the instances that can be created, this shall be taken into account when selecting an object identifier in step 1.

#### 9.16.2.4 Attempting to Create an Object with an Object Type Specifier and an Error in the Initial Values

Purpose: To verify the correct execution of the CreateObject service request when an object type is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule\_Default, which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.

Test Steps:

1. READ X1 = Object\_List
2. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any creatable object type),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
3. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
     RECEIVE CreateObject-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
   ELSE  
     RECEIVE CreateObject-Error,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE | OTHER  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
4. CHECK (Verify that the new object was not created)
5. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (object type from step 2),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being an inappropriate datatype)
6. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
     RECEIVE  
     CreateObject-Error,  
     Error Class = PROPERTY,  
     Error Code = INVALID\_DATATYPE  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
     | (BACnet-Reject-PDU  
     Reject Reason = INVALID\_PARAMETER\_DATATYPE | INVALID\_TAG)  
   ELSE  
     RECEIVE  
     CreateObject-Error,  
     Error Class = PROPERTY,  
     Error Code = VALUE\_OUT\_OF\_RANGE | INVALID\_DATATYPE | OTHER  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
     | (BACnet-Reject-PDU  
     Reject Reason = INVALID\_PARAMETER\_DATATYPE | INVALID\_TAG)
7. READ X2 = Object\_List

8. CHECK ( $X1 = X2$ )**9.16.2.5 Attempting to Create an Object with an Object Identifier and an Error in the Initial Values**

Purpose: To verify the correct execution of the CreateObject service request when an object identifier is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Concept: The TD shall attempt to create an object with an object type specifier and the 'List Of Initial Values' parameter containing a value which is out of range. The TD then attempts to create an object with a value of an inappropriate datatype in the 'List Of Initial Values' parameter. The selected datatype is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition, but which is not supported for P1 by the IUT. For instance, Schedule\_Default, which is defined to be of Any primitive datatype, would not be used in this test along with BitString datatype, even where the IUT's Schedule object cannot be configured for scheduling BitString values.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any unique object identifier of a type that is creatable and an instance number that is creatable),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
2. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  4) THEN  
     RECEIVE CreateObject-Error,  
         Error Class = PROPERTY,  
         Error Code = VALUE\_OUT\_OF\_RANGE  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
   ELSE  
     RECEIVE CreateObject-Error,  
         Error Class = PROPERTY,  
         Error Code = VALUE\_OUT\_OF\_RANGE | OTHER  
     'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
3. CHECK (Verify that the new object was not created)
4. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (object identifier from step 1),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being an inappropriate datatype)
5. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  4) THEN  
     RECEIVE  
         CreateObject-Error,  
             Error Class = PROPERTY,  
             Error Code = INVALID\_DATATYPE  
         'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE) |  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_TAG)  
   ELSE  
     RECEIVE  
         CreateObject-Error,  
             Error Class = PROPERTY,  
             Error Code = VALUE\_OUT\_OF\_RANGE | INVALID\_DATATYPE | OTHER  
         'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE | INVALID\_TAG)
6. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (the 'Object Identifier' used in step 1),  
     'Property Identifier' = Object\_Name

7. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
RECEIVE BACnet-Error PDU,  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT  
ELSE  
RECEIVE BACnet-Error PDU  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT | NO\_OBJECTS\_OF\_SPECIFIED\_TYPE | OTHER

#### 9.16.2.6 Deleted Clause

This test has been removed.

#### 9.16.2.7 Attempting to Create a non-Supported Object Type (by Object Type)

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object type that is not supported in the IUT.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any unsupported object type)
2. IF (Protocol\_Revision >= 10) THEN  
RECEIVE CreateObject-Error,  
'Error Class' = OBJECT,  
'Error Code' = UNSUPPORTED\_OBJECT\_TYPE  
'First Failed Element Number' = 0.  
ELSE  
RECEIVE CreateObject-Error,  
'Error Class' = (any valid error class),  
'Error Code' = (any valid error code)  
'First Failed Element Number' = 0
3. VERIFY (the IUT's Device object, Object\_List = (any object list that does not contain the object specified in step 1))

#### 9.16.2.8 Attempting to Create a non-Supported Object Type (by Object Identifier)

Purpose: To verify the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier for an object type that is not supported in the IUT.

Notes to Tester: If the IUT limits the instances that can be created, this shall be taken into account when selecting an object identifier in step 1.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Specifier' = (any object identifier having an unsupported object type)
2. IF (Protocol\_Revision >= 10) THEN  
RECEIVE CreateObject-Error,  
'Error Class' = OBJECT,  
'Error Code' = UNSUPPORTED\_OBJECT\_TYPE  
'First Failed Element Number' = 0  
ELSE  
RECEIVE CreateObject-Error,  
'Error Class' = (any valid error class),  
'Error Code' = (any valid error code)  
'First Failed Element Number' = 0
3. VERIFY (the IUT's Device object, Object\_List = (any object list that does not contain the object specified in step 1))

### 9.17 DeleteObject Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing DeleteObject service requests.

In each of the tests defined in this clause there is a step where the Object\_List list property of the Device object is read in order to verify that the newly deleted object no longer appears in the list. This procedure may not work for

implementations that do not support segmentation if the Object\_List is long. Under those circumstances an acceptable alternative procedure is to read every element of the Object\_List array, one element at a time and verifying that the newly deleted object is not present.

#### 9.17.1 Positive DeleteObject Service Execution Tests

The purpose of this test group is to verify correct execution of the DeleteObject service requests under circumstances where the service is expected to be successfully completed.

##### 9.17.1.1 Successful Deletion of an Object

Purpose: To verify the ability to successfully delete an object.

Configuration Requirements: The IUT shall be configured with an object X that can be deleted.

Test Steps:

1. VERIFY (X), Object\_Name = (any valid value)
2. TRANSMIT DeleteObject-Request,  
'Object Identifier' = X
3. RECEIVE BACnet-SimpleACK-PDU
4. TRANSMIT ReadProperty-Request,  
'Object Identifier' = X,  
'Property Identifier' = Object\_Name
5. RECEIVE BACnet-Error-PDU,  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT
6. VERIFY (X), Object\_List = (any object list that does not contain X)

#### 9.17.2 Negative DeleteObject Service Execution Tests

The purpose of this test group is to verify correct execution of the DeleteObject service requests under circumstances where the service is expected to fail.

##### 9.17.2.1 Attempting to Delete an Object That is Not Deletable

Purpose: To verify the correct response to an attempt to delete an object that is not deletable.

Configuration Requirements: The IUT shall be configured with an object X that can not be deleted.

Test Steps:

1. READ V1 = Object\_Name
2. TRANSMIT DeleteObject-Request,  
'Object Identifier' = X
3. RECEIVE BACnet-Error-PDU,  
Error Class = OBJECT,  
Error Code = OBJECT\_DELETION\_NOT\_PERMITTED
4. VERIFY (X), Object\_Name = V1
5. VERIFY (X), Object\_List = (any object list that contains X)

##### 9.17.2.2 Attempting to Delete an Object That Does Not Exist

Purpose: To verify the correct response to an attempt to delete an object that does not exist.

Test Steps:

1. TRANSMIT DeleteObject-Request,  
'Object Identifier' = X
2. RECEIVE BACnet-Error-PDU,  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT

#### 9.18 ReadProperty Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ReadProperty service requests.

**9.18.1 Positive ReadProperty Service Execution Tests**

The purpose of this test group is to verify correct execution of ReadProperty service requests under circumstances where the service is expected to be successfully completed. Let X be the instance number of the Device Object for the IUT.

**9.18.1.1 Reading the Size of an Array**

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array and the size of the array is requested.

Test Steps:

1. VERIFY (Device, X), Object\_List = (the size of the Object\_List specified in the EPICS), ARRAY INDEX = 0

**9.18.1.2 Reading a Single Element of an Array**

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array and a single element of the array is requested.

Test Steps:

1. READ V = (Device, X), Object\_List ARRAY INDEX=1
2. CHECK (V is of type object-identifier)

**9.18.1.3 Reading a Property From the Device Object using the Unknown Instance**

Purpose: This test case verifies that the IUT can execute ReadProperty service requests when the requested object-identifier references a Device Object with an unknown instance (4194303). Let X be the instance number of the Device Object for the IUT. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Device, 4194303),  
     'Property Identifier' = Object-Identifier
2. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = Object-Identifier,  
     'Property Value' = (Device, X)

Passing Result: The IUT shall respond as indicated conveying the value specified in the EPICS.

**9.18.1.4 Reading Entire Arrays**

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property is an array and the entire array is requested.

Test Steps:

1. VERIFY (Device, X), Object\_List = (the entire Object\_List array)

Passing Result: The IUT shall respond as indicated, conveying values specified in the EPICS. If the object list is too long to return given the APDU and segmentation limitations of the IUT and TD, an Abort message indicating "segmentation not supported" or "buffer overflow" is a passing result. If an Abort message is received and the IUT has another array that is small enough to read in its entirety without segmentation, then this test shall be repeated using that array. A passing result in this case is that the entire array is returned in response to the ReadProperty request.

**9.18.1.5 Reading Properties Based on Data Type**

Purpose: This test case verifies that the IUT can execute ReadProperty service requests for requested properties of each of the supported base data types.

Test Concept: This test is repeated once for each base data type that the IUT supports. For each execution of the test a property, P1, shall be selected that is of the data type being tested and the object containing P1 is designated Object1 in the test description.

Test Steps:

1. READ V = (Object1), P1
2. CHECK (V returns any valid value of the correct data type for property P1)

#### 9.18.1.6 Respects max-segments-accepted bit pattern

Purpose: To verify that the IUT abides by the 'max-segments-accepted' parameter, when the size of the response does require segmentation.

Configuration Requirements: Use a very small 50 octet 'max-APDU-length-accepted' size in the request. The BACnet-Confirmed-Request-PDU shall be one where the response size will exceed 2 times 'max-APDU-length-accepted' and so require at least three segments. If the largest response that the IUT can return is 100 or fewer octets, then this test shall be skipped.

Notes to Tester: An attempt to read the whole Object\_List might suffice. Or a ReadRange or ReadPropertyMultiple or AtomicReadFile request, if any of those services are executed.

Test Steps:

1. TRANSMIT BACnet-Confirmed-Request-PDU,  
'segmented-response-accepted' = TRUE  
'max-segments-accepted' = 2
2. RECEIVE BACnet-Abort-PDU,  
'Abort Reason' = BUFFER\_OVERFLOW

#### 9.18.1.7 Reading Array Properties at Different Array Indexes

Purpose: This test verifies the IUT can execute ReadProperty service requests on a single element in an array property.

Test Concept: This test will execute a ReadProperty service request to read a single element from the selected property by specifying the array-index in the request. Another request is made to read an element of an array where the array index is out of range.

Configuration Requirement: O1 is any object in the IUT database having array property P1 having size X.

Test Steps:

1. VERIFY P1 = X, ARRAY INDEX = 0
2. IF (X>0) THEN  
    READ V = P1, ARRAY INDEX = 1  
    CHECK (V is any valid value of the correct data type for property P1)  
    READ V = P1, Array Index = X  
    CHECK (V is any valid value of the correct data type for property P1)
3. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = O1,  
    'Property Identifier' = P1  
    'Property Array Index' = (X+1)
4. RECEIVE BACnet-Error-PDU,  
    'Error Class' = PROPERTY,  
    'Error Code' = INVALID\_ARRAY\_INDEX

#### 9.18.1.8 ReadProperty of the Network Port Object using the Unknown Instance

Purpose: Verify that the IUT selects the correct object when a Network Port is read using the special object instance 4194303.



Test Concept: Execute a ReadProperty service request specifying 'Object Identifier' = (Network Port, 4194303). Verify that the responding BACnet-user selects the local Network Port object representing the network port through which the request was received.

Configuration Requirements: Let X be the instance number of the Network Port object associated with the network port through which the TD will communicate with the IUT.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = Object-Identifier
2. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = Object-Identifier,  
    'Property Value' = (Network Port, X)
3. REPEAT P = (each property in the specified Network Port object) DO {  
    TRANSMIT ReadProperty-Request through the same port as above,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = P  
    RECEIVE ReadProperty-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = P,  
    'Property Value' = V  
    VERIFY (Network Port, X), P = V  
}

#### 9.18.1.9 ReadProperty Service when Non-BACnet Device Offline

Purpose: To verify that the ReadProperty service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadProperty Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1
5. RECEIVE ReadProperty-ACK,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (V, any valid value)

#### 9.18.2 Negative ReadProperty Service Execution Tests

The purpose of this test group is to verify correct execution of ReadProperty service requests under circumstances where the service is expected to fail.

##### 9.18.2.1 Reading Non-Array Properties with an Array Index

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property value is not an array but an ARRAY INDEX is included in the service request.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (Device, X),

```

'Property Identifier' = Vendor_Name,
'Priority Array Index' = 1
2. IF (Protocol_Revision is present AND Protocol_Revision >= 4) THEN
 RECEIVE BACnet-Error PDU,
 Error Class = PROPERTY,
 Error Code = PROPERTY_IS_NOT_AN_ARRAY
ELSE
 RECEIVE
 (BACnet-Error-PDU,
 Error Class =PROPERTY,
 Error Code =PROPERTY_IS_NOT_AN_ARRAY | INVALID_ARRAY_INDEX) |
 (BACnet-Error-PDU,
 Error Class = SERVICES,
 Error Code = INCONSISTENT_PARAMETERS)

```

#### 9.18.2.2 Reading Array Properties with an Array Index that is Out of Range

Purpose: To verify that the IUT can execute ReadProperty service requests when the requested property value is an array but the ARRAY INDEX is out of range.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (Device, X),  
'Property Identifier' = Object\_List,  
'Priority Array Index' = (a value larger than the size of the Object\_List)
2. RECEIVE BACnet-Error-PDU,  
Error Class = PROPERTY,  
Error Code = INVALID\_ARRAY\_INDEX

#### 9.18.2.3 Reading an Unknown Object

Purpose: To verify that the IUT can execute ReadProperty service requests under circumstances where the requested object does not exist.

Test Concept: The TD attempts to read a property of an object that does not exist in the device.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (any standard object not contained in the IUT's database),  
'Property Identifier' = (any property defined for the specified object)
2. RECEIVE BACnet-Error-PDU,  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT

#### 9.18.2.4 Reading an Unknown Property

Purpose: To verify that the IUT can execute ReadProperty service requests under circumstances where the requested property does not exist.

Test Concept: The TD attempts to read a property that is not defined for the specified object.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (Device, X),  
'Property Identifier' = (any property not defined for the Device object)
2. RECEIVE BACnet-Error-PDU,  
Error Class = PROPERTY,  
Error Code = UNKNOWN\_PROPERTY

### 9.19 ReadPropertyConditional Service Execution Tests

This clause specifies the tests necessary to demonstrate support for executing ReadPropertyConditional service requests. The following subsections contain test requirements instead of specific tests because of the flexibility and complexity of the ReadPropertyConditional service. The details of these tests must be customized for a particular IUT based on an analysis of the IUT's test database. Taken together, the tests should cover a broad range of situations: different Boolean operations, different comparison operators, different property value datatypes, and different properties returned in the response. All BACnet objects must support the Object\_Identifier, Object\_Name, and Object\_Type properties, so these may be convenient properties to use in developing the specific tests for a particular IUT.

#### 9.19.1 'OR' Selection Logic With Matches in the Object Database

**Purpose:** To verify that the IUT correctly executes a ReadPropertyConditional service request that uses 'OR' selection logic and results in a match with one or more objects in the IUT's database.

**Test Concept:** 'OR' selection logic shall be used with two or more selection criteria. The selection criteria shall be chosen so that at least one object satisfies only one of the selection criteria. Preferably, for each selection criterion there should be one or more objects that satisfy that particular criterion. The 'Comparison Value' used in a particular criterion shall be of the same datatype as the property selected by the 'Property Identifier' for at least one object in the IUT's database. The 'List of Property References' parameter shall contain one or more properties that are likely to be present in objects which satisfy the selection criteria,

**Test Steps:** The TD shall transmit a ReadPropertyConditional service request that meets the requirements noted in the Test Concept.

**Notes to Tester:** The IUT shall respond with a correctly encoded BACnet-Complex-ACK. The 'List of Read Access Results' parameter shall contain a list of values limited to objects that satisfy at least one of the selection criteria and properties that were specified in the 'List of Property References' parameter of the request.

#### 9.19.2 'OR' Negative Test

**Purpose:** To verify that the IUT correctly executes a ReadPropertyConditional service request that uses 'OR' selection logic and results in a match with no objects in the IUT's database.

**Test Concept:** 'OR' selection logic shall be used with two or more selection criteria. The selection criteria shall be chosen so that no object satisfies any of the selection criteria. For each selection criterion there shall be one or more objects that contain the property used in that criterion. The 'Comparison Value' used in a particular criterion shall be of the same datatype as the property selected by the 'Property Identifier' for at least one object in the IUT's test database. The 'List of Property References' parameter shall contain one or more arbitrary properties.

**Test Steps:** The TD shall transmit a ReadPropertyConditional service request that meets the requirements of the Test concept.

**Notes to Tester:** The IUT shall respond with a correctly encoded BACnet-Complex-ACK. The 'List of Read Access Results' parameter shall be an empty (zero-length) list.

### 9.20 ReadPropertyMultiple Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ReadPropertyMultiple service requests.

**Configuration Requirements:** The IUT shall be configured with a minimum of two BACnet objects in its database.

**Test Concept:** Two objects shall be selected by the tester from the IUT's database. The various tests consist of reading combinations of properties from one or both of these objects. In the test descriptions the Object Identifier for these objects are designated Object1 and Object2. Properties selected by the tester are designated P1, P2, P3, etc. as needed.

#### 9.20.1 Positive ReadPropertyMultiple Service Execution Tests

The purpose of this test group is to verify correct execution of ReadPropertyMultiple service requests under circumstances where the service is expected to be successfully completed.

##### 9.20.1.1 Reading a Single Property from a Single Object

**Purpose:** To verify the ability to read a single property from a single object.

Test Concept: A single supported property is read from the Device object. The property is selected by the TD and is designated as P1 in the test description.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1 | Object2,  
     'Property Identifier' = P1
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = (the object selected in step 1),  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value)

#### 9.20.1.2 Reading Multiple properties from a Single Object

Purpose: To verify the ability to read multiple properties from a single object. Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1 | Object 2,  
     'Property Identifier' = P1,  
     'Property Identifier' = P2  
     -- (Two properties are required but more may be selected.)
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = (the object selected in step 1),  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value for P1),  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value for P2)  
     -- (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

#### 9.20.1.3 Reading a Single Property from Multiple Objects

Purpose: To verify the ability to read a single property from multiple objects.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Object Identifier' = Object2,  
     'Property Identifier' = P2  
     -- (Two properties are required but more may be selected.)
2. RECEIVE ReadPropertyMultiple-ACK,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value for P1),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value for P2)  
     -- (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

#### 9.20.1.4 Reading Multiple Properties from Multiple Objects

Purpose: To verify the ability to read multiple properties from multiple objects.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Identifier' = P2,  
     'Property Identifier' = P3,

'Object Identifier' = Object2,  
 'Property Identifier' = P4,  
 'Property Identifier' = P5,  
 'Property Identifier' = P6

-- (Two objects must be included but but more may be selected.)

2. RECEIVE ReadPropertyMultiple-ACK,

'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any valid value for P1),  
 'Property Identifier' = P2,  
 'Property Value' = (any valid value for P2),  
 'Property Identifier' = P3,  
 'Property Value' = (any valid value for P3),  
 'Object Identifier' = Object2,  
 'Property Identifier' = P4,  
 'Property Value' = (any valid value for P4),  
 'Property Identifier' = P5,  
 'Property Value' = (any valid value for P5),  
 'Property Identifier' = P6  
 'Property Value' = (any valid value for P6),

-- (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

#### 9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains a specification for an unsupported property.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,

'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Identifier' = P2,  
 'Property Identifier' = (any property, P3, not supported in this object),  
 'Property Identifier' = P4

2. RECEIVE ReadPropertyMultiple-ACK,

'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any valid value for P1),  
 'Property Identifier' = P2,  
 'Property Value' = (any valid value for P2),  
 'Property Identifier' = P3,  
 'Error Class' = PROPERTY,  
 'Error Code' = UNKNOWN\_PROPERTY,  
 'Property Identifier' = P4,  
 'Property Value' = (any valid value for P4),

#### 9.20.1.6 Reading Multiple Properties with Multiple Embedded Access Errors

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for multiple unsupported properties.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,

'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Identifier' = P2,  
 'Property Identifier' = (any property, P3, not supported in this object),  
 'Property Identifier' = (any property, P4, not supported in this object),  
 'Object Identifier' = (any non-existent object, Object2, which is of a type supported by the IUT),  
 'Property Identifier' = P5,

- 'Property Identifier' = P6
2. RECEIVE ReadPropertyMultiple-ACK,
    - 'Object Identifier' = Object1,
    - 'Property Identifier' = P1,
    - 'Property Value' = (any valid value for P1),
    - 'Property Identifier' = P2,
    - 'Property Value' = (any valid value for P2),
    - 'Property Identifier' = P3,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = UNKNOWN\_PROPERTY,
    - 'Property Identifier' = P4,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = UNKNOWN\_PROPERTY,
    - 'Object Identifier' = Object2,
    - 'Property Identifier' = P5,
    - 'Error Class' = OBJECT,
    - 'Error Code' = UNKNOWN\_OBJECT,
    - 'Property Identifier' = P6,
    - 'Error Class' = OBJECT,
    - 'Error Code' = UNKNOWN\_OBJECT

#### 9.20.1.7 Reading ALL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier ALL. One instance of each object-type supported is tested.

Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision ≥ 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
  - TRANSMIT ReadPropertyMultiple-Request,
    - 'Object Identifier' = ObjectX,
    - 'Property Identifier' = ALL
  - RECEIVE ReadPropertyMultiple-ACK,
    - 'Object Identifier' = ObjectX,
    - 'List Of Results' = (a list of all standard properties documented for ObjectX in the EPICS plus all proprietary properties present in ObjectX, each with a valid value, excluding the Property\_List property)

#### 9.20.1.8 Reading OPTIONAL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier OPTIONAL by returning all of the objects's optional properties. The property identifier OPTIONAL means that only those standard properties present in the object that have a conformance code "O" shall be returned.

Notes to Tester: If no optional properties are supported, then an empty 'List of Results' shall be returned for the specified property. If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision ≥ 7, then the entry shall contain Error Class: PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property.

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
  - TRANSMIT ReadPropertyMultiple-Request,
    - 'Object Identifier' = ObjectX,
    - 'Property Identifier' = OPTIONAL



```

RECEIVE ReadPropertyMultiple-ACK,
 'Object Identifier' = ObjectX,
 'List Of Results' = (a list of all standard properties with a conformance code
 of O documented for ObjectX in the EPICS, each with a valid value)
}

```

#### 9.20.1.9 Reading REQUIRED Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier REQUIRED by returning all of the object's required properties.

Notes to Tester: If a property which is not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry is returned, or an error code is returned. If Protocol\_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
 TRANSMIT ReadPropertyMultiple-Request,
 'Object Identifier' = ObjectX,
 'Property Identifier' = REQUIRED
 RECEIVE ReadPropertyMultiple-ACK,
 'Object Identifier' = ObjectX,
 'List Of Results' = (a list of all standard properties with a conformance code
 of R or W documented for ObjectX in the EPICS, each with a
 valid value, excluding the Property\_List property)
 }

#### 9.20.1.10 Reading the Size of an Array

Purpose: To verify that the IUT can execute ReadPropertyMultiple service requests when the requested property is an array and the size of the array is requested.

Test Concept: The test in 9.18.1.1, Reading the Size of an Array, is repeated using ReadPropertyMultiple instead of ReadProperty.

#### 9.20.1.11 Reading a Property From the Device Object using the Unknown Instance

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests when the requested object-identifier references a Device Object with an unknown instance (4194303). Let X be the instance number of the Device Object for the IUT. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,
 'Object Identifier' = (Device, 4194303),
 'Property Identifier' = Object-Identifier
2. RECEIVE ReadPropertyMultiple-ACK,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object-Identifier,
 'Property Value' = (Device, X)

Passing Result: The IUT shall respond as indicated conveying the value specified in the EPICS.

#### 9.20.1.12 Reading Maximum Multiple Properties

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be read using a single ReadPropertyMultiple request.

Test Concept: The object-identifier is read from the device object as many times as can be conveyed in the largest request accepted by the IUT or as can be returned in the largest response that the IUT can generate. The calculation of the maximum request/response size shall be based on the IUT's Max\_APDU\_Length\_Accepted and maximum segments per request/response.

The procedure to determine the number of object-identifiers to use is:

MaxAPDU = IUT's Max\_APDU\_Length\_Accepted  
 MaxRxSegs = IUT's maximum segments accepted per request  
 MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4  
 SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6  
 NonSegRespHdrSize = size of (non-segmented BACnet-ComplexACK-PDU header) = 3  
 SegRespHdrSize = size of (segmented BACnet-ComplexACK-PDU header) = 5  
 ObjIdSize = size of (an Object-Identifier) = 5  
 TagsSize = size of (an open and a close tag) = 2  
 PropIdSize = size of ('Object-Identifier' property Id) = 2

If the IUT does not support receiving segmented requests:

MaxPropsPerRqst =  

$$\frac{(\text{MaxAPDU} - \text{NonSegRqstHdrSize} - \text{ObjIdSize} - \text{TagsSize})}{\text{PropIdSize}} =$$

$$(\text{MaxAPDU} - 11) / 2$$

If the IUT does support receiving segmented requests:

MaxPropsPerRqst =  

$$\frac{((\text{MaxAPDU} - \text{SegRqstHdrSize}) * \text{MaxRxSegs} - \text{ObjIdSize} - \text{TagsSize})}{\text{PropIdSize}} =$$

$$((\text{MaxAPDU} - 6) * \text{MaxRxSegs} - 7) / 2$$

If the IUT does not support sending segmented responses:

MaxPropsPerResp =  

$$\frac{(\text{MaxAPDU} - \text{NonSegRespHdrSize} - \text{ObjIdSize} - \text{TagsSize})}{(\text{PropIdSize} + \text{TagsSize} + \text{ObjIdSize})} =$$

$$(\text{MaxAPDU} - 10) / 9$$

If the IUT does support sending segmented responses:

MaxPropsPerResp =  

$$\frac{((\text{MaxAPDU} - \text{SegRespHdrSize}) * \text{MaxTxSegs} - \text{ObjIdSize} - \text{TagsSize})}{(\text{PropIdSize} + \text{TagsSize} + \text{ObjIdSize})} =$$

$$((\text{MaxAPDU} - 5) * \text{MaxTxSegs} - 7) / 9$$

NumPropertiesToUse = min(MaxPropsPerRqst, MaxPropsPerResp)

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
 'Object Identifier' = (Device, X),  
 'Property Identifier' = Object-Identifier,  
 'Property Identifier' = Object-Identifier,  
 'Property Identifier' = Object-Identifier,  
 ...  
 'Property Identifier' = Object-Identifier
2. RECEIVE ReadPropertyMultiple-ACK,  
 'Object Identifier' = (Device, X),  
 'Property Identifier' = Object-Identifier,  
 'Property Value' = (Device, X),  
 'Property Identifier' = Object-Identifier,  
 'Property Value' = (Device, X),  
 'Property Identifier' = Object-Identifier,  
 'Property Value' = (Device, X),  
 ...  
 'Property Identifier' = Object-Identifier,  
 'Property Value' = (Device, X)

**9.20.1.13 Reading Properties Based on Data Type**

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests for requested properties of each of the supported base data types.

Test Concept: The test 9.18.1.5 Reading Properties Based on Data Type is repeated using ReadPropertyMultiple instead of ReadProperty.

**9.20.1.14 ReadPropertyMultiple of the Network Port Object using the Unknown Instance**

Purpose: Verify that the IUT selects the correct object when a Network Port is read using the special object instance 4194303.

Test Concept: Execute a ReadPropertyMultiple service request specifying 'Object Identifier' = (Network Port, 4194303). The responding BACnet-user selects the local Network Port object representing the network port through which the request was received.

Configuration Requirements: Let X be the instance number of the Network Port object associated with the network port through which the TD will communicate with the IUT.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = Object-Identifier
2. RECEIVE ReadPropertyMultiple-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = Object-Identifier,  
    'Property Value' = (Network Port, X)
3. REPEAT P = (each property in the specified Network Port object) {  
    TRANSMIT ReadPropertyMultiple-Request through the same port as above,  
    'Object Identifier' = (Network Port, 4194303),  
    'Property Identifier' = P  
    RECEIVE ReadPropertyMultiple-ACK,  
    'Object Identifier' = (Network Port, X),  
    'Property Identifier' = P,  
    'Property Value' = V  
    VERIFY (Network Port, X), P=V  
  }

**9.20.1.15 ReadPropertyMultiple Service when Non-BACnet Device Offline**

Purpose: To verify that the ReadPropertyMultiple service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadPropertyMultiple-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1
5. RECEIVE ReadPropertyMultiple-ACK,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (any valid value)

### 9.20.2 Negative ReadPropertyMultiple Service Execution Tests

The purpose of this test group is to verify correct execution of ReadPropertyMultiple service requests under circumstances where the service is expected to fail.

#### 9.20.2.1 Reading a Single, Unsupported Property from a Single Object

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for a single unsupported property.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1 | Object2,  
     'Property Identifier' = (any property, P1, that is not supported in the selected object)
2. RECEIVE BACnet-Error-PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = UNKNOWN\_PROPERTY  
     ( ReadPropertyMultiple-ACK,  
     'Object Identifier' = (the object identifier from step 1),  
     'Property Identifier' = P1,  
     'Error Class' = PROPERTY,  
     'Error Code' = UNKNOWN\_PROPERTY)

#### 9.20.2.2 Reading Multiple Properties with Access Errors for Every Property

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for only unsupported properties.

Test Concept: The selections for objects and properties for this test shall consist of either objects that are not supported, properties that are not supported for the selected objects, or a combination of the two such that there are no object, property combinations that represent a supported property.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Identifier' = P2,  
     'Property Identifier' = P3,  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Property Identifier' = P5,  
     'Property Identifier' = P6
2. RECEIVE  
     (BACnet-Error-PDU,  
     'Error Class' = OBJECT | PROPERTY,  
     'Error Code' = (any valid error code for the returned error class) ) |  
     (ReadPropertyMultiple-ACK,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Error Class' = OBJECT | PROPERTY,  
     'Error Code' = (any valid error code for the returned error class),  
     'Property Identifier' = P2,  
     'Error Class' = OBJECT | PROPERTY,  
     'Error Code' = (any valid error code for the returned error class),  
     'Property Identifier' = P3,  
     'Error Class' = OBJECT | PROPERTY,  
     'Error Code' = (any valid error code for the returned error class),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Error Class' = OBJECT | PROPERTY,  
     'Error Code' = (any valid error code for the returned error class),

'Property Identifier' = P5,  
 'Error Class' = OBJECT | PROPERTY,  
 'Error Code' = (any valid error code for the returned error class),  
 'Property Identifier' = P6,  
 'Error Class' = OBJECT | PROPERTY,  
 'Error Code' = (any valid error code for the returned error class) )

### 9.20.2.3 Reading a Single Non-Array Property with an Array Index

Purpose: This test case verifies that the IUT can execute ReadPropertyMultiple service requests when the requested property value is not an array but an ARRAY INDEX is included in the service request. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT ReadPropertyMultiple-Request,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = Vendor\_Name,  
     'Priority Array Index' = 1
2. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
     RECEIVE  
         (BACnet-Error-PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY) |  
         (ReadPropertyMultiple-ACK,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = Vendor\_Name,  
         'Priority Array Index' = 1,  
         'Error Class' = PROPERTY,  
         'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY)
- ELSE  
     RECEIVE  
         (BACnet-Reject-PDU,  
         'Reject Reason' = INCONSISTENT\_PARAMETERS) |  
         (BACnet-Reject-PDU,  
         'Reject Reason' = INVALID\_TAG) |  
         (BACnet-Error-PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = INVALID\_ARRAY\_INDEX) |  
         (ReadPropertyMultiple-ACK,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = Vendor\_Name,  
         'Priority Array Index' = 1,  
         'Error Class' = PROPERTY,  
         'Error Code' = INVALID\_ARRAY\_INDEX) |  
         (ReadPropertyMultiple-ACK,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = Vendor\_Name,  
         'Priority Array Index' = 1  
         'Error Class' = SERVICES,  
         'Error Code' = INCONSISTENT\_PARAMETERS) |  
         (BACnet-Error-PDU,  
         'Error Class' = SERVICES,  
         'Error Code' = INCONSISTENT\_PARAMETERS) |  
         (BACnet-Error-PDU,  
         'Error Class' = PROPERTY,  
         'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY) |  
         (ReadPropertyMultiple-ACK,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = Vendor\_Name,

'Priority Array Index' = 1  
 'Error Class' = PROPERTY,  
 'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY)

### 9.21 ReadRange Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ReadRange service requests.

Configuration Requirements: When testing a Log\_Buffer property, the IUT shall be configured with a logging object that contains a set of known log records. The TD must have exact knowledge of the log records in order to evaluate the results of the tests. The value of the Enable property shall be FALSE so that the Log\_Buffer does not change during the tests.

When testing a property other than the Log\_Buffer, steps shall be taken to ensure that the value of the property does not change outside the control of the tester during the execution of the test.

The following sample buffer is used as explanation for the tests in this section.

Sample Log\_Buffer, (Trend Log, Instance 1)

| Arbitrary Record Designation | Position (index) | Implied Sequence # | Timestamp (Date excluded for clarity) | LogDatum                         |
|------------------------------|------------------|--------------------|---------------------------------------|----------------------------------|
| a                            | 1                | 16                 | 13:01:00.00                           | log-status, buffer-purged        |
| b                            | 2                | 17                 | 13:02:00.00                           | log-status, log-disabled = FALSE |
| c                            | 3                | 18                 | 13:05:00.00                           | real-value = 5.0                 |
| d                            | 4                | 19                 | 13:10:00.00                           | real-value = 10.0                |
| e                            | 5                | 20                 | 13:15:00.00                           | real-value = 15.0                |
| f                            | 6                | 21                 | 13:16:00.00                           | log-status, log-disabled = TRUE  |
| g                            | 7                | 22                 | 13:21:00.00                           | log-status, log-disabled = FALSE |
| h                            | 8                | 23                 | 13:25:00.00                           | real-value = 25.0                |
| i                            | 9                | 24                 | 13:30:00.00                           | real-value = 30.0                |
| j                            | 10               | 25                 | 13:35:00.00                           | real-value = 35.0                |
| k                            | 11               | 26                 | 13:36:00.00                           | log-status, log-disabled = TRUE  |

#### 9.21.1 Positive ReadRange Service Execution Tests

The purpose of this test group is to verify the correct execution of ReadRange service requests under circumstances where the service is expected to be successfully completed.

##### 9.21.1.1 Reading All Items in the List

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return all of the available data items.

Test Concept: A list property, P is read using ReadRange by position with no range specified. It is verified that the complete list is returned.

Configuration Requirements: Property P is configured with a value that is small enough to be returned in a single ReadRange response. If the IUT cannot be configured in this manner, see the Notes to Tester.

Notes to Tester: The property P may have more items than can be returned in a single message. Under these circumstances 'Result Flags' will have the value {TRUE, FALSE, TRUE} and the 'Item Count' and 'Item Data' parameters would reflect the actual number of items that were able to be returned.

Test Steps:

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P
2. RECEIVE Read-Range-ACK,

'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Result Flags' = {TRUE, TRUE, FALSE},  
 'Item Count' = (the number of entries in P),  
 'Item Data' = (all of the entries in P)

#### 9.21.1.2 Reading Items by Position with Positive Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a position and the number of items after that position to return.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the list property, P. This range is specified using the 'By Position' option and a positive value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.

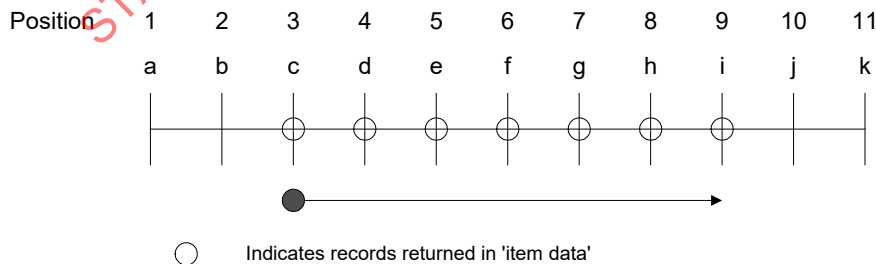
Configuration Requirements: A list property, P, is configured with N items.

Test Steps:

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Reference Index' = (any value x:  $1 \leq x \leq N$ ),  
 'Count' = (any value y:  $0 < y \leq N - x + 1$ )
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Result Flags' = {?, ?, FALSE},  
 'Item Count' = y,  
 'Item Data' = (all of the items specified in order of increasing position. The items specified include the item at the index specified by x, plus (y-1) items following.)

Test Example (using the sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (Trend Log, Instance 1),  
 'Property Identifier' = Log\_Buffer,  
 'Reference Index' = 3,  
 'Count' = 7
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (Trend Log, Instance 1),  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {FALSE, FALSE, FALSE},  
 'Item Count' = 7,  
 'Item Data' = Records < c, d, e, f, g, h, i > in that order.



#### 9.21.1.3 Reading Items by Position with Negative Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a position and the number of items before that position to return.



Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the list property, P. This range is specified using the 'By Position' option and a negative value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

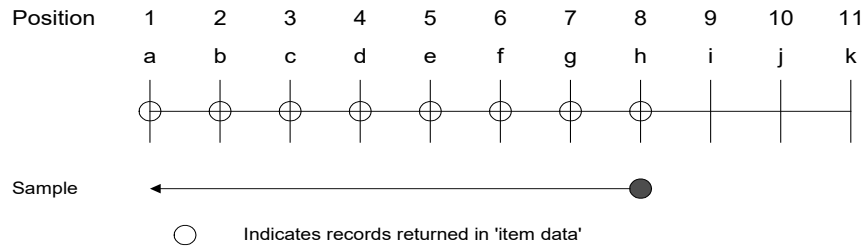
Configuration Requirements: A list property, P, is configured with N items and  $N > 1$ .

Test Steps:

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Reference Index' = (any value x:  $1 \leq x \leq N$ ),  
 'Count' = (any value y:  $y < 0$  AND  $|y| \leq x$ )
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Result Flags' = {?, ?, FALSE},  
 'Item Count' =  $|y|$ ,  
 'Item Data' = (all of the items specified in order of increasing position. The items specified include the item at the index specified by x, plus  $|y|-1$  items preceding.)  
 --read the first Data-Item
3. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Reference Index' = 2  
 'Count' = -2
4. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Result Flags' = {TRUE, ?, FALSE},  
 'Item Count' = 2,  
 'Item Data' = (The items specified include the item at the index 1 and 2 in order of increasing position)  
 --read the last Data-Item
5. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Reference Index' = N,  
 'Count' = (any value y:  $y < 0$  AND  $|y| \leq N$ )
6. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (the log object configured for this test),  
 'Property Identifier' = P,  
 'Result Flags' = {?, TRUE, FALSE},  
 'Item Count' =  $|y|$ ,  
 'Item Data' = (all of the items specified in order of increasing position. The items specified include the item at the index specified by N, plus  $|y|-1$  items preceding.)

Test Example (using the sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (Trend Log, Instance 1),  
 'Property Identifier' = Log\_Buffer,  
 'Reference Index' = 8,  
 'Count' = -8
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (Trend Log, Instance 1),  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {TRUE, FALSE, FALSE},  
 'Item Count' = 8  
 'Item Data' = Records < a, b, c, d, e, f, g, h > in that order.



#### 9.21.1.4 Reading Items by Time

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a time and the number of items after that time to return.

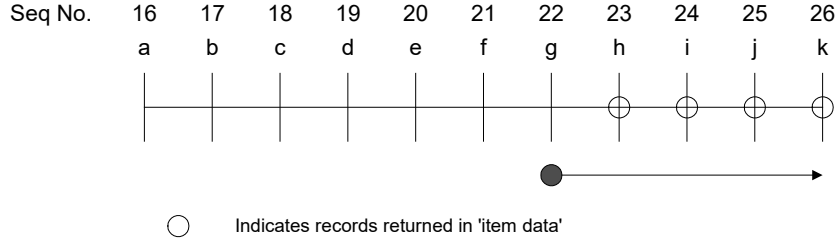
**Test Concept:** A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Time' option and a positive value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

**Test Steps:**

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Time' = (any value x: x is older (of earlier time) than the time of an entry in the buffer which has a sequence number of S, and x is newer than or equal to the time of any preceding entry),
  - 'Count' = (any value y:  $0 < y \leq \text{Total\_Record\_Count} - S + 1$ )
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {?, ?, FALSE},
  - 'Item Count' = y,
  - 'Item Data' = (all of the specified trend records in order of increasing sequence number. The items specified include the first item with a timestamp newer than x, plus (y-1) items following.)
  - 'First Sequence Number' = S

**Test Example (using sample buffer at beginning of section):**

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Time' = 13:21:00.00
  - 'Count' = 4
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE, TRUE, FALSE},
  - 'Item Count' = 4,
  - 'Item Data' = Records < h, i, j, k > in that order.
  - 'First Sequence Number' = 23



#### 9.21.1.4.1 Reading Items by Time with Negative Count

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a time and the number of items after that time to return.

**Test Concept:** A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Time' option and a negative value for 'Count'. The 'Reference Time' selected,  $x$ , should be newer than the last time in the buffer. The 'Reference Time' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

**Configuration Requirements:** Configure the logging object such that it contains at least 3 items in the Log\_Buffer.

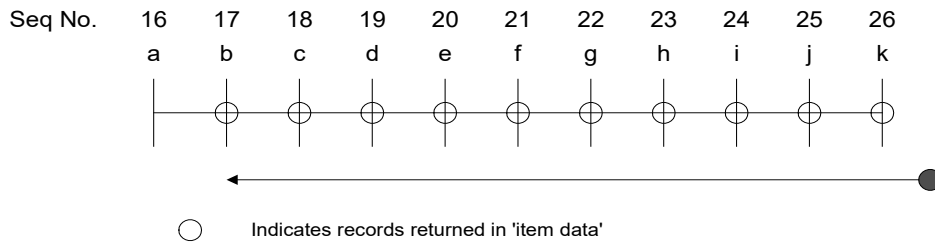
**Test Steps:**

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Time' = ( $x$ , selected as described above),
  - 'Count' = (any value  $y$ :  $0 < |y| \leq \text{number of records in the buffer}$ )
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {?, TRUE, FALSE},
  - 'Item Count' =  $|y|$ ,
  - 'Item Data' = (All of the specified trend records in order of increasing sequence number. The items specified include the last item with a timestamp older than  $x$ , plus  $|y|-1$  items preceding.)
  - 'First Sequence Number' = (Total\_Record\_Count -  $|y| + 1$ )

**Notes to Tester:** All items returned shall contain a timestamp older than the time specified by reference time parameter. The items returned shall be the last 'count' items from the log buffer. If there is an entry in the Log\_Buffer with a timestamp that exactly matches the 'Reference Time' parameter, that entry shall not be included in the 'Item Data'.

**Test Example (using the sample buffer at beginning of section):**

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Time' = 13:40:00.00,
  - 'Count' = -10
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (Trend Log, Instance 1),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result Flags' = {FALSE, TRUE, FALSE},
  - 'Item Count' = 10,
  - 'Item Data' = (records < b, c, d, e, f, g, h, i, j, k > in that order.)
  - 'First Sequence Number' = 17



#### 9.21.1.5 Deleted Clause

This clause has been removed.

#### 9.21.1.6 Reading a Range of Items that do not Exist (by Position)

Purpose: To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified by position range.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known not to be in the list property P. The IUT shall respond by returning an empty list.

Configuration Requirements: The list property, P, is configured with N items.

Test Steps:

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Reference Index' = (any value x:  $x > N$ ),  
 'Count' = (any value y:  $y > 0$ )
2. RECEIVE Read-Range-ACK,  
 'Object Identifier' = (the object configured for this test),  
 'Property Identifier' = P,  
 'Result flags' = {TRUE, TRUE, FALSE},  
 'Item Count' = 0,  
 'Item Data' = (an empty list)

#### 9.21.1.7 Reading a Range of Items that do not Exist (Using by Sequence)

Purpose: To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified criteria.

Test Concept: A ReadRange request is transmitted by the TD requesting a specified sequence number and count of items known not to be in the Log Buffer. The IUT shall respond by returning an empty list.

Test Steps:

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the log object configured for this test),  
 'Property Identifier' = Log\_Buffer,  
 'Reference Sequence Number' = (any value that will result in no items being present )  
 'Count' = (any non-zero number)
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = (the log object configured for this test),  
 'Property Identifier' = Log\_Buffer,  
 'Result flags' = {FALSE, FALSE, FALSE},  
 'Item Count' = 0,  
 'Item Data' = (an empty list)  
 'First Sequence Number' = (should be absent)

Test Example (using sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,

- |                               |                          |
|-------------------------------|--------------------------|
| 'Object Identifier' =         | (Trend Log, Instance 1), |
| 'Property Identifier' =       | Log_Buffer,              |
| 'Reference Sequence Number' = | 34                       |
| 'Count' =                     | 4                        |
2. RECEIVE ReadRange-ACK,
- |                         |                          |
|-------------------------|--------------------------|
| 'Object Identifier' =   | (Trend Log, Instance 1), |
| 'Property Identifier' = | Log_Buffer,              |
| 'Result flags' =        | {FALSE, FALSE, FALSE},   |
| 'Item Count' =          | 0,                       |
| 'Item Data' =           | (an empty list)          |

#### 9.21.1.8 Reading a Range of Items that do Not Exist (Using by Time)

Purpose: To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified criteria.

Test Concept: A ReadRange request is transmitted by the TD requesting a specified reference time and count of items known not to be in the Log\_Buffer. The IUT shall respond by returning an empty list.

Test Steps:

- TRANSMIT ReadRange-Request,
 

|                         |                                                        |
|-------------------------|--------------------------------------------------------|
| 'Object Identifier' =   | (the log object configured for this test),             |
| 'Property Identifier' = | Log_Buffer,                                            |
| 'Reference Time' =      | (any value that will result in no items being present) |
| 'Count' =               | (any non-zero number)                                  |
- RECEIVE ReadRange-ACK,
 

|                           |                                            |
|---------------------------|--------------------------------------------|
| 'Object Identifier' =     | (the log object configured for this test), |
| 'Property Identifier' =   | Log_Buffer,                                |
| 'Result flags' =          | {FALSE, FALSE, FALSE},                     |
| 'Item Count' =            | 0,                                         |
| 'Item Data' =             | (an empty list)                            |
| 'First Sequence Number' = | (should be absent)                         |

Test Example (using sample buffer at beginning of section):

- TRANSMIT ReadRange-Request,
 

|                         |                          |
|-------------------------|--------------------------|
| 'Object Identifier' =   | (Trend Log, Instance 1), |
| 'Property Identifier' = | Log_Buffer,              |
| 'Reference Time' =      | 12:00:00.00              |
| 'Count' =               | -10                      |
- RECEIVE ReadRange-ACK,
 

|                         |                          |
|-------------------------|--------------------------|
| 'Object Identifier' =   | (Trend Log, Instance 1), |
| 'Property Identifier' = | Log_Buffer,              |
| 'Result flags' =        | {FALSE, FALSE, FALSE},   |
| 'Item Count' =          | 0,                       |
| 'Item Data' =           | (an empty list)          |

#### 9.21.1.9 Reading Items by Sequence with Positive Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a sequence number and the number of items after that sequence to return.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Sequence' option and a positive value for 'Count'. The 'Reference Sequence Number' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.

Test Steps:

- TRANSMIT ReadRange-Request,
 

|                         |                                            |
|-------------------------|--------------------------------------------|
| 'Object Identifier' =   | (the log object configured for this test), |
| 'Property Identifier' = | Log_Buffer,                                |

'Reference Sequence Number' = (any value  $x$ :  $(\text{Total\_Record\_Count} - \text{Record\_Count} + 1) \leq x \leq (\text{Total\_Record\_Count} - y + 1)$ ), 'Count' = (any value  $y$ :  $0 < y \leq \text{Record\_Count}$ )

2. RECEIVE ReadRange-ACK,

'Object Identifier' = (the log object configured for this test),

'Property Identifier' = Log\_Buffer,

'Result Flags' = {?, ?, FALSE},

'Item Count' =  $y$ ,

'Item Data' = (All of the specified records in the order of increasing sequence number.  
The items specified are all items with the sequence number in the range  
of  $x$  through  $(x+y-1)$  in that order)

'First Sequence Number' =  $x$

Test Example (using sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,

'Object Identifier' = 20:1,

'Property Identifier' = Log\_Buffer,

'Reference Sequence Number' = 16,

'Count' = 11

2. RECEIVE ReadRange-ACK,

'Object Identifier' = 20:1,

'Property Identifier' = Log\_Buffer,

'Result Flags' = {TRUE, TRUE, FALSE},

'Item Count' = 11,

'Item Data' = Records < a, b, c, d, e, f, g, h, i, j, k > in that order.

'First Sequence Number' = 16

#### 9.21.1.10 Reading Items by Sequence with Negative Count

Purpose: To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a sequence number and the number of items after that sequence to return.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Sequence' option and a negative value for 'Count'. The 'Reference Sequence Number' and 'Count' are selected so that the results can be conveyed in a single acknowledgment.

Test Steps:

1. TRANSMIT ReadRange-Request,

'Object Identifier' = (the log object configured for this test),

'Property Identifier' = Log\_Buffer,

'Reference Sequence Number' = (any value  $x$ :  $(\text{Total\_Record\_Count} - \text{Record\_Count} + 2) < x \leq \text{Record\_Count}$ ),

'Count' = (any value  $y$ :  $0 < |y| < (\text{Record\_Count} - (\text{Total\_Record\_Count} - x) + 1)$ )

2. RECEIVE ReadRange-ACK,

'Object Identifier' = (the log object configured for this test),

'Property Identifier' = Log\_Buffer,

'Result Flags' = {?, ?, FALSE},

'Item Count' =  $y$ ,

'Item Data' = (All of the specified records in order of increasing sequence number. The  
items specified are all items in the range of  $(x-|y|+1)$  through  $x$  in  
that order.)

'First Sequence Number' =  $(x - |y| + 1)$

Test Example (using sample buffer at beginning of section):

1. TRANSMIT ReadRange-Request,

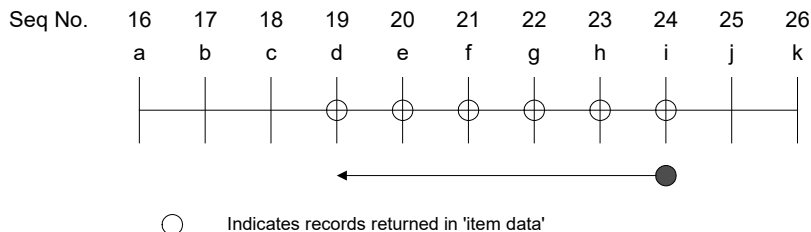
'Object Identifier' = (Trend Log, 1),

'Property Identifier' = Log\_Buffer,

'Reference Sequence Number' = 24,

'Count' = -6

2. RECEIVE ReadRange-ACK,  
'Object Identifier' = (Trend Log, 1),  
'Property Identifier' = Log\_Buffer,  
'Result Flags' = {FALSE, FALSE, FALSE},  
'Item Count' = 6,  
'Item Data' = Records < d, e, f, g, h, i > in that order.  
'First Sequence Number' = 19



#### 9.21.1.11 Deleted Clause

This clause has been removed.

#### 9.21.1.12 Status/Failure logging

Moved to 7.3.2.24.20

#### 9.21.1.13 Reading Items with Negative Count and MOREITEMS

Purpose: To verify that the IUT correctly responds to a ReadRange service request by returning the correct subset of items when a sequence number or byTime and a negative count are requested, and the count is more items than the IUT actually returns.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using a negative value for 'Count.' The TD shall be configured such that its Max APDU Length Accepted, Segmented Response Accepted, in combination with the chosen 'Count' selected, mean that the results cannot be conveyed in a single ReadRange-Ack, thus forcing the MOREITEMS flag to be TRUE in the response.

Test Steps:

1. TRANSMIT ReadRange-Request,  
'Object Identifier' = (the logging object configured for this test),  
'Property Identifier' = Log\_Buffer,  
'Reference Sequence Number' = (any value x: known to be in the Log\_Buffer),  
'Count' = (any value y:  $y < 0$  and which forces the MOREITEMS flag  
TRUE in the response)
2. RECEIVE ReadRange-ACK,  
'Object Identifier' = (the logging object configured for this test),  
'Property Identifier' = Log\_Buffer,  
'Result Flags' = {?, ?, TRUE},  
'Item Count' = (any value z:  $0 < z < |y|$ ),  
'Item Data' = (the specified z records in order of increasing sequence number.  
The items specified are all items in the range of  $(x - z + 1)$   
through x  
in that order),  
'First Sequence Number' =  $(x - z + 1)$

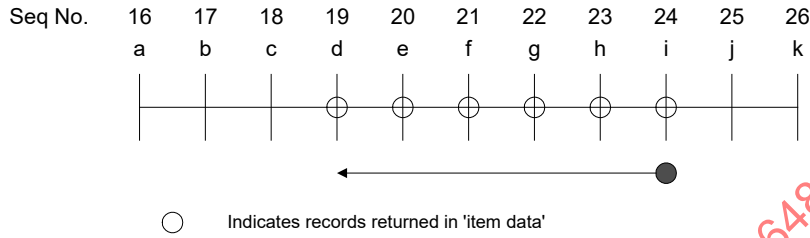
Notes to Tester: Although expressed as a bySequence exchange, these could alternately be expressed byTime.

Test Example (using sample buffer shown in diagram below):

1. TRANSMIT ReadRange-Request,  
'Object Identifier' = 20:1,  
'Property Identifier' = Log\_Buffer,



- 'Reference Sequence Number' = 24,  
 'Count' = -9
2. RECEIVE ReadRange-ACK,  
 'Object Identifier' = 20:1,  
 'Property Identifier' = Log\_Buffer,  
 'Result Flags' = {FALSE, FALSE, TRUE},  
 'Item Count' = 6,  
 'Item Data' = Records < d, e, f, g, h, i > in that order.  
 'First Sequence Number' = 19



#### 9.21.1.14 ReadRange Support for All List Properties

Purpose: To verify that all list properties of all objects can be read using the 3 by position forms of the ReadRange service.

Configuration Requirements: The IUT must be configured with at least one non-empty list property.

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {
  - REPEAT Y = (all list properties in object X) DO {
    - TRANSMIT ReadRange-Request
      - 'Object Identifier' = X,
      - 'Property Identifier' = Y,
    - RECEIVE
      - (ReadRange-ACK
        - 'Object Identifier' = X,
        - 'Property Identifier' = Y,
        - 'Result Flags' = (?, ?, ?),
        - 'Item Count' = (C: up to number of items in Y)
        - 'Item Data' = (the first C elements of Y) )
      - | (ReadRange-ACK
        - 'Object Identifier' = X,
        - 'Property Identifier' = Y,
        - 'Result Flags' = (FALSE, FALSE, FALSE),
        - 'Item Count' = (C = 0)
        - 'Item Data' = ( )
    - IF (C > 0) THEN
      - TRANSMIT ReadRange-Request
        - 'Object Identifier' = X,
        - 'Property Identifier' = Y,
        - 'Reference Index' = 1,
        - 'Count' = (C: any valid positive value)
      - RECEIVE ReadRange-ACK
        - 'Object Identifier' = X,
        - 'Property Identifier' = Y,
        - 'Result Flags' = (TRUE, ?, ?),
        - 'Item Count' = (C2: up to C)
        - 'Item Data' = (the first C2 elements of Y)
      - TRANSMIT ReadRange-Request
        - 'Object Identifier' = X,
        - 'Property Identifier' = Y,
        - 'Reference Index' = (the number of elements in Y),

```

 'Count' = (C: any valid negative value)
 RECEIVE ReadRange-ACK
 'Object Identifier' = X,
 'Property Identifier' = Y,
 'Result Flags' = (?, TRUE, ?),
 'Item Count' = (C2: up to abs(C))
 'Item Data' = (the last C2 elements of Y)
 }
}

```

#### 9.21.1.15 ReadRange Service when Non-BACnet Device Offline

Purpose: To verify that the ReadRange Service executes successfully when a non-BACnet device is offline.

Test Concept: Object1 is an object which contains information from a non-BACnet device. The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which contains a dynamic value derived from the data in the non-BACnet device is read from the IUT.

Test Steps:

1. CHECK (any vendor-specified indication, that the non-BACnet device is online)
2. MAKE (the non-BACnet device go offline)
3. MAKE (the IUT notice that the non-BACnet device is offline)
4. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1
5. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value)

#### 9.21.2 Negative ReadRange Service Execution Tests

##### 9.21.2.1 Attempting to Read a Property That Does not Exist

Purpose: To verify the correct execution of the ReadRange service request when the requested property does not exist. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Configuration Requirements: If all the list properties applicable for the object under testing are supported, then this test shall be skipped.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (any object that exists in the IUT),
  - 'Property Identifier' = (any list property applicable for that object but not supported by the IUT),
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = UNKNOWN\_PROPERTY

##### 9.21.2.2 Attempting to Read a Property That is not a List

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not a list. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (any object that exists in the IUT),
  - 'Property Identifier' = (any non-list property supported by and present in the IUT),
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,

'Error Code' = PROPERTY\_IS\_NOT\_A\_LIST

### 9.21.2.3 Attempting to Read a non-Array Property with an Array Index

Purpose: To verify the correct execution of the ReadRange service request when the requested property is not an array of lists. This test is only applied to devices with a Protocol\_Revision of 10 or higher.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (any object that exists in the IUT),
  - 'Property Identifier' = (any non-array list property supported by and present in the IUT),
  - 'Property Array Index' = (any valid value)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY

### 9.21.2.4 Reading a Range of Items that do not Exist (by Position)

Purpose: To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified criteria.

Test Concept: A ReadRange request is transmitted by the TD requesting a range of items known not to be in the Log\_Buffer. The IUT shall respond by returning an empty list.

Test Steps:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = (any value that will result in no items being present)
  - 'Count' = (any non-zero number)
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result flags' = {FALSE, FALSE, FALSE},
  - 'Item Count' = 0,
  - 'Item Data' = (an empty list)
  - 'First Sequence Number' = (should be absent)

Test Example 1 (using index that does not exist):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = 0
  - 'Count' = 5
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result flags' = {FALSE, FALSE, FALSE},
  - 'Item Count' = 0,
  - 'Item Data' = (an empty list)

Test Example 2 (using index that does not exist):

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Index' = Index= Buffer\_Size + 1
  - 'Count' = -20
2. RECEIVE ReadRange-ACK,
  - 'Object Identifier' = (the log object configured for this test),

'Property Identifier' = Log\_Buffer,  
 'Result flags' = {FALSE, FALSE, FALSE},  
 'Item Count' = 0,  
 'Item Data' = (an empty list)

## 9.22 WriteProperty Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing WriteProperty service requests.

Test Concept: The tester shall select an object from the IUT's database that has writable properties suitable for the purpose of the test case. In the test descriptions the Object\_Identifier for this object is designated Object1.

### 9.22.1 Positive WriteProperty Service Execution Tests

The purpose of this test group is to verify correct execution of WriteProperty service requests under circumstances where the service is expected to be successfully completed.

#### 9.22.1.1 Writing a Single Element of an Array

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is an array and a single array element is written.

Test Concept: The TD shall select an object in the IUT that contains a writable array property. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writing array values it shall be configured with at least one writable property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1 ARRAY INDEX = (any value N:  $1 \leq N \leq$  the size of the array)
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Array Index' = N  
     'Property Value' = (any valid value of the correct datatype subject to the restrictions specified  
                           in the EPICS as defined in 4.4.2 for this array, except the value X read  
                           for this element in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2), ARRAY INDEX = N

#### 9.22.1.2 Writing a Commandable Property Without a Priority

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is commandable but a priority is not specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is commandable and has no internal algorithm writing to it at priority 16. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports commandable properties that have no internal algorithm writing at priority 16, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), Priority\_Array, ARRAY INDEX = 16
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = Present\_Value,  
     'Property Value' = (any valid value of the correct datatype for this property subject to  
                           the restrictions specified in the EPICS as defined in 4.4.2,  
                           except the value X read in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), Priority\_Array = (the value used in step 2), ARRAY INDEX = 16

**9.22.1.3 Writing a Non-Commandable Property with a Priority**

Purpose: To verify that the IUT can execute WriteProperty service requests when the property is not commandable but a priority is specified.

Test Concept: The TD shall select an object in the IUT that contains a writable property that is not commandable and has no internal algorithm writing to it. If no suitable property can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports non-commandable properties that have no internal algorithm writing to them, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Priority' = (any valid priority)  
     'Property Value' = (any valid value defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1)
3. RECEIVE BACnet-BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

**9.22.1.4 Writing an Array Size**

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to the array size of a writable, non-fixed size array property.

Test Concept: The TD shall select an object in the IUT that contains a writable array property of a non-fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports writable non-fixed size array properties, it shall be configured with at least one writable non-fixed size array property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1 ARRAY INDEX = 0
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Array Index' = 0  
     'Property Value' = (any valid array size defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1[0] = (the value used in step 2)

**9.22.1.5 Writing to Properties Based on Data Type**

Purpose: This test case verifies that the IUT can execute WriteProperty service requests to specific data types supported by the IUT.

Test Concept: For the specified base data type, the TD shall select an object in the IUT that contains a writable property of that data type. This property is designated P1.

Configuration Requirements: The IUT shall be configured with at least one writable property of the specified data type to be used for this test.

Test Steps:

1. X = READ (Object1), P1

2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value defined for this property subject to the  
                           restrictions specified in the EPICS as defined in 4.4.2,  
                           except the value X determined in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

### 9.22.2 Negative WriteProperty Service Execution Tests

The purpose of this test group is to verify correct execution of WriteProperty service requests under circumstances where the service is expected to fail.

#### 9.22.2.1 Writing Non-Array Properties with an Array Index

Purpose: To verify that the IUT can execute WriteProperty service requests when the property value is not an array but an array index is included in the service request.

Test Concept: The TD shall select an object in the IUT that contains a writable scalar property designated P1. An attempt will be made to write to this property using an ARRAY INDEX. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the correct datatype for this property subject to  
                           the restrictions specified in the EPICS as defined in 4.4.2, except  
                           the value X read in step 1),  
     'Property Array Index' = (any positive integer)
3. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
     RECEIVE BACnet-Error PDU,  
     Error Class =PROPERTY,  
     Error Code =PROPERTY\_IS\_NOT\_AN\_ARRAY  
   ELSE  
     RECEIVE BACnet-Error PDU,  
     Error Class = SERVICES,  
     Error Code =INCONSISTENT\_PARAMETERS
4. VERIFY (Object1), P1 = X

#### 9.22.2.2 Writing Array Properties with an Array Index that is Out of Range

Purpose: To verify that the IUT can execute WriteProperty service requests when the requested property value is an array but the array index is out of range.

Test Concept: The TD shall select an object in the IUT that contains a writable array property designated P1. An attempt will be made to write to this property using an ARRAY INDEX that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,

'Property Identifier' = P1,  
 'Property Value' = (any valid value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1),  
 'Property Array Index' = (any value that is larger than the current size of the array)

3. RECEIVE BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = INVALID\_ARRAY\_INDEX
4. VERIFY (Object1), P1 = X

#### 9.22.2.3 Writing with a Property Value Having the Wrong Datatype

Purpose: To verify that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but which is not compliant with the property definition given by the BACnet standard.

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value with an invalid datatype)
3. RECEIVE (BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = INVALID\_DATATYPE)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_TAG)
4. VERIFY (Object1), P1 = X

#### 9.22.2.4 Writing with a Property Value that is Out of Range

Purpose: To verify that the IUT can execute WriteProperty service requests when an attempt is made to write a value that is outside of the supported range.

Test Concept: The TD attempts to write to a property using a value that is outside of the supported range. If the IUT does not contain any writable properties that have restricted ranges, then this test shall be skipped.

Notes to Tester: The value used in step 2 shall be of the correct datatype. For bit string types, the bit count shall be correct, for Date and Time values, the value shall be within the range defined by the standard for the datatype, for constructed values, the constructed value shall match the structure defined by the ASN.1, and all field values shall be within the ranges defined by the standard for those field values. In this test, the error code OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED is permitted only if the value written would require the device to exhibit non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (Object1, any object with writable properties),  
     'Property Identifier' = (P1, any writable property with a restricted range of values),  
     'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. IF (Protocol\_Revision is present AND Protocol\_Revision >= 4) THEN  
     RECEIVE  
         BACnet-Error PDU,



```

 Error Class = PROPERTY,
 Error Code = VALUE_OUT_OF_RANGE
 | BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED)
ELSE
 RECEIVE
 BACnet-Error-PDU,
 Error Class = PROPERTY,
 Error Code = VALUE_OUT_OF_RANGE)
 | BACnet-Error-PDU,
 'Error Class' = PROPERTY,
 'Error Code' = OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
 | BACnet-Reject-PDU,
 Reject Reason = PARAMETER_OUT_OF_RANGE)

```

4. VERIFY (Object1), P1 = X

#### 9.22.2.5 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the object specified in the service request does not exist. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, that does not exist in the IUT. Object1 shall be of a type supported by the IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE BACnet-Error PDU,  
 Error Class = OBJECT,  
 Error Code = UNKNOWN\_OBJECT

Passing Result: While OBJECT::UNKNOWN\_OBJECT is the desired error for this condition, in some implementations other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are: PROPERTY::UNKNOWN\_PROPERTY, PROPERTY::WRITE\_ACCESS\_DENIED, PROPERTY::INVALID\_DATATYPE, PROPERTY::VALUE\_OUT\_OF\_RANGE and RESOURCES::NO\_SPACE\_TO\_WRITE\_PROPERTY.

#### 9.22.2.6 Writing To Non-Existent Properties

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the property specified in the service request is not supported by the object specified in the service request. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object in the IUT, designated Object1. The TD shall select a property, designated P1, that is not supported by the specific object instance. An attempt will be made to write to this property.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any valid value for this property)
2. RECEIVE BACnet-Error PDU,  
 Error Class = PROPERTY,

Error Code = UNKNOWN\_PROPERTY

#### 9.22.2.7 Writing To Non-Writable Properties

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the property specified in the service request is not writable. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a non-writable property designated P1. An attempt will be made to write to this property. If no object supports non-writable properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = WRITE\_ACCESS\_DENIED

#### 9.22.2.8 Writing An Object\_Name With A Value That Is Already In Use

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when an Object\_Name property is written to with a value already in use by a different object in the device. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable Object\_Name property. An attempt will be made to write to this property with a value that is in use by the Object\_Name property of another object in the device. If no object supports writable Object\_Name properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = Object\_Name,  
     'Property Value' = (an Object\_Name value already in use by another object)
2. RECEIVE BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = DUPLICATE\_NAME

#### 9.22.2.9 Writing Non-Array Read-only Property with an Array Index

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value when the property value is not an array, but an array index is included in the service request, and the property specified in the service request is not writable.

Test Concept: Select an object, designated Object1, in the IUT that contains a non-writable scalar property designated P1. An attempt will be made to write to this property with an array index included. If no object supports non-writable scalar properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property)  
     'Property Array Index' = (any positive integer)
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 4) THEN  
     RECEIVE BACnet-Error PDU,  
     'Error Class' = PROPERTY,  
     'Error Code' = WRITE\_ACCESS\_DENIED | PROPERTY\_IS\_NOT\_AN\_ARRAY

```

ELSE
 RECEIVE (BACnet-Error PDU,
 'Error Class' = SERVICES,
 'Error Code' = INCONSISTENT_PARAMETERS) |
 (BACnet-Error PDU,
 'Error Class' = PROPERTY,
 'Error Code' = WRITE_ACCESS_DENIED | PROPERTY_IS_NOT_AN_ARRAY)

```

#### 9.22.2.10 Resizing a writable fixed size array property

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WriteProperty service.

Test Concept: Select an object (O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Test Steps:

1. READ X = (O1), P1 ARRAY INDEX = 0
2. WRITE P1 = (Entire Array with any valid value greater than Array Size X)
3. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
4. VERIFY (O1), P1 = X, ARRAY INDEX = 0
5. WRITE P1 = (Entire Array with any valid value less than Array Size X)
6. RECEIVE BACnet-Error PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE
7. VERIFY (O1), P1 = X, ARRAY INDEX = 0
8. WRITE P1 = (any valid value greater than Array Size X), ARRAY INDEX = 0
9. RECEIVE BACnet-Error PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE | WRITE\_ACCESS\_DENIED
10. VERIFY (O1), P1 = X, ARRAY INDEX = 0,
11. WRITE P1 = (any valid value less than Array Size X), ARRAY INDEX = 0
12. RECEIVE BACnet-Error PDU,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE | WRITE\_ACCESS\_DENIED
13. VERIFY (O1), P1 = X, ARRAY INDEX = 0

#### 9.22.2.11 Writing a Property Value Related to Non-supported Optional Functionality

Purpose: To verify that the IUT responds correctly to WriteProperty service requests when the value provided would require the device to exhibit non-supported optional functionality.

Test Concept: A writable property, P1, is selected for which the standard defines optional behavior which the IUT does not support, and the writing of specific values to the property would require the device to exhibit the non-supported behavior. The TD attempts to write to the property using a selected value, V1, which would require the IUT to exhibit non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = (Object1, the object containing a writable P1),
  - 'Property Identifier' = (P1, any writable property with a restricted range of values),
  - 'Property Value' = (V1, a value that would require a non-supported optional functionality)
3. RECEIVE BACnet-Error-PDU
  - 'Error Class' = PROPERTY,
  - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED
4. VERIFY (Object1), P1 = X

### 9.23 WritePropertyMultiple Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing WritePropertyMultiple service requests.

Configuration Requirements: The WritePropertyMultiple service execution tests require that the IUT be configured with a minimum of two BACnet objects in its database that contain writable properties. The Object\_Identifier of these objects are designated Object1 and Object2 in the test descriptions.

#### 9.23.1 Positive WritePropertyMultiple Service Execution Tests

The purpose of this test group is to verify correct execution of WritePropertyMultiple service requests under circumstances where the service is expected to be successfully completed.

##### 9.23.1.1 Writing a Single Property to a Single Object

Purpose: To verify the ability to write a single property to a single object.

Test Concept: This test case attempts to write to a single scalar property, P1, that is not commandable. If no such writable property exists the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1, any valid value of the appropriate datatype for this property subject to  
         the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = V1

##### 9.23.1.2 Writing Multiple properties to a Single Object

Purpose: To verify the ability to write multiple properties to a single object.

Test Concept: This test case attempts to write to multiple scalar properties, P1 and P2, that are not commandable. If two such writable properties don't exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Configuration Requirements: If the IUT supports any object that has two writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured, if possible, with writable array or commandable properties and the test steps modified to account for this variation. If no object type is supported that has two or more writable properties this test may be omitted. The IUT must support either the configuration required for this test or a configuration required for test 9.23.1.3

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1, any valid value of the appropriate datatype for this property subject to  
         the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)  
     'Property Identifier' = P2,  
     'Property Value' = (V2, any valid value of the appropriate datatype for this property subject to  
         the restrictions specified in the EPICS as defined in 4.4.2,

except the value Y read in step 2)

4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = V1
6. VERIFY (Object1), P2 = V2

### 9.23.1.3 Writing a Single Property to Multiple Objects

Purpose: To verify the ability to write a single property from multiple objects.

Test Concept: This test case attempts to write to single scalar properties, P1 and P2, that reside in different objects but are not commandable. If two such writable properties don't exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object2), P2
3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (V1, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1)
  - 'Object Identifier' = Object2,
  - 'Property Identifier' = P2,
  - 'Property Value' = (V2, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y read in step 2)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = V1
6. VERIFY (Object2), P2 = V2

### 9.23.1.4 Writing Multiple Properties to Multiple Objects

Purpose: To verify the ability to write multiple properties to multiple objects.

Test Concept: This test case attempts to write properties, P1 and P2, that reside in Object1, and properties P3 and P4 that reside in Object2. P1, P2, P3 and P4 are not commandable properties. If four such writable properties do not exist the test can be modified to write to an array property or to a commandable property with a write priority high enough to ensure that the commandable property's value will change.

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
3. READ Z = (Object2), P3
4. READ A = (Object2), P4
5. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (V1, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1)
  - 'Property Identifier' = P2,
  - 'Property Value' = (V2, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value Y read in step 2)
  - 'Object Identifier' = Object2,
  - 'Property Identifier' = P3,
  - 'Property Value' = (V3, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2,

except the value Z read in step 3)  
 'Object Identifier' = Object2,  
 'Property Identifier' = P4,  
 'Property Value' = (V4, any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value A read in step 4)

6. RECEIVE BACnet-BACnet-SimpleACK-PDU
7. VERIFY (Object1), P1 = V1
8. VERIFY (Object2), P2 = V2
9. VERIFY (Object1), P3 = V3
10. VERIFY (Object2), P4 = V4

#### 9.23.1.5 Writing a Non-Commandable Property with a Priority

Purpose: To verify that the IUT can execute WritePropertyMultiple service requests when the property is not commandable but a priority is specified.

Test Concept: Repeat test in 9.22.1.3, Writing a Non-Commandable Property with a Priority, using WritePropertyMultiple instead of WriteProperty.

#### 9.23.1.6 Writing a Commandable Property Without a Priority

Purpose: To verify that the IUT can execute WritePropertyMultiple service requests when the property is commandable but a priority is not specified.

Test Concept: Repeat test in 9.22.1.2, Writing a Commandable Property Without a Priority, using WritePropertyMultiple instead of WriteProperty.

#### 9.23.1.7 Writing Maximum Multiple Properties

Purpose: This test case verifies that IUT does not arbitrarily restrict the number of properties that can be written to it using a single WritePropertyMultiple request to object O1.

Test Concept: A writable property is written to an object in the IUT as many times as can be conveyed in the largest request accepted by the IUT. The calculation of the maximum request size shall be based on the IUT's Max\_APDU\_Length\_Accepted and maximum segments per request.

The procedure to determine the number of values to use is:

MaxAPDU = IUT's Max\_APDU\_Length\_Accepted  
 MaxRxSegs = IUT's maximum segments accepted per request  
 MaxTxSegs = IUT's maximum segments generated per response

NonSegRqstHdrSize = size of (non-segmented BACnetConfirmed-RequestPDU header) = 4  
 SegRqstHdrSize = size of (segmented BACnetConfirmed-RequestPDU header) = 6  
 ObjIdSize = size of (an Object-Identifier) = 5  
 TagsSize = size of (an open and a close tag) = 2

PropIdSize = size of (chosen property Id) = depends on property ID and includes ARRAY INDEX size if required  
 ValueSize = size of (chosen property value) = depends on property and value chosen

If the IUT does not support receiving segmented requests:

NumPropertiesToWrite =  

$$\frac{(\text{MaxAPDU} - \text{NonSegRqstHdrSize} - \text{ObjIdSize} - \text{TagsSize})}{(\text{PropIdSize} + \text{TagsSize} + \text{ValueSize})} = \frac{(\text{MaxAPDU} - 11)}{(\text{PropIdSize} + 2 + \text{ValueSize})}$$

If the IUT does support receiving segmented requests:

NumPropertiesToWrite =  

$$\frac{((\text{MaxAPDU} - \text{SegRqstHdrSize}) * \text{MaxRxSegs} - \text{ObjIdSize} - \text{TagsSize})}{\text{PropIdSize}} = \frac{((\text{MaxAPDU} - 6) * \text{MaxRxSegs} - 7)}{2}$$

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = (O1),  
     'Property Identifier' = P1,  
     'Priority Array Index' = A1,     -- only if required  
     'Property Value' = V1,  
     ...  
     'Property Identifier' = P1,  
     'Priority Array Index' = A1,     -- only if required  
     'Property Value' = V1
2. RECEIVE BACnet-BACnet-SimpleACK-PDU
3. VERIFY O1, P1 = V1

#### 9.23.1.8 Writing to Properties Based on Data Type

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests to any data type supported by the IUT.

Test Concept: For the specified data type, the TD shall select an object in the IUT that contains a writable property of that data type. This property is designated P1.

Configuration Requirements: Configure the IUT with at least one writable property of the data type that can be used for this test.

Test Steps:

1. READ V = Object1, P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value defined for this property subject to the restrictions  
                             specified in the EPICS as defined in Clause 4.4.2,  
                             except the value, V, read in step 1)
3. RECEIVE BACnet-SimpleACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

#### 9.23.1.9 Writing an Array Size

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests to the array size of a writable, non-fixed size array property.

Test Concept: Repeat test 9.22.1.4 Writing an Array Size using WritePropertyMultiple instead of WriteProperty.

### 9.23.2 Negative WritePropertyMultiple Service Execution Tests

The purpose of this test group is to verify correct execution of WritePropertyMultiple service requests under circumstances where the service is expected to fail.

#### 9.23.2.1 Writing Multiple Properties with a Property Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for an unsupported property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is not supported for this object. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the unsupported property used for this test.



Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1, any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)  
     'Property Identifier' = (P2, an unsupported property for this object)  
     'Property Value' = (any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2)
3. RECEIVE WritePropertyMultiple-Error,  
     Error Class = PROPERTY,  
     Error Code = UNKNOWN\_PROPERTY,  
     objectIdentifier = Object1,  
     propertyIdentifier = P2
4. VERIFY (Object1), P1 = V1

#### 9.23.2.2 Writing Multiple Properties with an Object Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for an unsupported object.

Test Concept: An attempt is made to write to a single property in two different objects. The first object is supported and the property is writable. The second object is not supported. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 and P1 will be used to designate the writable object and property used for this test. The designation BadObject will be used to indicate an object that is not supported.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1, any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2)
3. RECEIVE WritePropertyMultiple-Error,  
     Error Class = OBJECT,  
     Error Code = UNKNOWN\_OBJECT,  
     objectIdentifier = BadObject,  
     propertyIdentifier = P2
4. VERIFY (Object1), P1 = V1

#### 9.23.2.3 Writing Multiple Properties with a Write Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the 'List of Write Access Specifications' contains a specification for a read only property.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported and writable. The second property is supported but read only. The objective is to verify that an appropriate error response is returned and that all writes up to the first failed write attempt take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description Object1 will be used to designate the object, P1 the writable property, and P2 the read only property used for this test.

Test Steps:

1. READ X = (Object1), P1
2. READ Y = (Object1), P2
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (V1, any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)  
     'Property Identifier' = P2,  
     'Property Value' = (V2, any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value Y read in step 2)
4. RECEIVE WritePropertyMultiple-Error,  
     Error Class = PROPERTY,  
     Error Code = WRITE\_ACCESS\_DENIED,  
     objectIdentifier = Object1,  
     propertyIdentifier = P2
5. VERIFY (Object1), P1 = V1
6. VERIFY (Object1), P2 = Y

#### 9.23.2.4 Writing Non-Array Properties with an Array Index

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property value is not an array but an array index is included in the service request. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable scalar property designated P1 having a value X. An attempt will be made to write to this property using an array index, Y. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are scalars, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject  
         to the restrictions specified in the EPICS as defined in 4.4.2,  
         except the value X read in step 1)  
     'Property Array Index' = (Y, any positive integer)
3. RECEIVE WritePropertyMultiple-Error,  
     Error Class = PROPERTY,  
     Error Code = PROPERTY\_IS\_NOT\_AN\_ARRAY,  
     objectIdentifier = Object1,  
     propertyIdentifier = P1
4. VERIFY (Object1), P1 = X

**9.23.2.5 Writing Array Properties with an Array Index that is Out of Range**

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the requested property value is an array but the array index is out of range. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable array property designated P1 having a value X. An attempt will be made to write to this property using an array index Y, that is out of range. If no suitable object can be found, then this test shall be omitted.

Configuration Requirements: If the IUT supports any writable properties that are arrays, it shall be configured with at least one such property that can be used for this test.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value X read in step 1)
  - 'Property Array Index' = (Y, any value that is larger than the supported size of the array)
3. RECEIVE WritePropertyMultiple-Error,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_ARRAY\_INDEX,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Array Index' = Y
4. VERIFY (Object1), P1 = X

**9.23.2.6 Writing with a Property Value Having the Wrong Datatype**

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using a datatype that the IUT supports but which is not compliant with the property definition given by the BACnet standard

Configuration Requirements: The value to be written shall not be of a datatype which is compliant with the property definition. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any value with an invalid datatype)
3. RECEIVE
  - WritePropertyMultiple-Error,
    - Error Class = PROPERTY,
    - Error Code = INVALID\_DATATYPE,
    - objectIdentifier = Object1,
    - propertyIdentifier = P1
  - | BACnet-Reject-PDU
    - 'Reject Reason' = INVALID\_PARAMETER\_DATATYPE
  - | BACnet-Reject-PDU
    - 'Reject Reason' = INVALID\_TAG
4. VERIFY (Object1), P1 = X

### 9.23.2.7 Writing with a Property Value that is Out of Range

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that is outside of the supported range.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. The TD attempts to write to this property using a value that is outside of the supported range.

Notes to Tester: In this test, the error code OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED is permitted only if the value written would require the device to exhibit non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = (Object1, any object with writable properties),  
     'Property Identifier' = (P1, any property with a restricted range of values),  
     'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. IF (Protocol\_Revision < 4) THEN  
     RECEIVE  
         (WritePropertyMultiple-Error,  
         'Error Class' = PROPERTY,  
         'Error Code' = VALUE\_OUT\_OF\_RANGE,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1) |  
         (WritePropertyMultiple-Error,  
         'Error Class' = PROPERTY,  
         'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1) |  
         (BACnet-Reject-PDU,  
         'Reject Reason' = PARAMETER\_OUT\_OF\_RANGE)  
     ELSE  
         RECEIVE  
         (WritePropertyMultiple-Error,  
         'Error Class' = PROPERTY,  
         'Error Code' = VALUE\_OUT\_OF\_RANGE,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1) |  
         (WritePropertyMultiple-Error,  
         'Error Class' = PROPERTY,  
         'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED,  
         'Object Identifier' = Object1,  
         'Property Identifier' = P1)
4. VERIFY (OBJECT1), P1 = X

### 9.23.2.8 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the object specified in the service request does not exist. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, that does not exist in the IUT. Object1 shall be of a object type supported by IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property)

2. RECEIVE WritePropertyMultiple-Error,  
     Error Class = OBJECT,  
     Error Code = UNKNOWN\_OBJECT,  
     objectIdentifier = Object1,  
     propertyIdentifier = P1

Passing Result: While OBJECT::UNKNOWN\_OBJECT is the desired error for this condition, in some implementations, other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are: PROPERTY::UNKNOWN\_PROPERTY, PROPERTY::WRITE\_ACCESS\_DENIED, PROPERTY::INVALID\_DATATYPE, PROPERTY::VALUE\_OUT\_OF\_RANGE and RESOURCES::NO\_SPACE\_TO\_WRITE\_PROPERTY.

#### 9.23.2.9 Writing To Non-Existent Properties

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property specified in the service request is not supported by the object specified in the service request. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object in the IUT, designated Object1. The TD shall select a property, designated P1, that is not supported by the object. An attempt will be made to write to this property.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for the property),
2. RECEIVE BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = UNKNOWN\_PROPERTY,  
     objectIdentifier = Object1,  
     propertyIdentifier = P1

#### 9.23.2.10 Writing To Non-Writable Properties

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the property specified in the service request is not writable. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object 1, in the IUT that contains a non-writable property designated P1. An attempt will be made to write to this property. If no object supports non-writable properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE WritePropertyMultiple-Error,  
     Error Class = PROPERTY,  
     Error Code = WRITE\_ACCESS\_DENIED,  
     objectIdentifier = Object1,  
     propertyIdentifier = P1

#### 9.23.2.11 Writing An Object\_Name With A Value That Is Already In Use

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an Object\_Name property is written to with a value already in use by a different object in the device. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable Object\_Name property. An attempt will be made to write to this property with a value that is in use by the Object\_Name property of another object in the device. If no object supports writable Object\_Name properties, then this test shall be omitted.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = Object\_Name,  
     'Property Value' = (an Object\_Name value already in use by another object)
2. RECEIVE WritePropertyMultiple-Error,  
     Error Class = PROPERTY,  
     Error Code = DUPLICATE\_NAME

#### 9.23.2.12 WritePropertyMultiple Reject Test

Purpose: This test case verifies that the IUT does not send a Reject-PDU after applying part of a WritePropertyMultiple.

Test Concept: Two writable properties, P1 and P2, are written to the IUT, but the portion of the WritePropertyMultiple specifying P2 is made invalid by omitting the 'Property Value' parameter. If the IUT returns a Reject, then the value of the first property is checked to ensure it has not changed.

Test Steps:

1. READ OldValue = O1, P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = (NewValue: any value other than OldValue that would be accepted by the IUT for P1)  
     'Object Identifier' = O2,  
     'Property Identifier' = P2
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = SERVICES,  
     'Error Code' = INVALID\_TAG  
     'Object Identifier' = O2  
     'Property Identifier' = P2 |  
     RECEIVE BACnet-Reject-PDU,  
     'Reject Reason' = INVALID\_TAG  
         | MISSING\_REQUIRED\_PARAMETER  
         | INCONSISTENT\_PARAMETERS  
         | INVALID\_PARAMETER\_DATA\_TYPE  
         | TOO\_MANY\_ARGUMENTS
4. IF (a WritePropertyMultiple-Error was received in step 3) THEN  
     VERIFY (O1), P1 = NewValue  
     ELSE -- a Reject-PDU was received  
     VERIFY (O1), P1 = OldValue

#### 9.23.2.13 Resizing a Writable Fixed Size Array Property Using WritePropertyMultiple Service

Purpose: This test case verifies that the IUT correctly responds to an attempt to resize a writable fixed size array property using WritePropertyMultiple service.

Test Concept: Select an object(O1) in the IUT that contains a writable array property of a fixed size. This property is designated P1. If no suitable object can be found, then this test shall be omitted.

Test Steps:

1. READ X = (O1), P1, ARRAY INDEX = 0
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,

- 'Property Identifier' = P1,  
'Property Value' = (Entire Array with any valid value greater than Array Size X)
3. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
'ObjectIdentifier' = O1,  
'PropertyIdentifier' = P1
  4. VERIFY (O1), P1= X, ARRAY INDEX = 0
  5. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (Entire Array with any valid value less than Array Size X)
  6. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
'ObjectIdentifier' = O1,  
'PropertyIdentifier' = P1
  7. VERIFY (O1), P1= X, ARRAY INDEX = 0
  8. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value greater than Array Size X),  
'Property Array Index' = 0
  9. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
'ObjectIdentifier' = O1,  
'PropertyIdentifier' = P1  
'Property Array Index'=0
  10. VERIFY (O1), P1= X, ARRAY INDEX = 0
  11. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value less than Array Size X),  
'Property Array Index' = 0
  12. RECEIVE WritePropertyMultiple-Error,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX | VALUE\_OUT\_OF\_RANGE,  
'ObjectIdentifier' = O1,  
'PropertyIdentifier' = P1  
'Property Array Index'= 0
  13. VERIFY (O1), P1= X, ARRAY INDEX = 0

#### 9.23.2.14 Writing First Element of 'List of Write Access Specifications' with Object Access Error

**Purpose:** To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported object and all writes after the first failed write attempt do not take place.

**Test Concept:** An attempt is made to write to a single property in two different objects. The first object is not supported. The second object is supported, and the property is writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

**Configuration Requirements:** If the IUT supports any writable scalar properties that are not commandable it shall be configured with one for use in this test. If no such properties are supported the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description O2 and P2 will be used to designate the writable object and property having value X used for this test. The designation Bad Object will be used to indicate an object that is not supported or not present in IUT database P1 is any valid Property Identifier.



Test Steps:

1. VERIFY (O2), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P1,  
     'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)  
     'Object Identifier' = O2,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P1 |  
     (RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = OBJECT,  
     'Error Code' = UNSUPPORTED\_OBJECT\_TYPE,  
     'Object Identifier' = BadObject,  
     'Property Identifier' = P1)
4. VERIFY (O2), P2 = X

#### 9.23.2.15 Writing First Element of 'List of Write Access Specifications' with a Write Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for a read only property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is supported but read only. The second property is supported and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable, it shall be configured with one for use in this test. If no such properties are supported, the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description, O1 will be used to designate the object, P1 the read only property having value X, P2 the writable property having value Y used for this test.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O1), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1,  
     'Property Value' = X,  
     'Property Identifier' = P2,  
     'Property Value' = (any valid value not equal to Y)
4. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = WRITE\_ACCESS\_DENIED,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1
5. VERIFY (O1), P2 = Y

#### 9.23.2.16 WritePropertyMultiple Reject Test for First Element of 'List of Write Access Specifications'

Purpose: This test case verifies that if IUT does sends a Reject-PDU or Error-PDU, then the write attempt for the remaining element of 'List of Write Access Specifications' do not take place.

Test Concept: Two writable properties, P1 having value X and P2 having value Y, are written to the IUT, but the portion of the WritePropertyMultiple specifying P1 is made invalid by omitting the 'Property Value' parameter. The value of the properties are checked to ensure that it has not changed.

Test Steps:

1. VERIFY (O1), P1= X
2. VERIFY (O2), P2=Y
3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Object Identifier' = O2,
  - 'Property Identifier' = P2
  - 'Property Value' = (Any valid value not equal to Y))
4. RECEIVE
  - WritePropertyMultiple-Error,
  - 'Error Class' = SERVICES,
  - 'Error Code' = INVALID\_TAG
  - 'Object Identifier' = O1
  - 'Property Identifier' = P1) |
  - (RECEIVE BACnet-Reject-PDU,
  - 'Reject Reason' = INVALID\_TAG
    - | MISSING\_REQUIRED\_PARAMETER
    - | INCONSISTENT\_PARAMETERS
    - | INVALID\_PARAMETER\_DATA\_TYPE
    - | TOO\_MANY\_ARGUMENTS)
4. VERIFY (O1), P1 = X
5. VERIFY (O2), P2 = Y

#### 9.23.2.17 Writing First Element of 'List of Write Access Specifications' with a Property Access Error

Purpose: To verify the ability to correctly execute a WritePropertyMultiple service request for which the first element of the 'List of Write Access Specifications' contains a specification for an unsupported property and all writes after the first failed write attempt do not take place.

Test Concept: An attempt is made to write to two properties in a single object. The first property is not supported for this object. The second property is supported for this object and writable. The objective is to verify that an appropriate error response is returned and that all writes after the first failed write attempt do not take place.

Configuration Requirements: If the IUT supports any writable scalar properties that are not commandable, it shall be configured with one for use in this test. If no such properties are supported, the IUT shall be configured with a writable array or commandable property and the test steps modified to account for this variation. In the test description, O1 will be used to designate the object, P1 the unsupported property, and P2 the writable property having value X used.

Test Steps:

1. VERIFY (O1), P2 = X
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value of the appropriate datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2)
  - 'Property Identifier' = P2,
  - 'Property Value' = (any valid value not equal to X),
3. RECEIVE WritePropertyMultiple-Error,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = UNKNOWN\_PROPERTY,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1
4. VERIFY (O1), P2 = X

### 9.23.2.18 Writing a Property Value Related to a Non-supported Optional Functionality

Purpose: To verify that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that would require a non-supported optional functionality

Test Concept: The TD attempts to write to a property using a value that would require a non-supported optional functionality.

Test Steps:

1. READ X = (Object1), P1
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = (Object1, any object with writable properties),  
     'Property Identifier' = (P1, any property with a restricted range of values),  
     'Property Value' = (a value that would require a non-supported optional functionality)
3. RECEIVE WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1
4. VERIFY (Object1), P1 = X

### 9.23.2.19 Date Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special date field values to ensure that the property does not accept them. A date is selected which is within the date range that the IUT will accept for the property. The value, V1, written to the property is the date D1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {

1. TRANSMIT WritePropertyMultiple-Request  
     'Object Identifier' = (O1, any object appropriate for this test),  
     'Property Identifier' = P1,  
     'Property Value' = (V1 updated with the special value SV)
2. RECEIVE  
     WritePropertyMultiple-Error,  
     'Error Class' = PROPERTY,  
     'Error Code' = VALUE\_OUT\_OF\_RANGE,  
     'Object Identifier' = O1,  
     'Property Identifier' = P1)  
     | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
     'Reject Reason' = INVALID\_TAG)

### 9.23.2.20 Time Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special time field values.

Test Concept: The property being tested, P1, is written with each of the special time field values to ensure that the property does not accept them. A time is selected which is within the time range that the IUT will accept for the property. The value, V1, written to the property is the time T1 with one of its fields replaced with one of the time special values. If the

property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

REPEAT SV = (hour unspecified, minute unspecified, second unspecified, hundredths unspecified) Do {

1. TRANSMIT WritePropertyMultiple-Request  
 'Object Identifier' = (O1, any object appropriate for this test),  
 'Property Identifier' = P1,  
 'Property Value' = (V1 updated with the special value SV)
2. RECEIVE  
 WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = VALUE\_OUT\_OF\_RANGE,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_TAG)

}

#### 9.23.2.21 DateTime Non-Pattern Properties Test using WritePropertyMultiple Service

Purpose: To verify that the property being tested does not accept special date field values.

Test Concept: The property being tested, P1, is written with each of the special datetime field values to ensure that the property does not accept them. A datetime DT1 is selected which is within the range that the IUT will accept for the property. The value, V1, written to the property is the datetime DT1 with one of its fields replaced with one of the date or time special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified, hundredths unspecified) DO {

1. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = (O1, any object appropriate for this test),  
 'Property Identifier' = P1,  
 'Property Value' = (DT1 updated with the special value SV)
2. RECEIVE  
 WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = VALUE\_OUT\_OF\_RANGE,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1)  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_TAG)

}

**9.23.2.22 BACnetDateRange Non-Pattern Properties Test using WritePropertyMultiple Service**

Purpose: To verify that the property being tested does not accept special date field values, except for fully unspecified start of the range or fully unspecified end of the range, or both.

Test Concept: A BACnetDateRange property, or property that is a complex datatype containing BACnetDateRange P1 is written with each of the special field values to ensure that the property does not accept them. Each half of the dateRange DR1 is selected so it is within the range that the IUT will accept for the property. The value, V1, written to the property is the dateRange DR1 with one of its fields replaced with one of the date special values. If the property is a complex datatype, the other fields in the value shall be set within the range accepted by the IUT. This test shall only be applied to devices claiming Protocol\_Revision 11 or higher.

Notes to Tester: if P1 is an array, then a non-zero array index may be provided in the TRANSMIT and the same array index observed in the WritePropertyMultiple-Error.

Test Steps:

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months, even months, last day of month, even days, odd days) DO {

1. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = (O1, any object appropriate for this test),  
 'Property Identifier' = P1,  
 'Property Value' = (DR1 with startDate updated with special value SV)
2. RECEIVE  
 WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = VALUE\_OUT\_OF\_RANGE,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_TAG)
3. TRANSMIT WritePropertyMultiple-Request,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1,  
 'Property Value' = (DR1 with endDate updated with special value SV)
4. RECEIVE  
 WritePropertyMultiple-Error,  
 'Error Class' = PROPERTY,  
 'Error Code' = VALUE\_OUT\_OF\_RANGE,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_PARAMETER\_DATATYPE)  
 | (BACnet-Reject-PDU  
 'Reject Reason' = INVALID\_TAG)

**9.24 DeviceCommunicationControl Service Execution Test****9.24.1 Positive DeviceCommunicationControl Service Execution Tests**

The purpose of this test group is to verify the correct execution of DeviceCommunicationControl service requests under circumstances where the service is expected to be successfully completed. Let X be the instance number of the Device object for the IUT.

Configuration Requirements: If the IUT requires the use of a password for DeviceCommunicationControl a valid password shall be provided and used in the test cases of this clause. If the IUT does not provide password protection the 'Password' parameter shall contain an arbitrary password or shall be omitted at the discretion of the tester. Note that passwords are to be ignored if password protection is not provided.

**9.24.1.1 Indefinite Time Duration Restored by DeviceCommunicationControl**

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when indefinite time duration is specified and communication is restored using the DeviceCommunicationControl service.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
     'Enable/Disable' = DISABLE,  
     'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. WHILE (an arbitrary time > **Internal Processing Fail Time** selected by the tester has not expired) DO {  
     TRANSMIT ReadProperty-Request,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = (any required non-array property of the Device object)  
     TRANSMIT  
         DESTINATION = LOCAL BROADCAST,  
         Who-Is-Request  
     WAIT (1 second) -- poll delay  
   }
5. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2)
6. TRANSMIT DeviceCommunicationControl-Request,  
     'Enable/Disable' = ENABLE,  
     'Password' = (any appropriate password as described in the Configuration Requirements)
7. RECEIVE BACnet-SimpleACK-PDU
8. VERIFY (Device, X), (any required non-array property) = (any valid value)

**9.24.1.2 Indefinite Time Duration Restored by ReinitializeDevice**

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when indefinite time duration is specified and communication is restored using the ReinitializeDevice service.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
     'Enable/Disable' = DISABLE,  
     'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. WHILE (an arbitrary time > **Internal Processing Fail Time** selected by the tester has not expired) DO {  
     TRANSMIT ReadProperty-Request,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = (any required non-array property of the Device object)  
     TRANSMIT  
         DESTINATION = LOCAL BROADCAST,  
         Who-Is-Request  
     WAIT (1 second) -- poll delay  
   }
5. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2)
6. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = WARMSTART,  
     'Password' = (any appropriate password as described in the Test Concept)
7. RECEIVE BACnet-SimpleACK-PDU
8. CHECK (Did the IUT perform a WARMSTART reboot?)
9. VERIFY (Device, X), (any required non-array property) = (any valid value)

**9.24.1.3 Finite Time Duration**

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when finite time duration is specified.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value  $T > 1$ , in minutes, selected by the tester),  
     'Enable/Disable' = DISABLE,  
     'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. WHILE (Time Duration  $T$  in step 1 has not expired) DO {  
     TRANSMIT ReadProperty-Request,  
         'Object Identifier' = (Device, X),  
         'Property Identifier' = (any required non-array property of the Device object)  
     TRANSMIT  
         DESTINATION = LOCAL BROADCAST,  
         Who-Is-Request  
     WAIT (1 second)                      -- poll delay  
   }
5. CHECK (Verify that the IUT did not transmit any APDUs between the acknowledgment in step 2 and expiration of timer  $T$ )
6. VERIFY (Device, X), (any required non-array property) = (any valid value)

#### 9.24.1.4 Finite Time Duration Restored by DeviceCommunicationControl

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when a finite time duration is specified and communication is restored using the DeviceCommunicationControl service.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value  $T > 1$ , in minutes, selected by the tester),  
     'Enable/Disable' = DISABLE,  
     'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (any required non-array property of the Device object)
5. WAIT (an arbitrary time  $> \text{Internal Processing Fail Time}$  selected by the tester, and  $< T$  as specified in the DeviceCommunicationControl-Request)
6. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2)
7. TRANSMIT DeviceCommunicationControl-Request,  
     'Enable/Disable' = ENABLE,  
     'Password' = (any appropriate password as described in the Configuration Requirements)
8. RECEIVE BACnet-SimpleACK-PDU
9. VERIFY ((Device, X), (any required non-array property) = (the value for this property as described in the Configuration Requirements))

#### 9.24.1.5 Finite Time Duration Restored by ReinitializeDevice

Purpose: To verify the correct execution of the DeviceCommunicationControl request service procedure when a finite time duration is specified and communication is restored using the ReinitializeDevice service.

Test Steps:

1. READ  $Y = (\text{Device}, X), \text{Object\_Name}$
2. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value  $T > 1$ , in minutes, selected by the tester)  
     'Enable/Disable' = DISABLE,  
     'Password' = (any appropriate password as described in the Test Concept)
3. RECEIVE BACnet-SimpleACK-PDU
4. WAIT **Internal Processing Fail Time**



5. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (Device, X),  
    'Property Identifier' = (any required non-array property of the Device object)
6. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester, and < T as specified in the DeviceCommunicationControl-Request)
7. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2.)
8. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialize State of Device' = WARMSTART,  
    'Password' = (any appropriate password as described in the Test Concept)
9. RECEIVE BACnet-SimpleACK-PDU
10. CHECK (Did the IUT perform a WARMSTART reboot?)
11. VERIFY (Device, X), Object\_Name = Y

#### 9.24.1.6 Indefinite Time Duration, Disable-Initiation, Restored by DeviceCommunicationControl

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when an indefinite time duration is specified, only initiation is disabled, and communication is restored using the DeviceCommunicationControl service. If the IUT does not initiate any services other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE-INITIATION,  
    'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (Device, X),  
    'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,  
    'Object Identifier' = (Device, X),  
    'Property Identifier' = (the 'Property Identifier' specified in step 7),  
    'Property Value' = (any valid value for the property)
9. TRANSMIT DeviceCommunicationControl-Request,  
    'Enable/Disable' = ENABLE,  
    'Password' = (any appropriate password as described in the Test Concept)
10. RECEIVE BACnet-SimpleACK-PDU
11. MAKE (do something to cause the IUT to initiate an APDU)
12. WAIT **Internal Processing Fail Time**
13. CHECK (Verify that the IUT initiated an APDU)

#### 9.24.1.7 Indefinite Time Duration, Disable-Initiation, Restored by ReinitializeDevice

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when an indefinite time duration is specified, only initiation is disabled, and communication is restored using the ReinitializeDevice service. If the IUT does not initiate any services other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE-INITIATION,  
    'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2.)

7. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (the 'Property Identifier' specified in step 7),  
     'Property Value' = (any valid value for the property)
9. TRANSMIT ReinitializeDevice-Request,  
     'Reinitialized State of Device' = WARMSTART,  
     'Password' = (any appropriate password as described in the Test Concept)
10. RECEIVE BACnet-SimpleACK-PDU
11. CHECK (Did the IUT perform a WARMSTART reboot?)
12. MAKE (do something to cause the IUT to initiate an APDU)
13. WAIT **Internal Processing Fail Time**
14. CHECK (Verify that the IUT initiated an APDU)

#### 9.24.1.8 Finite Time Duration, Disable Initiation

Purpose: This test case verifies the correct execution of the DeviceCommunicationControl request service procedure when finite time duration is specified and only initiation is disabled. If the IUT does not initiate any services, other than an I-Am in response to a Who-Is, then this test case shall be skipped.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value  $T > 1$ , in minutes, selected by the tester).  
     'Enable/Disable' = DISABLE-INITIATION,  
     'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time  $>$  **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any APDUs since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (any required non-array property of the Device object)
8. RECEIVE BACnet-ComplexACK-PDU,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = (the 'Property Identifier' specified in step 7),  
     'Property Value' = (any valid value for the property)
9. WAIT (T)
10. MAKE (do something to cause the IUT to initiate an APDU)
11. WAIT **Internal Processing Fail Time**
12. CHECK (Verify that the IUT initiated an APDU)

#### 9.24.1.9 Disable of Service Initiation Restored by Time Duration

Purpose: To verify the correct execution of the DeviceCommunicationControl service when DISABLE\_INITIATION is requested with a finite time duration. Communication is restored when the DeviceCommunicationControl 'Time Duration' parameter expires.

Configuration Requirements: The IUT shall be configured to initiate client requests.

Test Steps:

1. MAKE (a condition that would normally cause the IUT to initiate requests)
2. CHECK (that the IUT is initiating requests)
3. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value  $T > 1$ , in minutes, selected by the tester),  
     'Enable/Disable' = DISABLE\_INITIATION,  
     'Password' = (any appropriate password if required)
4. RECEIVE BACnet-SimpleACK-PDU

5. MAKE (a condition that would normally cause the IUT to initiate requests)
6. CHECK (that the IUT has stopped initiating requests)
7. VERIFY (any supported property) = (any valid value)
8. TRANSMIT Who-Is-Request
9. RECEIVE I-Am-Request
10. WAIT (time T)
11. MAKE (a condition that would normally cause the IUT to initiate requests)
12. CHECK (that the IUT is initiating requests)

Notes to Tester: Steps 2 through 9 must be executed within time T.

#### 9.24.1.10 Disable of Service Initiation Restored by DeviceCommunicationControl

Purpose: To verify the correct execution of the DeviceCommunicationControl service when DISABLE\_INITIATION is requested for a finite time duration. Communication is restored using the DeviceCommunicationControl service.

Configuration Requirements: The IUT shall be configured to initiate client requests.

Test Steps:

1. MAKE (a condition that would normally cause the IUT to initiate requests)
2. CHECK (that the IUT is initiating requests)
3. TRANSMIT DeviceCommunicationControl-Request,  
     'Time Duration' = (a value in minutes > time required to execute all test steps),  
     'Enable/Disable' = DISABLE\_INITIATION,  
     'Password' = (any appropriate password if required)
4. RECEIVE BACnet-SimpleACK-PDU
5. MAKE (a condition that would normally cause IUT to initiate requests)
6. CHECK (that the IUT has stopped initiating requests)
7. VERIFY (any supported property) = (any valid value)
8. TRANSMIT Who-Is-Request
9. RECEIVE I-Am-Request
10. TRANSMIT DeviceCommunicationControl-Request,  
     'Enable/Disable' = ENABLE,  
     'Password' = (any appropriate password if required)
11. MAKE (a condition that would normally cause the IUT to initiate requests)
12. CHECK (that the IUT is initiating requests)

#### 9.24.1.11 Ensure that DISABLE Option is not Supported by IUT Claiming PR >= 20

Purpose: To verify that IUT claiming Protocol Revision (PR) greater than or equal to 20, does not accept 'Enable/Disable' parameter equal to DISABLE in the DeviceCommunicationControl request.

Test Concept: Send DeviceCommunicationControl request with 'Enable/Disable' parameter equal to DISABLE to IUT. Then IUT is verified that it correctly responds with a Result(-).

Configuration Requirements: If the IUT does not support an internal clock this test shall be tested with indefinite time duration.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request  
     'Time Duration' = (a value T>1, in minutes) | (no value)  
     'Enable/Disable' = DISABLE,  
     'Password' = Any appropriate password if required
2. RECEIVE BACnet-Error-PDU  
     Error Class = SERVICES,  
     Error Code = SERVICE\_REQUEST\_DENIED
3. VERIFY (Device, X), System\_Status = (any valid value)

**9.24.1.12 Disable of Service Initiation Restored by ReinitializeDevice**

Purpose: To verify the correct execution of the DeviceCommunicationControl service when DISABLE\_INITIATION is requested with a finite time duration. Communication is restored using the ReinitializeDevice service.

Configuration Requirements: The IUT shall be configured to initiate client requests.

Test Steps:

1. MAKE (a condition that would normally cause the IUT to initiate requests)
2. CHECK (that the IUT is initiating requests)
3. TRANSMIT DeviceCommunicationControl-Request,  
    'Time Duration' = (a value in minutes > time required to execute all test steps),  
    'Enable/Disable' = DISABLE\_INITIATION,  
    'Password' = (any appropriate password if required)
4. RECEIVE BACnet-SimpleACK-PDU
5. MAKE (a condition that would normally cause IUT to initiate requests)
6. CHECK (that the IUT has stopped initiating requests)
7. VERIFY (any supported property) = (any valid value)
8. TRANSMIT Who-Is-Request
9. RECEIVE I-Am-Request
10. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = WARMSTART,  
    'Password' = (any appropriate password)
11. RECEIVE BACnet-SimpleACK-PDU
12. CHECK (Did the IUT perform a WARMSTART reboot?)
13. MAKE (a condition that would normally cause the IUT to initiate requests)
14. CHECK (that the IUT is initiating requests)

**9.24.2 Negative DeviceCommunicationControl Service Execution Tests**

The purpose of this test group is to verify the correct execution of DeviceCommunicationControl service requests under circumstances where the service is expected to fail.

**9.24.2.1 Invalid Password**

Purpose: To verify the correct execution of DeviceCommunicationControl service procedure when an invalid password is provided. If the IUT does not provide password protection this test case shall be omitted.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE\_INITIATION,  
    'Password' = (any invalid password)
2. RECEIVE BACnet-Error-PDU,  
    Error Class = SECURITY,  
    Error Code = PASSWORD\_FAILURE
3. VERIFY (Device, X), System\_Status = (any valid value)

**9.24.2.2 Missing Password**

Purpose: To verify the correct execution of DeviceCommunicationControl service procedure when a password is required but not provided. If the IUT does not provide password protection, then this test case shall be omitted.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
    'Enable/Disable' = DISABLE\_INITIATION,
2. IF (Protocol\_Revision >= 7) THEN  
    RECEIVE BACnet-Error-PDU,  
    Error Class = SECURITY,  
    Error Code = PASSWORD\_FAILURE  
ELSE  
    RECEIVE BACnet-Error-PDU,

Error Class = SECURITY,  
 Error Code = PASSWORD\_FAILURE  
 | (RECEIVE BACnet-Error-PDU,  
 Error Class = SERVICES,  
 Error Code = MISSING\_REQUIRED\_PARAMETER)

3. VERIFY (Device, X), System\_Status = (any valid value)

#### 9.2.4.2.3 Restore by ReinitializeDevice with Invalid 'Reinitialized State of Device'

Purpose: To verify that the communications are not restored when a ReinitializeDevice request is received containing one of the backup or restore related values for service parameter 'Reinitialized State of Device'.

Test Concept: Disable the IUT's communications for a period time, T, which shall be longer than it will take to complete the test. Verify that, while communications are disabled, the IUT correctly responds with a Result(-) when it receives a ReinitializeDevice request containing a backup or restore related values.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,  
 'Enable/Disable' = DISABLE  
 'Password' = (any appropriate password),  
 'Time Duration' = (a value T >= 1, in minutes) | (no value)
2. RECEIVE BACnet-SimpleACK-PDU
3. WAIT **Internal Processing Fail Time**
4. TRANSMIT ReinitializeDevice-Request,  
 'Reinitialized State of Device' = STARTBACKUP | ENDBACKUP | STARTRESTORE  
 | ENDRESTORE | ABORTRESTORE,  
 'Password' = (any appropriate password)
5. IF (Protocol\_Revision is present AND Protocol\_Revision >= 7) THEN  
 IF (Device supports DM-BR-B) THEN  
 RECEIVE BACnet-Error-PDU,  
 Error Class = SERVICES,  
 Error Code = COMMUNICATION\_DISABLED  
 ELSE  
 RECEIVE BACnet-Error-PDU,  
 'Error Class' = SERVICES,  
 'Error Code' = COMMUNICATION\_DISABLED  
 | OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED  
 ELSE  
 CHECK(that the IUT responded with BACnet-Error-PDU with an Error Class of SERVICES and any appropriate  
 Error Code or that the IUT did not respond at all)
6. TRANSMIT DeviceCommunicationControl-Request,  
 'Enable/Disable' = ENABLE  
 'Password' = (any appropriate password),
7. RECEIVE BACnet-SimpleACK-PDU

#### 9.25 ConfirmedPrivateTransfer Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ConfirmedPrivateTransfer service requests.

##### 9.25.1 Positive ConfirmedPrivateTransfer Service Execute Tests

##### 9.25.1.1 Correctly Executes a Supported ConfirmedPrivateTransfer Service

Purpose: To verify the ability to correctly execute a ConfirmedPrivateTransfer service request.

Test Concept: The service procedure implied by a particular private transfer service is defined by the vendor. This test simply verifies that an appropriate acknowledgment is returned and that any externally visible actions defined by the vendor are observed.

Configuration Requirements: The IUT shall be configured to execute at least one ConfirmedPrivateTransfer service. The service parameters that are to be provided in the request and a list of any externally visible actions that should be apparent to the tester shall also be provided.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the EPICS),  
     'Service Number' = (any service number provided by the vendor),  
     'Service Parameters' = (the service parameters provided for this service)
2. RECEIVE ConfirmedPrivateTransfer-ACK,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the EPICS),  
     'Service Number' = (the service number used in step 1),  
     'Result Block' = (the expected results provided by the vendor)
3. CHECK (Did the externally visible actions take place?)

### 9.25.2 Negative ConfirmedPrivateTransfer Service Execute Tests

#### 9.25.2.1 Correctly Executes a Non-Supported ConfirmedPrivateTransfer Service

Purpose: To verify that the IUT correctly responds with an error when an unsupported ConfirmedPrivateTransfer is received.

Test Concept: Send a non-supported ConfirmedPrivateTransfer request to the IUT and verify that it responds with an error or rejects the service.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (VID: any valid value not supported by the IUT),  
     'Service Number' = (SID: any valid value, that when combined with Vendor ID is not supported by the IUT),  
     'Service Parameters' = (any valid values)
2. RECEIVE  
     (ConfirmedPrivateTransfer-Error,  
         'Error Class' = any  
         'Error Code' = any  
         'Vendor ID' = (VID),  
         'Service Number' = (SID),  
         'Result Block' = (the expected results provided by the vendor)) |  
     (BACnet-Reject-PDU,  
         'Reject Reason' = any reason) |  
     (BACnet-Abort-PDU  
         'Abort Reason' = any reason)
3. CHECK (that the IUT exhibits the vendor defined results)

### 9.26 UnconfirmedPrivateTransfer Service Execution Tests

BACnet does not define a service procedure for executing the UnconfirmedPrivateTransfer service and thus no tests are needed.

### 9.27 ReinitializeDevice Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing ReinitializeDevice service requests.

#### 9.27.1 Positive ReinitializeDevice Service Execution Tests

The purpose of this test group is to verify correct execution of ReinitializeDevice service requests under circumstances where the service is expected to be successfully completed.

##### 9.27.1.1 COLDSTART with no Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a COLDSTART is attempted and no password is required.



Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (Did the IUT perform a COLDSTART reboot?)

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

#### 9.27.1.2 COLDSTART with a Correct Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a COLDSTART is attempted and a password is provided.

Test Concept: A password is provided whether or not the IUT requires password protection. If the IUT provides password protection, the 'Password' parameter shall contain a suitable password provided by the vendor. If the IUT does not provide password protection the 'Password' parameter shall contain an arbitrary password. Note that passwords are to be ignored if password protection is not provided. See BACnet 16.4.1.1.2.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,  
'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (Did the IUT perform a COLDSTART reboot?)

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic shall be used to confirm reinitialization whenever possible.

#### 9.27.1.3 WARMSTART with no Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a warmstart is attempted and no password is required.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (Did the IUT perform a WARMSTART reboot?)

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

#### 9.27.1.4 WARMSTART with a Correct Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a WARMSTART is attempted and a password is provided.

Test Concept: A password is provided whether or not the IUT requires password protection. If the IUT provides password protection, the 'Password' parameter shall contain a suitable password provided by the vendor. If the IUT does not provide password protection the 'Password' parameter shall contain an arbitrary password. Note that passwords are to be ignored if password protection is not provided. See BACnet 16.4.1.1.2.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,  
'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-SimpleACK-PDU



## 3. CHECK (Did the IUT perform a WARMSTART reboot?) )

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic shall be used to confirm reinitialization whenever possible.

**9.27.2 Negative ReinitializeDevice Service Execution Tests**

The purpose of this test group is to verify correct execution of Reinitialize service requests under circumstances where the service is expected to fail.

**9.27.2.1 Deleted Clause**

This test has been removed.

**9.27.2.2 Deleted Clause**

This test has been removed.

**9.27.2.3 COLDSTART with Missing Password**

Purpose: To verify that the correct BACnet Error PDU is returned when a COLDSTART is attempted and the password is invalid or a password is required but no password is provided.

Configuration Requirements: The IUT shall be configured to require a password for ReinitializeDevice.

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(RECEIVE BACnet-Error-PDU,  
Error Class = SERVICES,  
Error Code = MISSING\_REQUIRED\_PARAMETER)
3. CHECK (The IUT did NOT perform a COLDSTART reboot)
4. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = COLDSTART,  
'Password' = (any invalid password)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = MISSING\_REQUIRED\_PARAMETER)

6. CHECK (The IUT did NOT perform a COLDSTART reboot)

#### 9.27.2.4 WARMSTART with Missing Password

Purpose: To verify that the correct BACnet Error PDU is returned when a WARMSTART is attempted and the password is invalid or a password is required but no password is provided.

Configuration Requirements: The IUT shall be configured to require a password for ReinitializeDevice.

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,
2. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE  
ELSE  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(RECEIVE BACnet-Error-PDU,  
Error Class = SERVICES,  
Error Code = MISSING\_REQUIRED\_PARAMETER)
3. CHECK (The IUT did NOT perform a WARMSTART reboot)
4. TRANSMIT ReinitializeDevice-Request,  
'Reinitialized State of Device' = WARMSTART,  
'Password' = (any invalid password)
5. IF (Protocol\_Revision is present and Protocol\_Revision >= 7) THEN  
RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE  
ELSE  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SECURITY,  
'Error Code' = PASSWORD\_FAILURE) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = SERVICE\_REQUEST\_DENIED) |  
(RECEIVE BACnet-Error-PDU,  
'Error Class' = SERVICES,  
'Error Code' = MISSING\_REQUIRED\_PARAMETER)
6. CHECK (The IUT did NOT perform a WARMSTART reboot)

#### 9.27.2.5 Rejects Unsupported Reinitialize Types

Purpose: Verify that IUT correctly rejects unsupported 'Reinitialized State of Device' values.

Test Concept: Send each unsupported 'Reinitialized State of Device' value to the device and ensure that it correctly rejects the value.

Test Steps:

- REPEAT S = (each unsupported 'Reinitialized State of Device' value) DO {
1. TRANSMIT ReinitializeDevice

```

 'Reinitialized State of Device' = S,
 'Password' = (any valid value)
2. RECEIVE BACnet-Error-PDU
 'Error Class' = SERVICES,
 'Error Code' = VALUE_OUT_OF_RANGE |
 OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
}

```

### 9.28 ConfirmedTextMessage Service Execution Tests

The purpose of this test group is to verify the correct execution of the ConfirmedTextMessage service request. BACnet does not define what is to happen when a ConfirmedTextMessage service request is received except that an acknowledgement is to be returned. It is likely that some other externally observable action will take place but this is vendor specific. These tests verify that a correct acknowledgment is returned and any other action that is defined by the vendor.

#### 9.28.1 Text Message With No Message Class

Purpose: To verify the correct execution of the ConfirmedTextMessage service request when no 'Message Class' is provided.

Test Steps:

1. TRANSMIT ConfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

#### 9.28.2 Text Message With an Unsigned Message Class

Purpose: To verify the correct execution of the ConfirmedTextMessage service request when the Unsigned form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported Unsigned message classes.

Test Steps:

1. TRANSMIT ConfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Class' = (any Unsigned value from the list provided by the vendor),  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK (Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

#### 9.28.3 Text Message With a CharacterString Message Class

Purpose: To verify the correct execution of the ConfirmedTextMessage service request when the CharacterString form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported CharacterString message classes.

Test Steps:

1. TRANSMIT ConfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,

'Message Class' = (any CharacterString value from the list provided by the vendor),  
 'Message Priority' = NORMAL,  
 'Message' = (any CharacterString)

2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK(Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

#### 9.28.4 Text Message With Urgent Priority

Purpose: To verify the correct execution of the ConfirmedTextMessage service request when an urgent priority is used.

Test Steps:

1. TRANSMIT ConfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Class' = (any message class from the lists provided by the vendor),  
     'Message Priority' = URGENT,  
     'Message' = (any CharacterString)
2. RECEIVE BACnet-SimpleACK-PDU
3. CHECK(Did any vendor specified action for these circumstances occur?)

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

### 9.29 UnconfirmedTextMessage Service Execution Tests

#### 9.29.1 UnconfirmedTextMessage With No Message Class

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when no 'Message Class' is provided.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

#### 9.29.2 UnconfirmedTextMessage with an Unsigned Message Class

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the Unsigned form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported Unsigned message classes.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is appropriate.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Class' = (any Unsigned value from the list provided by the vendor),  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)
2. CHECK (Did any vendor specified action for these circumstances occur?)

**9.29.3 UnconfirmedTextMessage with a CharacterString Message Class**

Purpose: To verify the correct execution of the UnconfirmedTextMessage service request when the CharacterString form of the 'Message Class' is used.

Configuration Requirements: The vendor shall provide a list of supported CharacterString message classes.

Notes to Tester: The IUT shall respond with the indicated message and perform any vendor-specified action that is.

Test Steps:

1. TRANSMIT UnconfirmedTextMessage-Request,  
     'Text Message Source Device' = TD,  
     'Message Class' = (any CharacterString value from the list provided by the vendor),  
     'Message Priority' = NORMAL,  
     'Message' = (any CharacterString)
2. CHECK(Did any vendor specified action for these circumstances occur?)

**9.30 TimeSynchronization Service Execution Tests****9.30.1 Positive TimeSynchronization Service Execution Tests**

The purpose of this test group is to verify correct execution of TimeSynchronization service requests under circumstances where the service is expected to be successfully completed.

**9.30.1.1 TimeSynchronization Local Broadcast**

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast TimeSynchronization service request.

Notes to Tester: Select date and time such that either one or both of them is different from initial date and time.

Test Steps:

1. READ InitialDate = Local\_Date
2. READ InitialTime = Local\_Time
3. TRANSMIT  
     DA = LOCAL BROADCAST,  
     SA = TD,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = TimeSynchronization-Request,  
     date = NewDate; combined with NewTime is different than the  
             InitialDate/InitialTime pair  
     time = NewTime; combined with NewDate is different than the  
             InitialDate/InitialTime pair
4. VERIFY Local\_Date = NewDate
5. VERIFY Local\_Time == NewTime

**9.30.1.2 Directed to the IUT**

Purpose: To verify that the IUT resets its local time and date in response to a TimeSynchronization service request directed to the IUT's MAC address.

Notes to Tester: The passing results are identical to 9.30.1.1.

Test Steps: This test is identical to 9.30.1.1 except that the TimeSynchronization-Request shall be transmitted using the IUT's MAC address as the destination.

**9.31 UTCTimeSynchronization Service Execution Tests****9.31.1 Positive UTCTimeSynchronization Service Execution Tests**

The purpose of this test group is to verify correct execution of UTCTimeSynchronization service requests under circumstances where the service is expected to be successfully completed.

**9.31.1.1 Local Broadcast**

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast UTCTimeSynchronization service request.

Notes to Tester: Select date and time such that either one or both of them is different from initial date and time. The IUT may update the Daylight\_Savings\_Status during the execution of the UTCTimeSynchronization request.

Test Steps:

1. READ InitialDate = Local\_Date
2. READ InitialTime = Local\_Time
3. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = UTCTimeSynchronization-Request,
  - date = NewUtcDate: combined with NewUtcTime and converted to local time  
is different than the InitialDate/InitialTime pair
  - time = NewUtcTime: combined with NewUtcDate and converted to local time  
is different than the InitialDate/InitialTime pair
4. VERIFY Local\_Date = (NewUtcDate converted to local date/time using UTC\_Offset  
and Daylight\_Saving\_Status)
5. VERIFY Local\_Time ~ (NewUtcTime converted to local date/time using UTC\_Offset  
and Daylight\_Saving\_Status)

**9.31.1.2 Directed to the IUT**

Purpose: To verify that the IUT resets its local time and date in response to a UTCTimeSynchronization service request directed to the IUT's MAC address.

Test Steps: This test is identical to 9.31.1.1 except that the UTCTimeSynchronization request is used instead of TimeSynchronization-Request and the date and time conveyed represent UTC and the UTCTimeSynchronization-Request shall be transmitted using the IUT's MAC address as the destination.

Notes to Tester: The passing results are identical to 9.30.1.1.

**9.32 Who-Has Service Execution Tests**

The purpose of this test group is to verify the correct execution of the Who-Has service request.

**9.32.1 Execution of Who-Has Service Requests Originating from the Local Network**

The purpose of this test group is to verify the correct execution of the Who-Has request service procedure for messages originating from the local network.

**9.32.1.1 Object ID Version with No Device Range**

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and does not restrict device ranges.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Object Identifier' = Object1
3. BEFORE Unconfirmed Response Fail Time
  - RECEIVE
    - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    - I-Have-Request,

'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = Object1,  
 'Object Name' = V1

### 9.32.1.2 Object Name Version with no Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and does not restrict device ranges.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT  
     DA = LOCAL BROADCAST,  
     SA = TD,  
     Who-Has-Request,  
     'Object Name' = V1
3. BEFORE **Unconfirmed Response Fail Time**  
     RECEIVE  
     DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
     I-Have-Request,  
     'Device Identifier' = (the IUT's Device object),  
     'Object Identifier' = Object1,  
     'Object Name' = V1

### 9.32.1.3 Object ID Version with IUT Inside of the Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that includes the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT  
     DA = LOCAL BROADCAST,  
     SA = TD,  
     Who-Has-Request,  
     'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
     'Device Instance High Limit' = (any value H:  $H > \text{the Device object instance number of the IUT}$ ),  
     'Object Identifier' = Object1,
3. BEFORE **Unconfirmed Response Fail Time**  
     RECEIVE  
     DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
     I-Have-Request,  
     'Device Identifier' = (the IUT's Device object),  
     'Object Identifier' = Object1,  
     'Object Name' = V1

### 9.32.1.4 Object ID Version with IUT Outside of the Device Range

Purpose: To verify that the IUT ignores a local broadcast Who-Has service request that utilizes the object identifier form and specifies a device range restriction that does not include the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. TRANSMIT  
     DA = LOCAL BROADCAST,



SA = TD,  
 Who-Has-Request,  
 'Device Instance Low Limit' = (any value > 0: the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),  
 'Device Instance High Limit' = (any value > Device Instance Low Limit: the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),  
 'Object Identifier' = Object1

2. WAIT **Unconfirmed Response Fail Time**
3. CHECK (verify that the IUT does not respond)

#### 9.32.1.5 Object Name Version with IUT Inside of the Device Range

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Has service request that utilizes the object name form and specifies a device range restriction that includes the IUT.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT  
 DA = LOCAL BROADCAST,  
 SA = TD,  
 Who-Has-Request,  
 'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
 'Device Instance High Limit' = (any value H:  $H > \text{the Device object instance number of the IUT}$ ),  
 'Object Name' = V1
3. BEFORE **Unconfirmed Response Fail Time**  
 RECEIVE  
 DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 I-Have-Request,  
 'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = Object1,  
 'Object Name' = V1

#### 9.32.1.6 Object Name Version with IUT Outside of the Device Range

Purpose: To verify that the IUT ignores a local broadcast Who-Has service request that utilizes the object name form and specifies a device range restriction that does not include the IUT.

Test Steps:

1. TRANSMIT  
 DA = LOCAL BROADCAST,  
 SA = TD,  
 Who-Has-Request,  
 'Device Instance Low Limit' = (any value > 0 such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),  
 'Device Instance High Limit' = (any value > Device Instance Low Limit such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),  
 'Object Name' = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

#### 9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object identifier form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),
  - 'Device Instance High Limit' = (The Device object instance number of the IUT),
  - 'Object Identifier' = Object1
3. BEFORE **Unconfirmed Response Fail Time**
  - RECEIVE
    - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    - I-Have-Request,
    - 'Device Identifier' = (the IUT's Device object),
    - 'Object Identifier' = Object1,
    - 'Object Name' = V1

#### 9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object identifier form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (The Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H > \text{the Device object instance number of the IUT}$ ),
  - 'Object Identifier' = Object1
3. BEFORE **Unconfirmed Response Fail Time**
  - RECEIVE
    - DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,
    - I-Have-Request,
    - 'Device Identifier' = (the IUT's Device object),
    - 'Object Identifier' = Object1,
    - 'Object Name' = V1

#### 9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object name form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),
  - 'Device Instance High Limit' = (The Device object instance number of the IUT),
  - 'Object Name' = V1

**3. BEFORE Unconfirmed Response Fail Time**

RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,

I-Have-Request,

'Device Identifier' = (the IUT's Device object),

'Object Identifier' = Object1,

'Object Name' = V1

**9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range**

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range for Who-Has service requests that utilize the object name form.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name

2. TRANSMIT

DA = LOCAL BROADCAST,

SA = TD,

Who-Has-Request,

'Device Instance Low Limit' = (The Device object instance number of the IUT),

'Device Instance High Limit' = (any value H: H &gt; the Device object instance number of the IUT),

'Object Name' = V1

**3. BEFORE Unconfirmed Response Fail Time**

RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,

I-Have-Request,

'Device Identifier' = (the IUT's Device object),

'Object Identifier' = Object1,

'Object Name' = V1

**9.32.1.11 Object Name Version, Directed to a Specific MAC Address**

Purpose: To verify that the IUT responds with a broadcast I-Have service request even if the Who-Has service requests was not transmitted with a broadcast address.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name

2. TRANSMIT Who-Has-Request,

'Object Name' = V1,

**3. BEFORE Unconfirmed Response Fail Time**

RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,

I-Have-Request,

'Device Identifier' = (the IUT's Device object),

'Object Identifier' = Object1,

'Object Name' = V1

**9.32.1.12 Who-Has After Object\_Name Changed**

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Name property of an object in the device is changed.

Test Concept: The Object\_Name property of the referenced object is read to determine its initial value. The Object\_Name property is then changed to a different value, V2, which is not already used by an object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Name' parameter, using the values V1 and V2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Name property and has the value V1. If IUT does not support objects with modifiable Object\_Name properties, then this test shall be skipped.

Test Steps:

1. READ V1 = O1, Object\_Name
2. IF (Object\_Name is writable) THEN  
    WRITE O1, Object\_Name = V2  
ELSE  
    MAKE (O1, Object\_Name = V2)
3. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Name' = V1
4. WAIT **Unconfirmed Response Fail Time**
5. CHECK (Verify that the IUT does not respond with an I-Have request)
6. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Name' = V2
7. BEFORE **Unconfirmed Response Fail Time**  
    RECEIVE  
    DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
    I-Have-Request,  
    'Device Identifier' = (the IUT's Device object),  
    'Object Identifier' = O1,  
    'Object Name' = V2

#### 9.32.1.13 Who-Has After Object\_Identifier Changed

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Identifier property of an object in the device is changed.

Test Concept: The Object\_Identifier property of the referenced object, O1, is verified to contain the value O1. The Object\_Identifier property is then changed to a different value, O2, which is not already in use by a different object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Identifier' parameter, using the values O1 and O2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Identifier property. If the IUT does not support objects with modifiable Object\_Identifiers, then this test shall be skipped.

Test Steps:

1. VERIFY O1, Object\_Identifier = O1
2. IF (O1 is writable) THEN  
    WRITE O1, Object\_Identifier = O2  
ELSE  
    MAKE (O1, Object\_Identifier = O2)
3. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Identifier' = O1
4. WAIT **Unconfirmed Response Fail Time**
5. CHECK (Verify that the IUT does not respond with an I-Have request)
6. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Identifier' = O2
7. BEFORE **Unconfirmed Response Fail Time**  
    RECEIVE  
    DA = LOCAL BROADCAST | GLOBAL BROADCAST | TD,

I-Have-Request,  
 'Device Identifier' = (the IUT's Device object),  
 'Object Identifier' = O2  
 'Object Name' = V1

### 9.32.2 Execution of Who-Has Service Requests Originating from a Remote Network

The purpose of this test group is to verify the correct execution of the Who-Has request service procedure for messages originating from a remote network. A comprehensive set of variations in Who-Has request parameters is not included in this test group because they are tested in 9.32.1. The tests in this group only represent variations in network layer addressing information.

#### 9.32.2.1 Object ID Version, Global Broadcast from a Remote Network

Purpose: To verify the ability of the IUT to recognize the origin of a globally broadcast Who-Has service request and to respond such that the device originating the request receives the response.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - DNET = GLOBAL BROADCAST,
  - SNET = (X, any remote network number),
  - SADR = (Y, any MAC address valid for the specified network),
  - Who-Has-Request,
  - 'Object Identifier' = Object1
3. BEFORE **Unconfirmed Response Fail Time**
  - RECEIVE
  - DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to network X)
  - | TD (DNET = X, DADR = Y),
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = Object1,
  - 'Object Name' = V1

#### 9.32.2.2 Object ID Version, Remote Broadcast

Purpose: To verify the ability of the IUT to recognize the origin of a remotely broadcast Who-Has service request and to respond such that the device originating the request receives the response.

Configuration Requirements: Choose any object (Object1) that exists within the IUT.

Test Steps:

1. READ V1 = (Object1), Object\_Name
2. TRANSMIT
  - SNET = (any remote network number),
  - SADR = (any MAC address valid for the specified network),
  - Who-Has-Request,
  - 'Object Identifier' = Object1
3. BEFORE **Unconfirmed Response Fail Time**
  - RECEIVE
  - DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to network X)
  - | TD (DNET = X, DADR = Y),
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = Object1,
  - 'Object Name' = V1

**9.32.2.3 Who-Has for Non-existent Object\_Name**

Purpose: Verifies correct responses to Who-Has service requests by 'Object Name' when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has service request with 'Object Name' when that named object does not exist in the IUT.

Configuration Requirements: Choose any character string value V1, which is not the Object\_Name of any object in the IUT. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

Test Steps:

1. TRANSMIT Who-Has-Request,  
'Object Name' = V1
2. WAIT **Unconfirmed Response Fail Time**
3. CHECK (the IUT does not respond with an I-Have request with 'Object Name' containing V1)

**9.32.2.4 Who-Has for Non-existent Object\_Identifier**

Purpose: Verifies correct responses to Who-Has service requests when the object does not exist in the IUT.

Test Concept: The test verifies the correct non-response to Who-Has request with that 'Object Identifier' parameter for an object which does not exist.

Configuration Requirements: Choose any standard object (Object1) that does not exist within the IUT, i.e., any unsupported Object Type or any supported Object Type for which the instance does not exist. The IUT shall be placed in a state where it is not producing I-Have spontaneously.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = Object\_Identifier
2. RECEIVE BACnet-Error-PDU,  
'Error Class' = OBJECT,  
'Error Code' = UNKNOWN\_OBJECT
3. TRANSMIT Who-Has-Request,  
'Object Identifier' = Object1
4. WAIT **Unconfirmed Response Fail Time**
5. CHECK (the IUT does not respond with an I-Have request with 'Object Identifier' containing Object1)

**9.33 Who-Is Service Execution Tests**

The purpose of this test group is to verify the correct execution of the Who-Is service request.

**9.33.1 Execution of Who-Is Service Requests Originating from the Local Network**

The purpose of this test group is to verify the correct execution of the Who-Is request service procedure for messages originating from the local network.

**9.33.1.1 Local Broadcast, General Inquiry**

Purpose: To verify that the IUT can correctly respond to a local broadcast Who-Is service request that does not restrict device ranges.

Test Steps:

1. TRANSMIT  
DESTINATION = LOCAL BROADCAST,  
Who-Is-Request
2. BEFORE **Unconfirmed Response Fail Time**  
RECEIVE  
DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
I-Am-Request,

|                              |                                             |
|------------------------------|---------------------------------------------|
| 'I Am Device Identifier' =   | (the IUT's Device object),                  |
| 'Max APDU Length Accepted' = | (the value specified in the EPICS),         |
| 'Segmentation Supported' =   | (the value specified in the EPICS),         |
| 'Vendor Identifier' =        | (the identifier registered for this vendor) |

### 9.33.1.2 Global Broadcast, General Inquiry

Purpose: To verify that the IUT can correctly respond to a global broadcast Who-Is request that does not restrict device ranges.

Test Steps:

1. TRANSMIT  
DESTINATION = GLOBAL BROADCAST,  
Who-Is-Request
2. BEFORE **Unconfirmed Response Fail Time**  
RECEIVE  
DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
I-Am-Request,  
'I Am Device Identifier' = (the IUT's Device object),  
'Max APDU Length Accepted' = (the value specified in the EPICS),  
'Segmentation Supported' = (the value specified in the EPICS),  
'Vendor Identifier' = (the identifier registered for this vendor)

### 9.33.1.3 Local Broadcast, Specific Device Inquiry with IUT Outside of the Device Range

Purpose: To verify that the IUT ignores Who-Is requests when it is excluded from the specified device range.

Test Steps:

1. TRANSMIT  
DESTINATION = LOCAL BROADCAST,  
Who-Is-Request,  
'Device Instance Range Low Limit' = (any value  $\geq 0$  such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),  
'Device Instance Range High Limit' = (any value  $\geq$  Device Instance Low Limit such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

### 9.33.1.4 Local Broadcast, Specific Device Inquiry with IUT Device Instance Equal to Low Limit of Device Range

Purpose: To verify that the IUT correctly recognizes the low limit of the specified device range.

Test Steps:

1. TRANSMIT  
DESTINATION = LOCAL BROADCAST,  
Who-Is-Request,  
'Device Instance Range Low Limit' = (The Device object instance number of the IUT),  
'Device Instance Range High Limit' = (any value H:  $H >$  the Device object instance number of the IUT)
2. BEFORE **Unconfirmed Response Fail Time**  
RECEIVE  
DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
I-Am-Request,  
'I Am Device Identifier' = (the IUT's Device object),  
'Max APDU Length Accepted' = (the value specified in the EPICS),  
'Segmentation Supported' = (the value specified in the EPICS),  
'Vendor Identifier' = (the identifier registered for this vendor)



**9.33.1.5 Local Broadcast, Specific Device Inquiry with IUT Device Instance Equal to High Limit of Device Range**

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range.

Test Steps:

1. TRANSMIT  
 DESTINATION = LOCAL BROADCAST,  
 Who-Is-Request,  
 'Device Instance Range Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
 'Device Instance Range High Limit' = (the Device object instance number of the IUT)
2. BEFORE **Unconfirmed Response Fail Time**  
 RECEIVE  
 DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
 I-Am-Request,  
 'I Am Device Identifier' = (the IUT's Device object),  
 'Max APDU Length Accepted' = (the value specified in the EPICS),  
 'Segmentation Supported' = (the value specified in the EPICS),  
 'Vendor Identifier' = (the identifier registered for this vendor)

**9.33.1.6 Local Broadcast, Specific Device Inquiry with IUT Inside of the Device Range**

Purpose: To verify that the IUT responds to Who-Is requests when it is included within the specified device range.

Test Steps:

1. TRANSMIT  
 DESTINATION = LOCAL BROADCAST,  
 Who-Is-Request,  
 'Device Instance Range Low Limit' = (any value L:  $0 \leq L < \text{the Device object instance number of the IUT}$ ),  
 'Device Instance Range High Limit' = (any value H:  $H > \text{the Device object instance number of the IUT}$ )
2. BEFORE **Unconfirmed Response Fail Time**  
 RECEIVE  
 DESTINATION = GLOBAL BROADCAST | LOCAL BROADCAST | TD  
 I-Am-Request,  
 'I Am Device Identifier' = (the IUT's Device object),  
 'Max APDU Length Accepted' = (the value specified in the EPICS),  
 'Segmentation Supported' = (the value specified in the EPICS),  
 'Vendor Identifier' = (the identifier registered for this vendor)

**9.33.2 Execution of Who-Is Service Requests Originating from a Remote Network**

The purpose of this test group is to verify the correct execution of the Who-Is request service procedure for messages originating from a remote network. A comprehensive set of variations in Who-Is request parameters is not included in this test group because they are tested in 9.33.1. The tests in this group only represent variations in network layer addressing information.

**9.33.2.1 General Inquiry, Global Broadcast from a Remote Network**

Purpose: To verify the ability of the IUT to recognize the origin of a globally broadcast Who-Is service request and respond such that the device originating the request receives the response.

Test Steps:

1. TRANSMIT  
 DESTINATION = GLOBAL BROADCAST,  
 SNET = (any remote network number),  
 SADR = (any MAC address valid for the specified network),  
 Who-Is-Request
2. BEFORE **Unconfirmed Response Fail Time**  
 RECEIVE

DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified by SNET in step 1) | TD,

I-Am-Request,  
 'I Am Device Identifier' = (the IUT's Device object),  
 'Max APDU Length Accepted' = (the value specified in the EPICS),  
 'Segmentation Supported' = (the value specified in the EPICS),  
 'Vendor Identifier' = (the identifier registered for this vendor)

### 9.33.2.2 General Inquiry, Remote Broadcast

Purpose: To verify the ability of the IUT to recognize the origin of a remotely broadcast Who-Is service request and respond such that the device originating the request receives the response.

Test Steps:

#### 1. TRANSMIT

DESTINATION = LOCAL BROADCAST,  
 SNET = (any remote network number),  
 SADR = (any MAC address valid for the specified network),  
 Who-Is-Request

#### 2. BEFORE Unconfirmed Response Fail Time

RECEIVE

DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified by SNET in step 1) | TD,

I-Am-Request,  
 'I Am Device Identifier' = (the IUT's Device object),  
 'Max APDU Length Accepted' = (the value specified in the EPICS),  
 'Segmentation Supported' = (the value specified in the EPICS),  
 'Vendor Identifier' = (the identifier registered for this vendor)

### 9.33.2.3 General Inquiry, Directed to a Remote Device

Purpose: To verify the ability of the IUT to recognize the origin of a Who-Is service request, directed to the IUT, and respond such that the device originating the request receives the response.

Test Steps:

#### 1. TRANSMIT

DESTINATION = IUT,  
 SNET = (any remote network number),  
 SADR = (any MAC address valid for the specified network),  
 Who-Is-Request

#### 2. BEFORE Unconfirmed Response Fail Time

RECEIVE

DESTINATION = GLOBAL BROADCAST  
 | REMOTE BROADCAST (to the network specified by SNET in step 1)  
 | TD

I-Am-Request,  
 'I Am Device Identifier' = (the IUT's Device object),  
 'Max APDU Length Accepted' = (the value specified in the EPICS),  
 'Segmentation Supported' = (the value specified in the EPICS),  
 'Vendor Identifier' = (the identifier registered for this vendor)

## 9.34 VT-Open Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing VT-Open service requests.

Test Concept: An attempt is made to open a VT-session for each terminal class supported by the IUT. Confirmation that the session is open consists of reading the Active\_VT\_Sessions property of the Device object. Exchange of VT data is done as part of the VT-Data service execution tests in 9.36. The VT-sessions are left open unless the IUT is unable to support multiple open sessions. This creates open sessions that can be used to test the VT-Close service execution in 9.35.

**9.34.1 Default Terminal VT-class**

Purpose: To verify that the IUT responds to a VT-Open request to establish a session using the default terminal class and that the VT-session is reflected in the Active\_VT\_Sessions property of the IUT's Device object.

Test Steps:

1. TRANSMIT VT-Open-Request,  
     'VT-class' = DEFAULT\_TERMINAL,  
     'Local VT Session Identifier' = (any valid unique session identifier)
2. RECEIVE VT-Open-ACK,  
     'Remote VT Session Identifier' = (any valid unique session identifier)
3. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = (the IUT's Device object),  
     'Property Identifier' = Active\_VT\_Sessions
4. RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (the IUT's Device object),  
     'Property Identifier' = Active\_VT\_Sessions,  
     'Property Value' = (any list of sessions that contains the session ID pair from steps 1 and 2)
5. IF (the IUT can have only one open VT-session) THEN {  
     TRANSMIT VT-Close-Request,  
     'List of Remote VT Session Identifiers' = (the remote session identifier from step 2)  
     RECEIVE BACnet-SimpleACK-PDU  
   }

Notes to Tester: Successfully completing steps 1 through 4 is a passing result. If step 5 fails, this indicates a failure of the VT-Close service.

**9.34.2 Other VT-classes**

Purpose: To verify that the IUT responds to VT-Open requests for all supported optional VT-classes and that the VT-sessions are reflected in the Active\_VT\_Sessions property of the IUT's Device object. If DEFAULT\_TERMINAL is the only VT-class supported this test shall be omitted.

Test Steps:

1. REPEAT X = (the supported optional VT-classes) DO {  
     TRANSMIT VT-Open-Request,  
         'VT-class' = X,  
         'Local VT Session Identifier' = (any valid unique session identifier)  
     RECEIVE VT-Open-ACK,  
         'Remote VT Session Identifier' = (any valid unique session identifier)  
     TRANSMIT ReadProperty-Request,  
         'Object Identifier' = (the IUT's Device object),  
         'Property Identifier' = Active\_VT\_Sessions  
     Receive ReadProperty-ACK,  
         'Object Identifier' = (the IUT's Device object),  
         'Property Identifier' = Active\_VT\_Sessions,  
         'Property Value' = (any list of sessions that contains the session ID pair created above)  
  
     IF (the maximum number of open VT-sessions has been reached) THEN {  
         TRANSMIT VT-Close-Request,  
             'List of Remote VT Session Identifiers' = (any active remote session identifier)  
  
         RECEIVE BACnet-SimpleACK-PDU  
     }  
   }

**9.35 VT-Close Service Execution Tests**

This clause defines the tests necessary to demonstrate support for executing VT-Close service requests.

**9.35.1 Closing One of Multiple Open VT Sessions**

Purpose: To verify that the IUT responds to a VT-Close request to terminate a single VT-session when multiple sessions are open. If the IUT does not support multiple open VT-sessions this test shall be omitted.

Configuration Requirements: The IUT shall be configured with more than one open VT-session. The VT-Open execution tests in 9.34 may be used to accomplish this.

Test Steps:

1. TRANSMIT VT-Close-Request,  
'List of Remote VT Session Identifiers' = (any single identifier for an open session identifier)
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions
4. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions,  
'Property Value' = (any list of sessions that does not contain the closed session)

**9.35.2 Closing Multiple Open VT Sessions**

Purpose: To verify that the IUT responds to a VT-Close request to terminate multiple open VT-sessions. If the IUT does not support multiple open VT-sessions this test shall be omitted.

Configuration Requirements: The IUT shall be configured with more than one open VT-session. The VT-Open execution tests in 9.34 may be used to accomplish this.

Test Steps:

1. TRANSMIT VT-Close-Request,  
'List of Remote VT Session Identifiers' = (at least two session identifiers corresponding to open sessions)
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions
4. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions,  
'Property Value' = (any list of sessions that does not contain the closed sessions)

**9.35.3 Closing a Single Open VT Session**

Purpose: To verify that the IUT responds to a VT-Close request to terminate a single open VT-session.

Configuration Requirements: The IUT shall be configured one open VT-session. The VT-Open execution tests in 9.34 may be used to accomplish this.

Test Steps:

1. TRANSMIT VT-Close-Request,  
'List of Remote VT Session Identifiers' = (the session identifier corresponding to the open session)
2. RECEIVE BACnet-SimpleACK-PDU
3. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions
4. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the IUT's Device object),  
'Property Identifier' = Active\_VT\_Sessions,  
'Property Value' = (an empty list)

### 9.36 VT-Data Service Execution Tests

The exchange of VT data using the VT-Data service requires both initiating and executing the service. Passing the tests in 8.38 is sufficient to demonstrate execution of the VT-Data service.

### 9.37 RequestKey Service Execution Test

**Purpose:** To verify the ability of a key server to correctly execute a RequestKey service request. If the IUT is not a key server this test shall be omitted.

**Test Concept:** The RequestKey service procedure prescribes a sequence of steps used to establish cryptographic session keys. This test is based on the existence of two devices, a client device A and a server device B. Client A wishes to send a secure service request to server B. Before this can be done a session key must be established. The test consists of a sequence of messages that must be exchanged if the key server (IUT) correctly executes the RequestKey service.

The IUT plays the role of the key server in this transaction. The TD plays the role of both device A and device B. This requires either the use of two separate TDs, each with a private key known by the key server or else one TD that has two separate keys and two unique Device object identifiers so that it can appear to be two devices from the perspective of the key server.

As a practical matter the IUT is required to know the maximum APDU length accepted by devices A and B, in order that its enciphered APDUs may be padded as required by BACnet Clause 24.1.4. How the IUT acquires this information is a local matter; it may acquire the information via BACnet services during the performance of this test. The PDUs that would be exchanged to accomplish this are not specified in the test description.

All PDUs in this test that are specified to be enciphered are first padded per BACnet Clause 24.1.4 and then enciphered per BACnet Clause 24.4 with the specified key.

**Configuration Requirements:** The IUT shall be configured with private 56-bit cryptographic keys ( $PK_A$  and  $PK_B$ ), Device object identifiers, and MAC addresses that correspond to device A and device B.

**Test Steps:**

1. TRANSMIT RequestKey-Request,

|                                  |                                              |
|----------------------------------|----------------------------------------------|
| DESTINATION =                    | IUT,                                         |
| SOURCE =                         | (device A),                                  |
| 'Requesting Device Identifier' = | (the Device object identifier for device A), |
| 'Requesting Device Address' =    | (a BACnetAddress for device A),              |
| 'Remote Device Identifier' =     | (the Device object identifier for device B), |
| 'Remote Device Address' =        | (a BACnetAddress for device B)               |

Note: The service request portion of this PDU shall be enciphered using  $PK_A$ .

2. BEFORE **Internal Processing Fail Time**

RECEIVE

|                          |                                   |
|--------------------------|-----------------------------------|
| DESTINATION =            | (device A),                       |
| SOURCE =                 | IUT,                              |
| Authenticate-Request,    |                                   |
| 'Pseudo Random Number' = | (any valid pseudo random number), |
| 'Expected Invoke ID' =   | (the invoke ID used in step 1)    |

Note: The service request portion of this PDU shall be enciphered using  $PK_A$ .

3. TRANSMIT

|                            |                                     |
|----------------------------|-------------------------------------|
| DESTINATION =              | IUT,                                |
| SOURCE =                   | (device A),                         |
| Authenticate-Request-ACK,  |                                     |
| 'Modified Random Number' = | (the modified pseudo random number) |

Note: The service request portion of this PDU shall be enciphered using  $PK_A$ . At this point the request in step 1 has been authenticated and the RequestKey service procedure continues.

4. BEFORE **Internal Processing Fail Time**

RECEIVE

|                         |                                              |
|-------------------------|----------------------------------------------|
| DESTINATION =           | (device B),                                  |
| SOURCE =                | IUT,                                         |
| AddListElement-Request, |                                              |
| 'Object Identifier' =   | (the Device object identifier for device B), |

'Property Identifier' = List\_Of\_Session\_Keys,  
'List of Elements' = (SK<sub>AB</sub>, BACnetAddress for device A)

Note: The List\_Of\_Session\_Keys property value shall be enciphered using PK<sub>B</sub>.

5. TRANSMIT

DESTINATION = IUT,  
SOURCE = (device B),  
Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number),  
'Expected Invoke ID' = (the invoke ID used in step 6)

Note: The service request portion of this PDU shall be enciphered using PK<sub>B</sub>.

6. BEFORE **Acknowledgment Fail Time**

RECEIVE

DESTINATION = (device B),  
SOURCE = IUT,  
Authenticate-Request-ACK,  
'Modified Random Number' = (the modified pseudo random number)

Note: The service request portion of this PDU shall be enciphered using PK<sub>B</sub>. At this point the request in step 4 has been authenticated.

7. TRANSMIT

DESTINATION = IUT,  
SOURCE = (device B),  
BACnet-SimpleACK-PDU

Note: This is the acknowledgment of a successful execution of the AddListElement request in step 4.

8. BEFORE **Internal Processing Fail Time**

RECEIVE

DESTINATION = (device A),  
SOURCE = IUT,  
AddListElement-Request,  
'Object Identifier' = (the Device object identifier for device A),  
'Property Identifier' = List\_Of\_Session\_Keys,  
'List of Elements' = (SK<sub>AB</sub>, BACnetAddress for device B)

Note: The List\_Of\_Session\_Keys property value shall be enciphered using PK<sub>A</sub>.

9. TRANSMIT

DESTINATION = IUT,  
SOURCE = (device A),  
Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number),  
'Expected Invoke ID' = (the invoke ID used in step 8)

Note: The service request portion of this PDU shall be enciphered using PK<sub>A</sub>.

10. BEFORE **Acknowledgment Fail Time**

RECEIVE

DESTINATION = (device A),  
SOURCE = IUT,  
Authenticate-Request-ACK,  
'Modified Random Number' = (the modified pseudo random number)

Note: The service request portion of this PDU shall be enciphered using PK<sub>A</sub>.

At this point the request in step 12 has been authenticated.

11. TRANSMIT

DESTINATION = IUT,  
SOURCE = (device A),  
BACnet-SimpleACK-PDU

Note: This is the acknowledgment of a successful execution of the AddListElement request in step 8.

12. BEFORE **Acknowledgment Fail Time**

RECEIVE

DESTINATION = (device A),  
SOURCE = IUT,  
BACnet-SimpleACK-PDU

Note: This is the acknowledgment of a successful execution of the RequestKey service request in step 1.

**9.38 Authenticate Service Execution Tests**

This clause defines the tests necessary to demonstrate support for executing Authenticate service requests.

The tests in this clause require the IUT to know the maximum APDU length accepted by the TD, in order that its enciphered APDUs may be padded as required by BACnet Clause 24.1.4. How the IUT acquires this information is a local matter; it may acquire the information via BACnet services during the performance of this test. The PDUs that would be exchanged to accomplish this are not specified in the test description.

All PDUs in these tests that are specified to be enciphered are first padded per BACnet Clause 24.1.4 and then enciphered per BACnet Clause 24.4 with the specified key.

**9.38.1 Establishing a Session Key**

Purpose: Subsequent tests in this clause require that a secure session be established prior to test execution. This test verifies that the IUT can respond to an attempt to establish such a session. If the IUT can only establish sessions at configuration time this test shall be omitted.

Test Concept: The TD functions as a key server and attempts to deliver a session key to the IUT. The IUT verifies that the key is in fact being delivered by the TD.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key.

Test Steps:

1. TRANSMIT AddListElement-Request,  
     'Object Identifier' = (the Device object identifier of the IUT),  
     'Property Identifier' = List\_Of\_Session\_Keys,  
     'List of Elements' = (SK<sub>TD,IUT</sub>, BACnetAddress for TD)

Note: The 'List of Elements' shall be enciphered using PK<sub>IUT</sub>.

2. BEFORE **Internal Processing Fail Time**  
     RECEIVE  
         Authenticate-Request,  
         'Pseudo Random Number' = (any valid pseudo random number),  
         'Expected Invoke ID' = (the invoke ID used in step 1)

Note: The service request portion of this PDU shall be enciphered using PK<sub>IUT</sub>.

3. TRANSMIT  
     Authenticate-Request-ACK,  
     'Modified Random Number' = (the modified pseudo random number)

Note: At this point the request in step 1 has been authenticated.

4. BEFORE **Acknowledgment Fail Time**  
     RECEIVE

BACnet-SimpleACK-PDU

Note: This is the acknowledgment of a successful execution of the AddListElement request in step 1.

**9.38.2 Peer Authentication**

Purpose: To verify the ability of a device to correctly execute an Authenticate service request to implement peer authentication. If the IUT is only a key server this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key, SK<sub>TD,IUT</sub>, shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. TRANSMIT Authenticate-Request,  
     'Pseudo Random Number' = (any valid pseudo random number)
2. BEFORE **Acknowledgment Fail Time**  
     RECEIVE Authenticate-Request-ACK,  
     'Modified Random Number' = (the modified pseudo random number)



**9.38.3 Message Execution Authentication**

Purpose: To verify the ability of a device to correctly execute an Authenticate service request to implement message execution authentication. If the IUT is only a key server this test shall be omitted

Test Concept: A secure session between the TD and the IUT has been established. The TD makes a ReadProperty request using the procedures for authenticated service execution.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. TRANSMIT Authenticate-Request,  
     'Pseudo Random Number' = (any valid pseudo random number),  
     'Expected Invoke ID' = (any valid invoke ID)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
2. TRANSMIT ReadProperty-Request,  
     Invoke ID = (the 'Expected Invoke ID' used in step 1),  
     'Object Identifier' = (any object supported by the IUT),  
     'Property Identifier' = (any supported property of the specified object)
3. BEFORE **Internal Processing Delay Time**  
     RECEIVE Authenticate-Request-ACK,  
     'Modified Random Number' = (the modified pseudo random number)
- Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .
4. BEFORE **Internal Processing Delay Time**  
     RECEIVE ReadProperty-ACK,  
     'Object Identifier' = (the object identifier used in step 2),  
     'Property Identifier' = (the property identifier used in step 2),  
     'Property Value' = (the value of the property as specified in the EPICS)

**9.38.4 Message Initiation Authentication**

This clause defines the tests necessary to demonstrate support for executing the Authenticate service for the purpose of message initiation authentication.

**9.38.4.1 Message Initiation Authentication by a Key Server**

Executing message initiation authentication as a key server is covered by test 9.37

**9.38.4.2 Message Initiation Authentication Peer-to-Peer**

Purpose: To verify the ability to correctly execute an Authenticate service request to implement message initiation authentication. If the IUT is a key server only, this test shall be omitted.

Test Concept: A secure session between the TD and the IUT has been established. The IUT initiates an application service request addressed to the TD. The TD then follows the procedures for authenticating the source of the request.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1. A means shall be provided to cause the IUT to initiate a ReadProperty service request.

Test Steps:

1. MAKE (the IUT initiate a ReadProperty service request)
2. RECEIVE ReadProperty-Request,  
     'Object Identifier' = (any standard object),  
     'Property Identifier' = (any property of the specified object)
3. TRANSMIT Authenticate-Request  
     'Pseudo Random Number' = (any valid pseudo random number),  
     'Expected Invoke ID' = (the invoke ID used in step 2)

Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .

#### 4. BEFORE **Acknowledgment Fail Time** RECEIVE

Authenticate-Request-ACK,  
'Modified Random Number' = (the modified pseudo random number)

Note: The service request portion of this PDU shall be enciphered using  $SK_{TD,IUT}$ .

#### 9.38.5 Operator Authentication

This clause defines the tests necessary to demonstrate support for executing the Authenticate service for the purpose of operator authentication. If the IUT is not a key server that performs operator authentication these tests shall be omitted.

Test Concept: The TD sends an enciphered request to authenticate an operator. The IUT then follows the procedures for authenticating the operator.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key,  $PK_{TD}$ , that corresponds to the TD and a known operator name/password combination.

##### 9.38.5.1 Logon Accepted

Purpose: To verify the ability of a key server to correctly execute an Authenticate service request to implement operator authentication under circumstances where the authentication is expected to succeed.

Test Steps:

1. TRANSMIT Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number),  
'Operator Name' = (the operator name configured for this test),  
'Operator Password' = (the operator password configured for this test)

Note: The service request portion of this PDU shall be enciphered using  $PK_{TD}$ .

2. BEFORE **Acknowledgment Fail Time**  
RECEIVE Authenticate-ACK,  
'Modified Random Number' (the modified pseudo random number)

Note: The service request portion of this PDU shall be enciphered using  $PK_{TD}$ .

##### 9.38.5.2 Logon Refused

Purpose: To verify the ability of a key server to correctly execute an Authenticate service request to implement operator authentication under circumstances where the authentication is expected to fail.

Test Steps:

1. TRANSMIT Authenticate-Request,  
'Pseudo Random Number' = (any valid pseudo random number),  
'Operator Name' = (the operator name configured for this test),  
'Operator Password' = (a password other than the one configured for this test)

Note: The service request portion of this PDU shall be enciphered using  $PK_{TD}$ .

2. BEFORE **Acknowledgment Fail Time**  
RECEIVE BACnet-Error-PDU,  
Error Class = SECURITY,  
Error Code = PASSWORD\_FAILURE

#### 9.38.6 Enciphered Session

Purpose: To verify the ability of the IUT to correctly execute an Authenticate service request to initiate and terminate an enciphered session.

Test Concept: The TD attempts to initiate an enciphered session with the IUT by transmitting an Authenticate request. The IUT is expected to follow the procedure in BACnet 24.3.1 to authenticate the request and start the session. Next, the TD then reads the List\_Of\_Session\_Keys property from the Device object of the IUT. The TD then attempts to end the session by transmitting another Authenticate request. The IUT responds by implementing the procedure in BACnet 24.3.2 for encoding an enciphered session. The TD again attempts to read the List\_Of\_Session\_Keys causing an error response. Note that the service request portion of all of the messages in this test are enciphered using  $SK_{TD,IUT}$ .

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key,  $SK_{TD,IUT}$ , shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. TRANSMIT Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Start Enciphered Session' = TRUE
  2. BEFORE **Internal Processing Fail Time**  
    RECEIVE Authenticate-Request,  
        'Pseudo Random Number' = (any valid pseudo random number),  
        'Expected Invoke ID' = (the invoke ID used in step 1)
  3. TRANSMIT Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 2)
  4. BEFORE **Acknowledgment Fail Time**  
    RECEIVE Authenticate-ACK,  
        'Modified Random Number' = (the modified random number from step 1)
- Note: At this point the enciphered session is initiated.
5. VERIFY (the IUT's Device object), List\_Of\_Session\_Keys = (a list containing the session key for this session)
  6. TRANSMIT Authenticate-Request,  
    'Pseudo Random Number' = (any valid pseudo random number),  
    'Start Enciphered Session' = FALSE
  7. BEFORE **Internal Processing Fail Time**  
    RECEIVE Authenticate-Request,  
        'Pseudo Random Number' = (any valid pseudo random number),  
        'Expected Invoke ID' = (the invoke ID used in step 6)
  8. TRANSMIT Authenticate-ACK,  
    'Modified Random Number' = (the modified random number from step 7)
  9. BEFORE **Acknowledgment Fail Time**  
    RECEIVE Authenticate-ACK,  
        'Modified Random Number' = (the modified random number from step 6)
  10. TRANSMIT ReadProperty-Request,  
    'Object Identifier' = (the Device object of the IUT),  
    'Property Identifier' = List\_Of\_Session\_Keys
  11. BEFORE **Acknowledgment Fail Time**  
    RECEIVE BACnet-Error-PDU,  
        Error Class = SECURITY,  
        Error Code = OTHER

### 9.39 General Testing of Service Execution

This subclause defines the tests necessary to demonstrate that a device can peacefully coexist on a BACnet internetwork. These are general tests that are not associated with any particular network service.

#### 9.39.1 Unsupported Confirmed Services Test

Purpose: This test case verifies that the IUT will reject any confirmed services that it does not support.

Test Steps:

1. REPEAT X = (all confirmed services that the IUT does not execute) DO {  
    TRANSMIT X  
    RECEIVE BACnet-Reject-PDU,  
        'Reject Reason' = UNRECOGNIZED\_SERVICE  
}
2. TRANSMIT (a currently undefined confirmed service)
3. RECEIVE BACnet-Reject-PDU,  
    'Reject Reason' = UNRECOGNIZED\_SERVICE

Passing Result: The device responds correctly for each unsupported confirmed service.

**9.39.2 Unsupported Unconfirmed Services Test**

Purpose: This test case verifies that the IUT will quietly accept and discard any unconfirmed services that it does not support. When determining the set of services to send to the IUT, the UnconfirmedPrivateTransfer service should be included regardless of whether the IUT supports it or not. The UnconfirmedPrivateTransfer service shall be sent with a vendor ID/Service Number pair not supported by the device.

Configuration Requirements: This test requires that the IUT be placed into a normal operating state in which it will not initiate any requests.

Test Steps:

1. VERIFY System\_Status == OPERATIONAL | OPERATIONAL\_READ\_ONLY
2. REPEAT X = (all unconfirmed services that the IUT does not execute) DO {  
     TRANSMIT X  
     BEFORE **Internal Processing Fail Time**  
     CHECK (verify that the IUT did not reset and that the IUT did not send any packets)  
     VERIFY System\_Status = (the value of System\_Status read in step 1)  
   }

Passing Result: The IUT does not reset and sends no packets in response to the services.

**9.40 AuditLogQuery Service Execution Tests****9.40.1 AuditLogQuery Service Positive Tests****9.40.1.1 AuditLogQuery By Target Test**

Purpose: Verify that an Audit Log correctly returns notifications filtered by a specific target.

Test Concept: An Audit Log, O1, containing a sequence of notifications related to a set of audit targets is queried about a specific target, T1. The query is repeated for each standard form of target-based query, and the result is compared against the expected set of returned notifications.

Configuration Requirements: The Audit Log, O1, contains a set of audit notifications which contains 0 or more notifications about audit target T1.

Test Steps:

- REPEAT Q = (each query in (
- By Target Device Identifier,
  - By Target Device Identifier & Target Device Address,
  - By Target Device Identifier & Target Object Identifier,
  - By Target Device Identifier & Target Property Identifier,
  - By Target Device Identifier & Target Array Index,
  - By Target Device Identifier & Target Priority,
  - By Target Device Identifier & Operations,
  - By Target Device Identifier & Result Filter (failed),
  - By Target Device Identifier & Result Filter (success),
  - By Target Device Identifier & Target Object Identifier & Target Property Identifier
- ) {
1. TRANSMIT AuditLogQuery-Request,  
     'Audit Log' = O1,  
     'Query Parameters' = Q,  
     'Requested Count' = Total\_Record\_Count
  2. RECEIVE AuditLogQuery-ACK,  
     'Audit Log' = O1,  
     'Records' = (RSEQ: a set of records),
  3. WHILE (the length of RSEQ is not the number of expected records) {  
     TRANSMIT AuditLogQuery-Request,  
     'Audit Log' = O1,  
     'Query Parameters' = Q,

```

 'Start At Sequence Number' = (the 'Sequence Number' from the last entry in RSEQ)
 'Requested Count' = Total_Record_Count
 RECEIVE AuditLogQuery-ACK,
 'Audit Log' = O1,
 Records' = (NXTSEQ: a set of records),
 IF (the length of NXTSEQ is 0) THEN
 ERROR "expected more records from the Audit Log"
 RSEQ = (RSEQ record with NXTSEQ records appended)
}
4. CHECK(that the records in RSEQ is the set expected and are returned in sequence number order)
}

```

#### 9.40.1.2 AuditLogQuery By Source Test

Purpose: Verify that an Audit Log correctly returns notifications filtered by a specific source.

Test Concept: An Audit Log, O1, containing a sequence of notifications related to a set of audit sources is queried about a specific source, S1. The query is repeated for each standard form of source-based query, and the result is compared against the expected set of returned notifications.

Configuration Requirements: The Audit Log, O1, contains a set of audit notifications which contains 0 or more notifications about audit source S1.

Test Steps:

```

REPEAT Q = (each query in (
 By Source Device Identifier,
 By Source Device Identifier & Source Device Address,
 By Source Device Identifier & Source Object Identifier,
 By Source Device Identifier & Operations,
 By Source Device Identifier & Result Filter (failed),
 By Source Device Identifier & Result Filter (success),
) {
1. TRANSMIT AuditLogQuery-Request,
 'Audit Log' = O1,
 'Query Parameters' = Q,
 'Requested Count' = Total_Record_Count
2. RECEIVE AuditLogQuery-ACK,
 'Audit Log' = O1,
 'Records' = (RSEQ: a set of records),
3. WHILE (the length of RSEQ is not the number of expected records) {
 TRANSMIT AuditLogQuery-Request,
 'Audit Log' = O1,
 'Query Parameters' = Q,
 'Start At Sequence Number' = (the 'Sequence Number' from the last entry in RSEQ)
 'Requested Count' = Total_Record_Count
 RECEIVE AuditLogQuery-ACK,
 'Audit Log' = O1,
 'Records' = (NXTSEQ: a set of records),
 IF (the length of NXTSEQ is 0) THEN
 ERROR "expected more records from the Audit Log"
 RSEQ = (RSEQ record with NXTSEQ records appended)
}
4. CHECK(that the records in RSEQ is the set expected and are returned in sequence number order)
}
}

```

#### 9.40.2 AuditLogQuery Negative Tests

##### 9.40.2.1 Attempting to Query a Non-existent Audit Log

Purpose: Verify that the correct error is returned when the queried log does not exist.

Test Concept: Send an AuditLogQuery request to the IUT for an AuditLog object which does not exist. Verify that the IUT returns an Error Class of OBJECT and an error code of UNKNOWN\_OBJECT.

Test Steps:

1. TRANSMIT AuditLogQuery-Request,  
     'Audit Log' = (an audit log not in the IUT),  
     'Query Parameters' = (any valid value),  
     'Requested Count' = (any valid value)
2. RECEIVE BACnet-Error PDU,  
     'Error Class' = OBJECT,  
     'Error Code' = UNKNOWN\_OBJECT

#### 9.41 WriteGroup Tests

##### 9.41.1 Positive WriteGroup Tests

###### 9.41.1.1 Channel and Group Number Test

Purpose: To verify that the Channel object executes a WriteGroup service request only when containing a specified channel number and Group Number by a request, and the Channel object ignores a request otherwise. If a Group Number is 0, the Channel object ignores a service even when its Control\_Groups property is set to 0.

Test Concept: The Channel Object, O1, will be assigned a specific value to its channel number and Group Number. When a device containing O1 receives a WriteGroup service, O1 executes the request and propagates a specified value to its destination only if A) the O1's channel number is the same as specified number by the request and B) the O1's Control\_Groups contains the specified Group Number, except for a case when a Group Number was 0.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commandable property of an object on either a local or remote device. For a commandable property, all prioritized commands have to be relinquished, and any **Minimum ON/OFF Time** has to be accounted for prior to the test. An initial value of a commandable property must be the same as RD, which is its Relinquish\_Default value. The value to be propagated must be a valid value that does not require coercion.

Test Steps:

- Obtain the data which will be used for the **Channel Write Fail Time** later in the steps
- 1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Obtain the Channel Object's target object reference
- 2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
- Set arbitrary numbers for Channel\_Number and Control\_Group to O1
- 3. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = Channel\_Number  
     'Property Value' = (CN: Any valid value)  
     'Property Identifier' = Control\_Groups  
     'Property Value' = (CG: Any length of an array containing at least 1 non-zero element)
- 4. RECEIVE BACnet-SimpleACK-PDU
- Send a WriteGroup with a mismatching channel number and Group Number
- 5. TRANSMIT WriteGroup-Request,  
     'Group Number' = (A valid value larger than 0 and not contained by CG)  
     'Write Priority' = (any valid value),  
     'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
- 6. WAIT **Channel Write Fail Time** \* LEN
- Make sure that O1 did not propagate a value to its target references
- 7. VERIFY PR = RD
- Send a WriteGroup with a matching channel number and a mismatching group number
- 8. TRANSMIT WriteGroup-Request,  
     'Group Number' = (A valid value larger than 0 and not contained by CG)  
     'Write Priority' = (any valid value),



- 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)
9. **WAIT Channel Write Fail Time \* LEN**  
 -- Make sure that O1 did not propagate value to its target references
10. **VERIFY PR = RD**  
 -- Send a WriteGroup with a mismatching channel number and a matching group number
11. **TRANSMIT WriteGroup-Request**,  
 'Group Number' = (Any non 0 values contained by CG)  
 'Write Priority' = (any valid value),  
 'Change List' = (Any valid value different than CN, no overriding priority, Y: a valid value different than RD)
12. **WAIT Channel Write Fail Time \* LEN**  
 -- Make sure that O1 did not propagate value to its target references
13. **VERIFY PR = RD**  
 -- Send a WriteGroup service with a matching channel number and group number
14. **TRANSMIT WriteGroup-Request**,  
 'Group Number' = (Any non 0 values contained by CG)  
 'Write Priority' = (any valid value),  
 'Change List' = (CN, no overriding priority, Y: a valid value different than RD)  
 -- Make sure that O1 did propagate value to its target references
15. **VERIFY PR = Y**  
 -- Change Control\_Groups to 0
16. **TRANSMIT WritePropertyMultiple-Request**,  
 'Object Identifier' = O1,  
 'Property Identifier' = Control\_Groups  
 'Property Value' = 0
17. **RECEIVE BACnet-SimpleACK-PDU**  
 -- Send a WriteGroup with 0 Group number
18. **TRANSMIT WriteGroup-Request**,  
 'Group Number' = 0  
 'Write Priority' = (any valid value),  
 'Change List' = (CN, no overriding priority, Z: a valid value different than Y)
19. **WAIT Channel Write Fail Time \* LEN**  
 -- Make sure that O1 did not propagate value to its target references
20. **VERIFY PR = Y**

#### 9.41.1.2 Write Priority and Overriding Priority Test

Purpose: To verify that the overridingPriority, if provided, specifies the priority for writing the value. Otherwise, the 'Write Priority' parameter specifies the priority for writing.

Test Concept: The Channel Object, O1, receives the WriteGroup with P1 as its 'Write Priority', and it is verified that P1 is used for writing the value. O1 then receives another WriteGroup with P1 as its 'Write Priority' and P2 as its overridingPriority, and it is verified that P2 is used for writing the value.

Configuration Requirements: Configure one of the entry for the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with Priority\_Array on either a local or remote device. All prioritized commands have to be relinquished, and any **Minimum ON/OFF Time** has to be accounted for prior to the test.

Test Steps:

- Obtain the data which will be used for the **Channel Write Fail Time** later in the steps
1. **READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0**  
 -- Write to O1 using P1 as its Write Priority
2. **TRANSMIT WriteGroup-Request**,  
 'Group Number' = (one of the Control\_Group values configured in O1),  
 'Write Priority' = (P1: Any valid value)  
 'Change List' = (O1's channel number, no overriding priority, Y: any valid value)
3. **WAIT Channel Write Fail Time \* LEN**  
 -- Make sure that P1 is used for writing the value
4. **VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1**  
 -- Write to O1 using P1 as its Write Priority, P2 as its overridingPriority



5. TRANSMIT WriteGroup-Request,  
 'Group Number' = (one of the Control\_Group values configured in O1),  
 'Write Priority' = P1  
 'Change List' = (O1's channel number, P2: Any valid value different than P1, Z: any valid value different than Y)  
 6. WAIT **Channel Write Fail Time** \* LEN  
 -- Make sure that P2 is used for writing the value  
 7. VERIFY (O2), Priority\_Array = Z, ARRAY INDEX = P2  
 -- Make sure that no update on Priority\_Array[P1]  
 8. VERIFY (O2), Priority\_Array = Y, ARRAY INDEX = P1

#### 9.41.1.3 Relinquish Control Test

Purpose: To verify that if a BACnetGroupChannelValue specifies a NULL value, it serves the same function as if NULL had been used with WriteProperty.

Test Concept: The Channel Object, O1, receives the WriteGroup service to propagate a value to its destination object property reference, PR. PR is verified to have the value updated accordingly. The O1 then receives another WriteGroup service with the BACnetGroupChannelValue specifying a NULL value. PR is verified to have its Relinquish\_Default value.

Configuration Requirements: Configure entry X of the Channel object's List\_Of\_Object\_Property\_References to refer to a commendable property of an object O2 with a Relinquish\_Default set to RD on either a local or remote device. All prioritized commands have to be relinquished, and any **Minimum ON/OFF Time** has to be accounted for prior to the test.

Test Steps:

- Obtain the data which will be used for the **Channel Write Fail Time** later in the steps
1. READ LEN = List\_Of\_Object\_Property\_References, ARRAY INDEX = 0
- Obtain the Channel Object's target object reference
2. READ PR = List\_Of\_Object\_Property\_References, ARRAY INDEX = X
- Let the Channel Object propagate a value to its target
3. TRANSMIT WriteGroup-Request,  
 'Group Number' = (One of the Control\_Group values configured in O1),  
 'Write Priority' = (Any valid value)  
 'Change List' = (O1's channel number, no overriding priority, X: any valid value different than RD)  
 4. WAIT **Channel Write Fail Time** \* LEN  
 5. VERIFY PR = X  
 -- Let the Channel Object relinquish control of the target
6. TRANSMIT WriteGroup-Request,  
 'Group Number' = (One of the Control\_Group values configured in O1),  
 'Write Priority' = (Any valid value)  
 'Change List' = (O1's channel number, no overriding priority, NULL)  
 7. WAIT **Channel Write Fail Time** \* LEN  
 8. VERIFY PR = RD

#### 9.41.1.4 Inhibit Delay Test with WriteGroup

Purpose: In the case of WriteGroup, verify that Execution\_Delay always occurs unless the WriteGroup service parameter 'Inhibit Delay' is TRUE, and the Channel object property Allow\_Group\_Delay\_Inhibit is present and has the value TRUE.

Test Concept: Setup List\_Of\_Object\_Property\_References of the Channel Object, O1, to contain 2 valid entries, PR1 and PR2, and provide each with an execution delay (ED1 and ED2) which are larger than 0. Allow\_Group\_Delay\_Inhibit is set to TRUE. When a WriteGroup service is sent to O1 without 'Inhibit Delay' parameter, it is verified that Execution\_Delay occurs. When another WriteGroup service is sent to O1 with 'Inhibit Delay' set to False, it is verified that Execution\_Delay still occurs. Finally, when another WriteGroup service is sent with 'Inhibit Delay' set to TRUE, it is verified that Execution\_Delay does not occur.

Configuration Requirements: PR1 and PR2 shall be references to writable properties and shall be the same datatype. ED1 and ED2 shall be values which are large enough that the delay between writes is sufficient for the test. V1 and V2 shall be of the expected datatype for PR1 and PR2 so that no coercion occurs and shall be different values.

Test Steps:

-- Setup the Channel object

1. WRITE List\_Of\_Object\_Property\_References = (PR1, PR2)
2. WRITE Execution\_Delay = (ED1, ED2)
3. WRITE Allow\_Group\_Delay\_Inhibit = TRUE
4. READ V1 = PR1
5. READ V2 = PR2

-- Send a WriteGroup without 'Inhibit Delay' parameter

6. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, X: any valid value different than V1 or V2)

7. WAIT **Channel Write Fail Time**

-- Make sure that Execution\_Delay occurs

8. VERIFY PR1 = V1
9. VERIFY PR2 = V2
10. WAIT (ED1)
11. VERIFY PR1 = X
12. VERIFY PR2 = V2
13. WAIT (ED2 – ED1)
14. VERIFY PR2 = X

-- Send a WriteGroup with 'Inhibit Delay' set to FALSE

15. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, Y: any valid value different from X)
  - 'Inhibit Delay' = FALSE

16. WAIT **Channel Write Fail Time**

-- Make sure that Execution\_Delay occurs

17. VERIFY PR1 = X
18. VERIFY PR2 = X
19. WAIT (ED1)
20. VERIFY PR1 = Y
21. VERIFY PR2 = X
22. WAIT (ED2 – ED1)
23. VERIFY PR2 = Y

-- Send a WriteGroup with 'Inhibit Delay' set to TRUE

24. TRANSMIT WriteGroup-Request,
  - 'Group Number' = (one of the Control\_Group values configured in O1),
  - 'Write Priority' = (Any valid value)
  - 'Change List' = (O1's channel number, no overriding priority, Z: any valid value different from Y)
  - 'Inhibit Delay' = TRUE

-- Make sure that Execution\_Delay does NOT occur

25. WAIT **Channel Write Fail Time**

26. VERIFY PR1 = Z
27. VERIFY PR2 = Z

## 9.42 SubscribeCOVPropertyMultiple Service Execution Tests

### 9.42.1 Positive SubscribeCOVPropertyMultiple Service Execution Tests

#### 9.42.1.1 Supports Non-Timestamped Notifications

Purpose: To verify that the IUT can execute a COVM Notification without providing a timestamp

Test Concept: A subscription for COVM notifications, with the Timestamped parameter set to FALSE. Verify that the IUT sends the appropriate COVM notification in response.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (ID1: any valid process identifier),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = (any valid delay between 1 and 3600),  
 'List of COV Subscription Specifications' = (any valid list with 'Timestamped' set to FALSE in all entries)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedCOVNotificationMultiple-Request,  
 'Subscriber Process Identifier' = ID1,  
 'Initiating Device Identifier' = TD,  
 'Time Remaining' = (a value ~ L),  
 -- 'Timestamp' = (absent)  
 'List of COV Notifications' = (values appropriate to the properties subscribed to)  
 TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
 WHILE (notifications have not been received for all subscribed to items)  
 BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
 'Subscriber Process Identifier' = ID1,  
 'Initiating Device Identifier' = TD,  
 'Time Remaining' = (a value ~ L),  
 -- 'Timestamp' = (absent)  
 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)

#### 9.42.1.2 Supports Timestamped Notifications

Purpose: To verify that the IUT can execute a COVM Notification providing a timestamp

Test Concept: A subscription for COVM notifications with the Timestamped parameter set to TRUE for at least 1 entry in the list of subscriptions, and FALSE for at least 1 entry in the list of subscriptions, is sent to the IUT for properties for which the IUT supports COVM. Verify that the IUT sends the appropriate COVM notification in response.

Notes to Tester: If the IUT only supports COVM for one property in one object, then the subscription shall be for the single property with Timestamped set to TRUE.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (ID1: any valid process identifier),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = (L: a valid lifetime),  
 'Max Notification Delay' = (any valid delay between 1 and 3600),  
 'List of COV Subscription Specifications' = (any valid list of properties which exist in the IUT for which the IUT supports COVM with Timestamped set to TRUE for at least one, and FALSE for at least one)
2. RECEIVE BACnet-SimpleACK-PDU

3. IF (the subscription was for confirmed notifications) THEN
  - BEFORE **Notification Fail Time**
    - RECEIVE ConfirmedCOVNotificationMultiple-Request,
      - 'Subscriber Process Identifier' = ID1,
      - 'Initiating Device Identifier' = IUT,
      - 'Time Remaining' = (a value  $\sim$  L),
      - 'Timestamp' = (an appropriate timestamp)
      - 'List of COV Notifications' = (values appropriate to the properties subscribed to along with 'Time of Change' values only for those for which timestamps were requested)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - WHILE (notifications have not been received for all subscribed to items)
      - BEFORE **Notification Fail Time**
        - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
          - 'Subscriber Process Identifier' = ID1,
          - 'Initiating Device Identifier' = IUT,
          - 'Time Remaining' = (a value  $\sim$  L),
          - 'Timestamp' = (an appropriate timestamp)
          - 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to along with 'Time of Change' values only for those for which timestamps were requested)

#### 9.42.1.3 Confirmed Change of Value Notification From Property Value

Purpose: To verify that the IUT initiates a ConfirmedCOVMultipleNotification service request when a subscribed to property changes.

Test Concept: A COVM subscription is made which contains a subscription to property P1 in object O1. The value of P1 is changed, and it is verified that the IUT sends a COVM notification.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (ID1: any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE,
  - 'Lifetime' = (L: any valid lifetime),
  - 'Max Notification Delay' = (any valid value),
  - 'List of COV Subscription Specifications' = (PROPS: a valid list of properties for which the IUT supports COVM including P1 in O1)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (a value  $\sim$  L),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent),
    - 'List of COV Notifications' = (values appropriate to the subscribed to properties)
4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (a change to P1 that should cause a COVM notification)
6. BEFORE **Notification Fail Time**
  - RECEIVE ConfirmedCOVNotificationMultiple-Request,
    - 'Subscriber Process Identifier' = ID1,
    - 'Initiating Device Identifier' = IUT,
    - 'Time Remaining' = (a value greater than 0 and less than L),
    - 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent),
    - 'List of COV Notifications' = (a list consisting of a valid value for P1 and values for any co-reported properties as described in clause 13.1)
7. TRANSMIT BACnet-SimpleACK-PDU

#### 9.42.1.4 Unconfirmed Change of Value Notification From Property Value

Purpose: To verify that the IUT initiates an UnconfirmedCOVMultipleNotification service request when a subscribed to property changes.

Test Concept: A COVM subscription is made which contains a subscription with 'Issue Confirmed Notifications' = FALSE to property P1 in object O1. The value of P1 is changed, and it is verified that the IUT sends a unicast COVM notification.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (ID1: any valid process identifier),  
 'Issue Confirmed Notifications' = FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = (MND: any valid value)  
 'List of COV Subscription Specifications' = (a valid list of properties for which the IUT supports COVM including P1 in O1)
2. RECEIVE BACnet-SimpleACK-PDU
3. WHILE (notifications have not been received for all subscribed to items)  
 BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
 'Subscriber Process Identifier' = ID1,  
 'Initiating Device Identifier' = IUT,  
 'Time Remaining' = (a value  $\sim$  L),  
 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent)  
 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)
4. MAKE (a change to the P1 that should cause a COVM notification)
5. BEFORE **Notification Fail Time**  
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
 'Subscriber Process Identifier' = ID1,  
 'Initiating Device Identifier' = IUT,  
 'Time Remaining' = (a value greater than 0 and less than the requested lifetime),  
 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent),  
 'List of COV Notifications' = (a list consisting of a valid value for P1 and values for any co-reported properties as described in clause 13.1)

#### 9.42.1.5 Supports Subscriptions to Multiple Properties Using Multiple Requests

Purpose: To verify the server adds new subscriptions to existing COVM contexts when requested.

Test Concept: A subscription for COVM notifications is established for property P1 of object O1. A second subscription is sent using the same COVM context for property P2 in object O2. Verify that the IUT's Active\_COV\_Multiple\_Subscriptions property is correctly updated after each subscription.

Configuration Requirements: There are no active COVM subscriptions for properties in the IUT. If the IUT cannot be configured to have 2 properties which support COVM subscriptions, then this test shall be skipped.

Notes to Tester: Objects O1 and O2 can be the same object, and properties P1 and P2 can be the same property, but (O1, P1) must be different than (O2, P2).

Test Steps:

1. CHECK (the IUT's Active\_COV\_Multiple\_Subscriptions property is empty)
2. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (ID1: any valid process identifier),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = (any valid notification delay),

```

'List of COV Subscription Specifications' = {(
 'Monitored Object' = O1,
 'List of COV References' = {(
 'Monitored Property' = P1,
 'COV Increment' = (any valid increment, or empty if P1 is
 not numeric),
 'Timestamped' = TRUE | FALSE)}
)}
3. RECEIVE BACnet-SimpleAck-PDU
4. IF (confirmed notifications were requested) THEN
 BEFORE Notification Fail Time
 RECEIVE ConfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = ID1,
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (any value ~= L),
 'Timestamp' = (an appropriate timestamp)
 'List of COV Notifications' = (a list of values of length 1 indicating P1's value)
 TRANSMIT BACnet-SimpleACK-PDU
ELSE
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = ID1,
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (any value ~= L),
 'Timestamp' = (an appropriate timestamp, or absent if timestamps not
 requested)
 'List of COV Notifications' = (a list of values of length 1 indicating P1's value)
5. VERIFY Active_COV_Multiple_Subscriptions = (a list with one entry for COVM context ID1 with 1 entry
 for the subscription to P1)
6. TRANSMIT SubscribeCOVPropertyMultiple-Request
 'Subscriber Process Identifier' = ID1,
 'Issue Confirmed Notifications' = TRUE | FALSE,
 'Lifetime' = L,
 'Max Notification Delay' = (any valid notification delay),
 'List of COV Subscription Specifications' = {(
 'Monitored Object' = O2,
 'List of COV References' = {(
 'Monitored Property' = P2,
 'COV Increment' = (any valid increment, or empty if P2 is
 Not numeric),
 'Timestamped' = TRUE | FALSE)}
)}
7. RECEIVE BACnet-SimpleAck-PDU
8. IF confirmed notifications were requested THEN
 BEFORE Notification Fail Time
 RECEIVE ConfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = ID1,
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (any value ~= L),
 'Timestamp' = (an appropriate timestamp, or absent if timestamps not
 requested)
 'List of COV Notifications' = (a list of values of length 1 indicating P2's new value)
 TRANSMIT BACnet-SimpleACK-PDU
ELSE
 WHILE (notifications have not been received for all subscribed to items)
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = ID1,
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (any value ~= L),

```



'Timestamp' = (an appropriate timestamp, or absent if timestamps not requested)  
 'List of COV Notifications' = (a list of values of length 1 indicating P2's new value)

9. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list with one entry for COVM context ID1 with 2 entries for P1 and P2)

#### 9.42.1.6 Ensuring 5 Concurrent COV-Multiple Contexts With 5 COV-References Per Context

Purpose: To verify that the IUT can support 5 COV-multiple contexts with 5 COV-references each.

Test Concept: Subscriptions for COVM notifications are made using process identifiers PID1 through PID5. The required post subscription notifications are verified. Once all subscriptions are made, the Active\_COV\_Multiple\_Subscriptions is verified to contain all subscriptions.

Configuration Requirements: The IUT has no active COVM subscriptions.

Test Steps:

1. REPEAT (X=PID1 to PID5) DO {
  - TRANSMIT SubscribeCOVPropertyMultiple-Request
    - 'Subscriber Process Identifier' = X,
    - 'Issue Confirmed Notifications' = TRUE | FALSE,
    - 'Lifetime' = (L, any value large enough to complete the test),
    - 'Max Notification Delay' = (any valid value),
    - 'List of COV Subscription Specifications' = (any valid list of properties for which the IUT supports COVM)
  - RECEIVE BACnet-SimpleACK-PDU
  - IF (if confirmed notifications were requested) THEN
    - BEFORE **Notification Fail Time**
      - RECEIVE ConfirmedCOVNotificationMultiple-Request,
        - 'Subscriber Process Identifier' = X,
        - 'Initiating Device Identifier' = IUT,
        - 'Time Remaining' = (a value  $\sim$  L),
        - 'Timestamp' = (any appropriate timestamp, if subscribed to timestamped notifications),
        - 'List of COV Notifications' = (values appropriate to the subscribed to properties)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - WHILE (notifications have not been received for all subscribed to items)
      - BEFORE **Notification Fail Time**
        - RECEIVE UnconfirmedCOVNotificationMultiple-Request,
          - 'Subscriber Process Identifier' = X,
          - 'Initiating Device Identifier' = IUT,
          - 'Time Remaining' = (any valid value),
          - 'Timestamp' = (any appropriate timestamp, if subscribed to timestamped notifications)
          - 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)

2. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list of 5 COVM contexts as subscribed to)

#### 9.42.1.7 Supports Client-Supplied COV Increment

Purpose: To verify that the IUT abides by client supplied COV increments from SubscribeCOVPropertyMultiple requests.

Test Concept: A subscription for COVM notifications is made to a numeric property P1 which supports COVM in object O1. The COV Increment, N, is specified in the subscription request. Verify that the COV Increment N is stored in the COVM context for this subscription. The value of P1 is changed by less than the COV Increment, and the TD waits to ensure the IUT does not generate a notification. The value of P1 is changed such that the total change is more than N, and it is verified that the IUT sends a notification within the delay time.



Configuration Requirements: If the property being subscribed to has a related COV\_Increment property in the object, then the value of N should be significantly different than the value of the COV\_Increment property. If the object does not have a COV\_Increment property, then N shall be significantly different than the device's internal COV Increment.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (ID1: any valid process identifier),  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = (MND: any valid value),  
 'List of COV Subscription Specifications' = {'Monitored Object' = O1,  
     'List of COV References' = {(  
       'Monitored Property' = P1,  
       'COV Increment' = N,  
       'Timestamped' = TRUE | FALSE)}  
     )}
2. RECEIVE BACnet-SimpleACK-PDU
3. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list containing a COVM context for ID1 containing 1 entry for P1 with a COV\_Increment of N)
4. MAKE (P1's value change by less than COV Increment)
5. WAIT **Notification Fail Time** + MND
6. CHECK (verify that the IUT did not transmit a notification message for the monitored property)
7. MAKE (P1's value change such that the total change to P1 is slightly more than N)
8. IF (the subscription was for confirmed notifications) THEN  
     BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = ID1,  
     'Initiating Device Identifier' = IUT,  
     'Time Remaining' = (any valid value greater than 0 and less than L),  
     'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent)  
     'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)  
     TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
     WHILE (notifications have not been received for all subscribed to items)  
     BEFORE **Notification Fail Time**  
     RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
     'Subscriber Process Identifier' = (ID1),  
     'Initiating Device Identifier' = IUT,  
     'Time Remaining' = (any valid value greater than 0 and less than L),  
     'Timestamp' = (an appropriate timestamp, if subscribed to timestamped notifications, otherwise absent)  
     'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)

#### 9.42.1.8 Updating Existing Subscriptions

Purpose: To verify that the IUT supports resubscriptions to extend the lifetime of COVM contexts.

Test Concept: A COVM subscription is made for 1 or more properties in the IUT. The IUT shall be made to transmit a notification to the TD and the Time Remaining value is validated. Before the subscription expires, the TD resubscribes with a different, and longer, lifetime and the new lifetime is verified in the resultant COVM notification.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = ID1,  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = (MND: any valid value)  
 'List of COV Subscription Specifications' = (PROPS: a valid list of subscriptions)

2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotificationMultiple-Request,  
         'Subscriber Process Identifier' = IUD1,  
         'Initiating Device Identifier' = IUT,  
         'Time Remaining' = (TR: TR  $\sim$  L),  
         'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
                             Notifications, otherwise absent)  
         'List of COV Notifications' = (values appropriate to each entry in PROPS)  
     TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
     WHILE (notifications have not been received for all subscribed to items)  
         **BEFORE Notification Fail Time**  
         RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
             'Subscriber Process Identifier' = ID1,  
             'Initiating Device Identifier' = IUT,  
             'Time Remaining' = (TR: TR  $\sim$  L),  
             'Timestamp' = (an appropriate timestamp, if subscribed to  
                                 timestamped notifications, otherwise absent),  
             'List of COV Notifications' = (values appropriate to some or all  
                                             entries in PROPS)  
 4. MAKE (a change to a monitored property, P1, that should cause a COVM notification)  
 5. WAIT N seconds, where  $L > N >$  the resolution of the IUT's COVM lifetime timer  
 6. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotificationMultiple-Request,  
         'Subscriber Process Identifier' = ID1,  
         'Initiating Device Identifier' = IUT,  
         'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),  
         'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
                             notifications, otherwise absent)  
         'List of COV Notifications' = (a list of values of length 1 indicating P1's new value)  
     TRANSMIT BACnet-SimpleACK-PDU  
 ELSE  
     WHILE (notifications have not been received for all subscribed to items)  
         **BEFORE Notification Fail Time**  
         RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
             'Subscriber Process Identifier' = ID1,  
             'Initiating Device Identifier' = IUT,  
             'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),  
             'Timestamp' = (an appropriate timestamp, if subscribed to  
                                 timestamped notifications, otherwise absent),  
             'List of COV Notifications' = (values appropriate to some or all  
                                             entries in PROPS)  
 7. TRANSMIT SubscribeCOVPropertyMultiple-Request  
     'Subscriber Process Identifier' = ID1,  
     'Issue Confirmed Notifications' = (the same value used previously),  
     'Lifetime' = (L2: where  $L < L2 \leq 28800$ ),  
     'Max Notification Delay' = MND,  
     'List of COV Subscription Specifications' = PROPS  
 8. RECEIVE BACnet-SimpleACK-PDU  
 9. IF (the subscription was for confirmed notifications) THEN  
     **BEFORE Notification Fail Time**  
     RECEIVE ConfirmedCOVNotificationMultiple-Request,  
         'Subscriber Process Identifier' = ID1,  
         'Initiating Device Identifier' = IUT,  
         'Time Remaining' = (TR2: TR  $\sim$  L2),  
         'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
                             notifications, otherwise absent)

```

 'List of COV Notifications' = (values appropriate to each entry in PROPS)
 TRANSMIT BACnet-SimpleACK-PDU
ELSE
 WHILE (notifications have not been received for all subscribed to items)
 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = (the same identifier used in step 2),
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (TR2: TR2 ~= L2),
 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped
 notifications, otherwise absent),
 'List of COV Notifications' = (values appropriate to some or all entries
 in PROPS)

```

#### 9.42.1.9 Canceling Subsets of COVM Subscriptions

Purpose: To verify that the IUT correctly cancels COVM subscriptions for some, not all, of the properties subscribed to in a COVM context.

Test Concept: A subscription for COVM notifications is established for multiple properties within the IUT. Before the subscriptions expire, one of the subscriptions is cancelled. Verify that the IUT's Active\_COV\_Multiple\_Subscriptions property only contains an entry for the remaining subscriptions.

Configuration Requirements: There are no active COVM subscription for properties in the IUT. If the IUT cannot be configured to have 2 properties which support COVM subscriptions, then this test shall be skipped.

Test Steps:

1. VERIFY Active\_COV\_Multiple\_Subscriptions = ()
2. TRANSMIT SubscribeCOVPropertyMultiple-Request
 

```

 'Subscriber Process Identifier' = (any valid process identifier),
 'Issue Confirmed Notifications' = FALSE,
 'Lifetime' = L,
 'Max Notification Delay' = (any valid notification delay),
 'List of COV Subscription Specifications' = (a list of 2 or more properties for which the IUT supports COVM)

```
3. RECEIVE BACnet-SimpleACK-PDU
4. WHILE (notifications have not been received for all subscribed to items)
 

```

 BEFORE Notification Fail Time
 RECEIVE UnconfirmedCOVNotificationMultiple-Request,
 'Subscriber Process Identifier' = ID1,
 'Initiating Device Identifier' = IUT,
 'Time Remaining' = (any value ~= L),
 'Timestamp' = (an appropriate timestamp, or absent if not requested)
 'List of COV Notifications' = (values appropriate to some or all of the properties subscribed to)

```
5. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list with 1 COVM context containing all properties subscribed to)
6. TRANSMIT SubscribeCOVPropertyMultiple-Request
 

```

 'Subscriber Process Identifier' = ID1,
 'Issue Confirmed Notifications' = FALSE
 'Lifetime' = (absent)
 'Max Notification Delay' = (absent)
 'List of COV Subscription Specifications' = (CANCELLED: a subset of the properties subscribed to)

```
7. RECEIVE BACnet-SimpleACK-PDU
8. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list with 1 COVM context containing all remaining properties subscribed to, excluding those in CANCELLED)

#### 9.42.1.10 Canceling Expired or Non-Existing Subscriptions

Purpose: To verify the IUT does not return an error when the client cancels a COVM subscription that doesn't match any of the COV contexts in the IUT's list of active subscriptions.

Test Concept: Send a SubscribeCOVPropertyMultiple request to cancel a subscription for property P1 in object O1, which is not in the list of subscriptions in the IUT's Active\_COV\_Multiple\_Subscriptions property. Verify that the IUT sends a BACnet-SimpleACK-PDU in response.

Configuration Requirements: The IUT is configured with 1 or more COVM subscriptions. One of the subscriptions is using a process identifier ID1 and includes a subscription to property P1 in object O1. Property P2 in object O2 shall not be included in the subscriptions for ID1 (but may in subscriptions using a different process identifier). Where possible, P2 in O2 should be a property for which the IUT supports COVM subscriptions.

Test Steps:

1. READ COVM\_LIST = Active\_COV\_Multiple\_Subscriptions
2. CHECK (COVM\_LIST contains an COVM context with a process identifier of ID1 and includes a subscription to property P1 in object O1)
3. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = (any valid process identifier which is not ID1),  
 'Issue Confirmed Notifications' = (the value matching the entry for ID1),  
 -- 'Lifetime' = (absent),  
 -- 'Max Notification Delay' = (absent),  
 'List of COV Subscription Specifications' = (a list with 1 entry matching the subscription details for P1 in O1)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY Active\_COV\_Multiple\_Subscriptions = COVM\_LIST
6. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = ID1,  
 'Issue Confirmed Notifications' = (the value matching the entry for ID1),  
 -- 'Lifetime' = (absent),  
 -- 'Max Notification Delay' = (absent),  
 'List of COV Subscription Specifications' = (a list with 1 entry referencing P2 in O2)
7. RECEIVE BACnet-SimpleACK-PDU
8. VERIFY Active\_COV\_Multiple\_Subscriptions = COVM\_LIST

#### 9.42.1.11 Subscription Expiration Test

Purpose: To verify that the IUT removes subscriptions from the list of active subscriptions once the subscription lifetime has elapsed.

Test Concept: A COVM subscription is made for 1 or more properties in the IUT. One of the subscribed to properties is made to change, and it is verified that the IUT transmits a notification to the TD containing an accurate Time Remaining value. The tester then waits for the subscription to expire, and it is verified that Active\_COV\_Multiple\_Subscriptions is updated. The property is changed again, and it is verified that the IUT does not send a notification.

Configuration Requirements: No existing subscription exists for ID1 for the TD. A value for L is chosen which is long enough to complete the initial test steps, but which is short enough to wait for it to expire.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
 'Subscriber Process Identifier' = ID1,  
 'Issue Confirmed Notifications' = TRUE | FALSE,  
 'Lifetime' = L,  
 'Max Notification Delay' = 0,  
 'List of COV Subscription Specifications' = (a valid list of subscriptions)
2. RECEIVE BACnet-SimpleACK-PDU
3. IF (the subscription was for confirmed notifications) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE ConfirmedCOVNotificationMultiple-Request,  
 'Subscriber Process Identifier' = ID1,  
 'Initiating Device Identifier' = IUT,  
 'Time Remaining' = (a value  $\sim$  L),  
 'Timestamp' = (an appropriate timestamp, if subscribed to timestamped Notifications, otherwise absent),  
 'List of COV Notifications' = (values appropriate to the properties subscribed to)

TRANSMIT BACnet-SimpleACK-PDU  
ELSE

WHILE (notifications have not been received for all subscribed to items)

BEFORE **Notification Fail Time**

RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
'Subscriber Process Identifier' = ID1,  
'Initiating Device Identifier' = IUT,  
'Time Remaining' = (a value approximately equal to, but not greater than,  
the requested subscription lifetime),  
'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
Notifications, otherwise absent),  
'List of COV Notifications' = (values appropriate to some or all of  
the properties subscribed to)

4. MAKE (a change to a monitored property, P1, that should cause a COVM notification)

5. WAIT N seconds, where  $L > N >$  the resolution of the IUT's COVM lifetime timer

6. IF (the subscription was for confirmed notifications) THEN

BEFORE **Notification Fail Time**

RECEIVE ConfirmedCOVNotificationMultiple-Request,  
'Subscriber Process Identifier' = ID1,  
'Initiating Device Identifier' = IUT,  
'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),  
'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
Notifications, otherwise absent)  
'List of COV Notifications' = (a list of values of length 1 indicating P1's value)

TRANSMIT BACnet-SimpleACK-PDU

ELSE

BEFORE **Notification Fail Time**

RECEIVE UnconfirmedCOVNotificationMultiple-Request,  
'Subscriber Process Identifier' = ID1,  
'Initiating Device Identifier' = IUT,  
'Time Remaining' = (TR:  $0 < TR < (L - N)$ ),  
'Timestamp' = (an appropriate timestamp, if subscribed to timestamped  
Notifications, otherwise absent)  
'List of COV Notifications' = (values appropriate to the properties subscribed to)

7. WAIT L seconds

8. MAKE (a change to a monitored property that would cause a COVM notification if there were an active subscription)

9. CHECK (verify that the IUT did not transmit a COVM notification message for the modified property)

10. VERIFY Active\_COV\_Multiple\_Subscriptions = (a list which does not contain a COVM context for ID1)

#### 9.42.2 Negative SubscribeCOVPropertyMultiple Service Execution Tests

##### 9.42.2.1 The Monitored Object Does Not Support COVM Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored object does not support COVM notifications.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where O1 does not support COVM. All requested subscriptions before O1 are selected such that they would succeed if O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall not support COVM notification for any of its properties. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

- TRANSMIT SubscribeCOVPropertyMultiple-Request  
'Subscriber Process Identifier' = (any valid process identifier),  
'Issue Confirmed Notifications' = TRUE | FALSE,  
'Lifetime' = (any valid lifetime),  
'Max Notification Delay' = (any valid value smaller than the lifetime),

'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property requested from O1)

2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = OBJECT,
    - 'Error Code' = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

#### 9.42.2.2 The Monitored Property Does Not Support COVM Notification

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property does not support COVM notifications.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where P1 does not support COVM. All requested subscriptions before P1 are selected such that they would succeed if P1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall support COVM notification for any of its properties. If the IUT does not support objects for which COVM is supported for only a subset of the properties, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = NOT\_COV\_PROPERTY

#### 9.42.2.3 Monitored Object Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored object does not exist.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where O1 does not exist but would support COVM for P1 if it did. All requested subscriptions before O1 are selected such that they would succeed if O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall be of a type for which the IUT supports COVM notifications for property P1. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property requested from O1)



2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = OBJECT,
    - 'Error Code' = UNKNOWN\_OBJECT

#### 9.42.2.4 Monitored Property Does Not Exist

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property does not exist.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1 in object O1 where P1 does not exist in O1. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The object, O1, shall exist, shall not contain P1, and be of a type for which the IUT supports COVM notifications. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to)
2. RECEIVE BACnet-Error-PDU,
  - 'First-Failed-Subscription' = {
    - 'Monitored Object Identifier' = O1,
    - 'Monitored Property Reference' = P1,
    - 'Error Class' = PROPERTY,
    - 'Error Code' = UNKNOWN\_PROPERTY

#### 9.42.2.5 Array Index Provided But Property is Not an Array

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property is not an array, but an array index is provided.

Test Concept: A subscription for COVM notifications is made which includes a request for property P1, with an array index, in object O1 where the IUT supports COVM for P1 in O1 but P1 is not an array. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: The property P1 shall be one which supports COVM and is not an array. If the IUT cannot be configured in this manner, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid lifetime),
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to. An array index shall be included in the entry for P1)



2. RECEIVE BACnet-Error-PDU,  
'First-Failed-Subscription' = {  
'Monitored Object Identifier' = O1,  
'Monitored Property Reference' = P1,  
'Error Class' = PROPERTY,  
'Error Code' = PROPERTY\_IS\_NOT\_AN\_ARRAY  
}

#### 9.42.2.6 Array Index Provided Is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the monitored property is an array, but the provided array index is outside the range of the array.

Test Concept: A subscription for COVM notifications is made which includes a request for an array property P1, with an array index, in object O1 where the IUT supports COVM for P1. All requested subscriptions before P1 in O1 are selected such that they would succeed if P1 in O1 were not in the list. It is verified that the IUT returns the correct error indicating O1 and P1 as the first failed element encountered.

Configuration Requirements: If the IUT does not support COVM on any array properties, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
'Subscriber Process Identifier' = (any valid process identifier),  
'Issue Confirmed Notifications' = TRUE | FALSE,  
'Lifetime' = (any valid lifetime),  
'Max Notification Delay' = (any valid value smaller than the lifetime),  
'List of COV Subscription Specifications' = (a list of subscriptions for properties in O1 and optionally other objects with P1 being the first property which cannot be subscribed to. The array index included in the entry for P1 shall be larger than the number of entries in P1)
2. RECEIVE BACnet-Error-PDU,  
'First-Failed-Subscription' = {  
'Monitored Object Identifier' = O1,  
'Monitored Property Reference' = P1,  
'Error Class' = PROPERTY,  
'Error Code' = INVALID\_ARRAY\_INDEX  
}

#### 9.42.2.7 No Space to Add List Element

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when there is no space for a subscription.

Test Concept: Repeatedly subscribe to the same object each time with a different Process Identifier until the device runs out of resources and returns the appropriate error.

Configuration Requirements: If the device cannot be configured such that the maximum number of subscriptions the IUT can accept is less than 10000, then this test shall be skipped.

Test Steps:

REPEAT PID = (1 through the maximum number of subscriptions the IUT can accept plus 1 or until the IUT returns an Error-PDU) DO {

1. TRANSMIT SubscribeCOVPropertyMultiple-Request  
'Subscriber Process Identifier' = PID,  
'Issue Confirmed Notifications' = TRUE | FALSE,  
'Lifetime' = (any valid lifetime large enough to complete the test),  
'Max Notification Delay' = (any valid value smaller than the lifetime),  
'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACNET-SimpleACK-PDU  
| (BACnet-Error-PDU,

```

'Error Class' = RESOURCES,
'Error Code' = NO_SPACE_TO_ADD_LIST_ELEMENT)
}

```

#### 9.42.2.8 The Lifetime Parameter is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Lifetime parameter is out of range.

Configuration Requirements: If the device supports lifetimes across the full range of valid lifetimes, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (a value larger than that supported by the IUT)
  - 'Max Notification Delay' = (any valid value smaller than the lifetime),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 9.42.2.9 The Max Notification Delay Parameter is Out Of Range

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Max Notification Delay parameter is out of range.

Configuration Requirements: If the device supports Max Notification Delays across the full range of valid values, then this test shall be skipped.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any valid value large enough to complete the test),
  - 'Max Notification Delay' = (a value larger than supported by the IUT),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE

#### 9.42.2.10 The Max Notification Delay is Greater Than the Lifetime

Purpose: To verify that the IUT correctly responds to a SubscribeCOVPropertyMultiple request to establish a subscription when the Max Notification Delay parameter is greater than the Lifetime parameter.

Test Steps:

1. TRANSMIT SubscribeCOVPropertyMultiple-Request
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (a value supported by the IUT but within the normal range of Max Notification Delay)
  - 'Max Notification Delay' = (a value greater than the lifetime),
  - 'List of COV Subscription Specifications' = (any valid list of subscriptions)
2. RECEIVE BACnet-Error-PDU,
  - 'Error Class' = SERVICES,
  - 'Error Code' = VALUE\_OUT\_OF\_RANGE

## 10. NETWORK LAYER PROTOCOL TESTS

### 10.1 General Network Layer Tests

#### 10.1.1 Processing Application Layer Messages Originating from Remote Networks

Purpose: To verify that the IUT can respond to requests that originate from a remote network.

Test Concept: The TD transmits a ReadProperty-Request message that contains network layer information indicating that it originated from a remote network. The response from the IUT shall include correct DNET and DADR information so that the message can reach the original requester. The MAC layer destination address in the response can be either a local broadcast, indicating that the IUT does not know the address of the router, or the MAC address of the TD.

Test Steps:

##### 1. TRANSMIT

DESTINATION = IUT,  
SOURCE = TD,  
SNET = (any network number that is not the local network),  
SADR = (any valid MAC address consistent with the source network),  
ReadProperty-Request,  
'Object Identifier' = (any supported object),  
'Property Identifier' = (any required property of the specified object)

##### 2. RECEIVE

DESTINATION = LOCAL BROADCAST | TD,  
SOURCE = IUT,  
DNET = (the SNET specified in step 1),  
DADR = (the SADR specified in step 1),  
Hop Count = 255,  
ReadProperty-ACK,  
'Object Identifier' = (the object specified in step 1),  
'Property Identifier' = (the property specified in step 1),  
'Property Value' = (any valid value for this property)

#### 10.1.2 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)

}

## 10.2 Router Functionality Tests

This clause defines the tests necessary to demonstrate BACnet router functionality. The tests assume that the router has two ports. Port 1 is directly connected to Network 1 and Port 2 is directly connected to Network 2. Routers with more than two ports shall be tested using these procedures for each possible combination of two ports. The logical configuration of the internetwork used for these tests is shown in Figure 10-1. The test descriptions in this clause assume that the TD can connect simultaneously to Networks 1 and 2 and mimic all of the other devices. The connection to Network 1 is referred to as "Port A" and the connection to Network 2 as "Port B." An acceptable alternative is to construct an internetwork with real devices as indicated. Logical networks 3 and 5 shall use different LAN technologies, both of which are different from networks 1 and 2 in order to ensure that the router can support remote networks with different size MAC addresses.

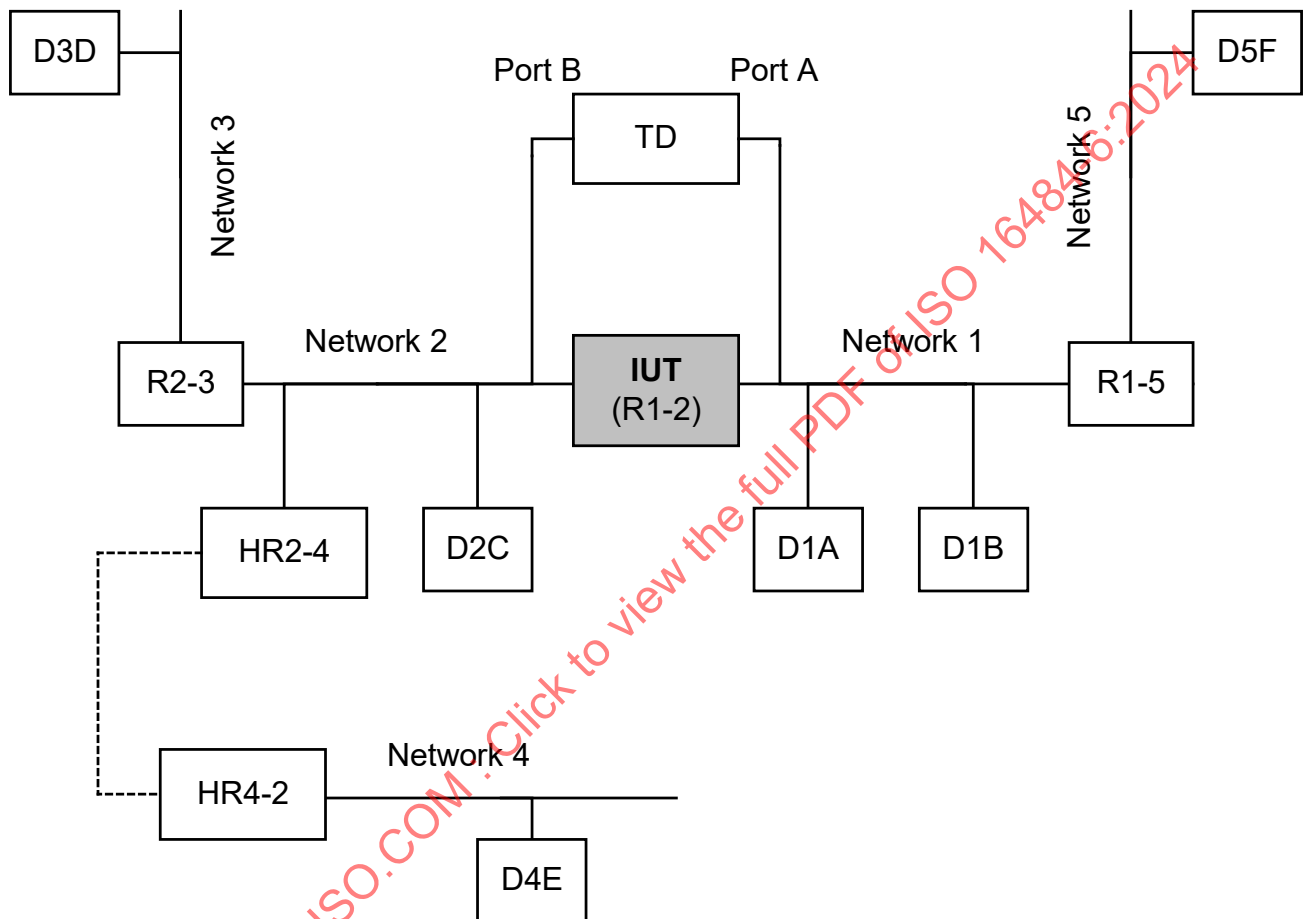


Figure 10-1. Logical internetwork configuration for router functionality tests

The logical devices included in the internetwork are:

- IUT: Implementation under test, a router between Networks 1 and 2
- D1A: device on Network 1
- D1B: device on Network 1
- D2C: device on Network 2
- D3D: device on Network 3
- D4E: device on Network 4
- D5F: device on Network 5
- R1-5: router between Network 1 and Network 5
- R2-3: router between Network 2 and Network 3
- HR2-4: half-router directly connected to Network 2 that can make a PTP connection to half-router H4-2 connected to Network 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.

The router functionality tests shall be conducted in the order they are defined in this standard. In some cases successful completion of a test case requires that the IUT begin in the final state from the previous test.

### 10.2.1 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 PORT A,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 2
3. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 1

### 10.2.2 Processing Network Layer Messages

This clause defines tests to verify that the IUT correctly processes network layer messages.

#### 10.2.2.1 Forward I-Am-Router-To-Network

Purpose: To verify that the IUT will broadcast an appropriate I-Am-Router-To-Network message when it receives an I-Am-Router-To-Network message from another router.

Test Concept: The TD simulates the start up of routers R2-3 and R1-5 by transmitting I-Am-Router-To-Network messages. The IUT shall update its routing table (verified in 10.2.2.6.1) and transmit I-Am-Router-To-Network messages on all ports except for the one on which the I-Am-Router-to-Network-Message was received.

Test Steps:

1. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 3
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 3
3. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 5
4. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 5

#### 10.2.2.2 Execute Who-Is-Router-To-Network

##### 10.2.2.2.1 No Specified Network Number

Purpose: To verify that the IUT will broadcast an I-Am-Router-To-Network message listing all downstream networks when it receives a Who-Is-Router-To-Network message with no specified network number.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-to-Network
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 2, 3 | 3, 2
3. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D2C,  
Who-Is-Router-to-Network
4. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 1, 5 | 5, 1

#### 10.2.2.2.2 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 PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network,  
Network Number = 2
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 2
3. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D2C  
Who-Is-Router-To-Network,  
Network Number = 5
4. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
I-Am-Router-To-Network,  
Network Numbers = 5

#### 10.2.2.2.3 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 PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network,  
Network Number = 1
2. Wait **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)
4. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,

SOURCE = D2C,  
Who-Is-Router-To-Network,  
Network Number = 3

5. Wait **Internal Processing Fail Time**
6. CHECK (verify that the IUT does not respond)

#### 10.2.2.2.4 An Unknown and Unreachable 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 attempt to locate the network.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network,  
Network Number = 4
2. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
SA = IUT,  
SNET = 1,  
SADR = D1A,  
Who-Is-Router-To-Network,  
Network Number = 4
3. CHECK (verify that no I-Am-Router-To-Network is transmitted on Port A)
4. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D2C,  
Who-Is-Router-To-Network,  
Network Number = 4
5. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
SNET = 2,  
SADR = D2C,  
Who-Is-Router-To-Network,  
Network Number = 4
6. CHECK (verify that no I-Am-Router-To-Network is transmitted on Port B)

#### 10.2.2.2.5 An Unknown Network is Discovered

Purpose: To verify that after searching for and discovering the path to an unknown network the IUT transmits the appropriate I-Am-Router-To-Network message.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network,  
Network Number = 6
2. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
SA = IUT,  
SNET = 1,  
SADR = D1A,  
Who-Is-Router-To-Network,  
Network Number = 6
3. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,



I-Am-Router-To-Network,  
Network Numbers = 6

4. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
I-Am-Router-To-Network,  
Network Numbers = 6

#### 10.2.2.2.6 Forwarding a Who-Is-Router-To-Network from a Remote Network

Purpose: To verify that the IUT will forward a Who-Is-Router-To-Network message if it receives a Who-Is-Router-To-Network message specifying an unknown network number and containing network layer source addressing information.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R1-5,  
SNET = 5,  
SADR = D5F,  
Who-Is-Router-To-Network,  
Network Number = 4
2. RECEIVE PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT  
SNET = 5,  
SADR = D5F,  
Who-Is-Router-To-Network,  
Network Number = 4
3. CHECK (verify that no I-Am-Router-To-Network is transmitted on Port A)
4. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
SNET = 3,  
SADR = D3D,  
Who-Is-Router-To-Network,  
Network Number = 4
5. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
SNET = 3,  
SADR = D3D,  
Who-Is-Router-To-Network,  
Network Number = 4
6. CHECK (verify that no I-Am-Router-To-Network is transmitted on Port B)

#### 10.2.2.3 Forward I-Could-Be-Router-To-Network

Purpose: To verify that the IUT will forward a received I-Could-Be-Router-To-Network message to the intended recipient.

Test Steps:

1. TRANSMIT PORT B,  
DA = IUT,  
SOURCE = HR2-4,  
DNET = 1,  
DADR = D1A,  
Hop Count = 255,  
I-Could-Be-Router-To-Network,  
Network Number = 4,  
Performance Index = 6

2. RECEIVE PORT A,  
DA = D1A,  
SA = IUT,  
SNET = 2,  
SADR = HR2-4,  
I-Could-Be-Router-To-Network,  
Network Number = 4,  
Performance Index = 6

#### 10.2.2.4 Router-Busy-To-Network

BACnet Clause Reference: 6.6.3.6

##### 10.2.2.4.1 Forwarding Router-Busy-to-Network Information for Specific DNETs

Purpose: To verify that the IUT correctly forwards information indicating that specific DNETs are temporarily unreachable because of traffic congestion.

Test Steps:

1. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Busy-To-Network,  
Network Numbers = 6
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Busy-To-Network,  
Network Numbers = 6

##### 10.2.2.4.2 Forwarding Router-Busy-To-Network Information for all DNETs

Purpose: To verify that the IUT correctly forwards information indicating that all DNETs reachable through a particular router are temporarily unreachable because of traffic congestion.

Test Steps:

1. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Busy-To-Network
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Busy-To-Network,  
Network Numbers = 3, 6 | 6, 3 | (absent)

##### 10.2.2.4.3 Receiving Messages for a Busy Router

Purpose: To verify that the IUT rejects a message destined for a busy router.

Test Steps:

1. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Busy-To-Network,  
Network Numbers = 3
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Busy-To-Network,  
Network Numbers = 3

3. TRANSMIT PORT A,  
     DA = IUT,  
     SOURCE = D1A,  
     DNET = 3,  
     DADR = D3D,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet standard object),  
     'Property Identifier' = (any required property of the specified object)
4. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Reject Reason = 2 (router busy),  
     DNET = 3

#### 10.2.2.4.4 Timeout

Purpose: To verify that the IUT restores the availability status of DNETs after the busy timer expires.

Test Steps:

1. TRANSMIT PORT B,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = R2-3,  
     Router-Busy-To-Network,  
     Network Numbers = 3
2. RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     Router-Busy-To-Network,  
     Network Numbers = 3
3. WAIT (30 seconds)
4. TRANSMIT PORT A,  
     DA = IUT,  
     SOURCE = D1A,  
     DNET = 3,  
     DADR = D3D,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet standard object),  
     'Property Identifier' = (any required property of the specified object)
5. RECEIVE PORT B,  
     DA = R2-3,  
     SOURCE = IUT,  
     DNET = 3,  
     DADR = D3D,  
     Hop Count = (any integer  $x$ :  $0 < x < 255$ ),  
     ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 4),  
     'Property Identifier' = (the property identifier used in step 4)

#### 10.2.2.5 Execute Router-Available-To-Network

##### 10.2.2.5.1 Restoring Specific DNETs

Purpose: To verify that the IUT updates its network availability information when a Router-Available-To-Network message conveying specific DNETs is received.

Test Steps:

1. TRANSMIT PORT B,

- DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Busy-To-Network
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Busy-To-Network,  
Network Numbers = 3, 6 | 6, 3
  3. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Available-To-Network,  
Network Numbers = 3
  4. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Available-To-Network,  
Network Numbers = 3
  5. TRANSMIT PORT A,  
DESTINATION = IUT,  
SOURCE = D1A,  
DNET = 3,  
DADR = D3D,  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any BACnet standard object),  
'Property Identifier' = (any required property of the specified object)
  6. RECEIVE PORT B,  
DESTINATION = R2-3,  
SOURCE = IUT,  
DNET = 3,  
DADR = D3D,  
Hop Count = (any integer  $x$ :  $0 < x < 255$ ),  
ReadProperty-Request,  
'Object Identifier' = (the object identifier used in step 5),  
'Property Identifier' = (the property identifier used in step 5)
  7. TRANSMIT PORT A,  
DESTINATION = IUT,  
SOURCE = D1A,  
DNET = 6,  
DADR = (any valid device address),  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any BACnet standard object),  
'Property Identifier' = (any required property of the specified object)
  8. RECEIVE PORT A,  
DESTINATION = D1A,  
SOURCE = IUT,  
Reject-Message-To-Network,  
Reject Reason = 2 (router busy),  
DNET = 6

#### 10.2.2.5.2 Restoring All DNETs

Purpose: To verify that the IUT updates its network availability information when a Router-Available-To-Network message conveying no DNETs is received.

Test Steps:

1. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,

- SOURCE = R2-3,  
Router-Busy-To-Network
2. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Busy-To-Network,  
Network Numbers = 3, 6 | 6, 3
  3. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
Router-Available-To-Network
  4. RECEIVE PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Router-Available-To-Network,  
Network Numbers = 3, 6 | 6, 3
  5. TRANSMIT PORT A,  
DA = IUT,  
SOURCE = D1A,  
DNET = 3,  
DADR = D3D,  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any BACnet standard object),  
'Property Identifier' = (any required property of the specified object)
  6. RECEIVE PORT B,  
DA = R2-3,  
SOURCE = IUT,  
DNET = 3,  
DADR = D3D,  
Hop Count = (any integer  $x$ :  $0 < x < 255$ ),  
ReadProperty-Request,  
'Object Identifier' = (the object identifier used in step 5),  
'Property Identifier' = (the property identifier used in step 5)
  7. TRANSMIT PORT A,  
DA = IUT,  
SOURCE = D1A,  
DNET = 6,  
DADR = (any valid device address),  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any BACnet standard object),  
'Property Identifier' = (any required property of the specified object)
  8. RECEIVE PORT B,  
DA = R2-3,  
SOURCE = IUT,  
DNET = 6,  
DADR = (the address used in step 6),  
Hop Count = (any integer  $x$ :  $0 < x < 255$ ),  
ReadProperty-Request,  
'Object Identifier' = (the object identifier used in step 7),  
'Property Identifier' = (the property identifier used in step 7)

#### 10.2.2.6 Execute Initialize-Routing-Table

##### 10.2.2.6.1 Query Routing Table

Purpose: To verify that the IUT will correctly respond to an Initialize-Routing-Table message with the Number of Ports field containing the value zero.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Initialize-Routing-Table,  
     Number of Ports = 0
2. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Initialize-Routing-Table-Ack,  
     Number of Ports = 5,  
     Connected DNET = 1,  
     Port ID = 1,  
     Port Info = (any valid port information),  
     Connected DNET = 2,  
     Port ID = 2,  
     Port Info = (any valid port information),  
     Connected DNET = 3,  
     Port ID = 2,  
     Port Info = (any valid port information),  
     Connected DNET = 5,  
     Port ID = 1,  
     Port Info = (any valid port information),  
     Connected DNET = 6,  
     Port ID = 2,  
     Port Info = (any valid port information)

Notes to Tester: The DNET table entries may be in any order.

#### 10.2.2.6.2 Add Entries to a Routing Table

Purpose: To verify that the IUT will correctly respond to an Initialize-Routing-Table message with the Number of Ports field containing a non-zero value and change its routing table.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Initialize-Routing-Table,  
         Number of Ports = 1,  
         Connected DNET = 1234,  
         Port ID = 1,  
         Port Info Length = 0
2. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Initialize-Routing-Table-Ack
3. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Initialize-Routing-Table,  
         Number of Ports = 0,
4. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Initialize-Routing-Table-Ack,  
         Number of Ports = 6,  
         Connected DNET = 1,  
         Port ID = 1,

Port Info = (any valid port information),  
 Connected DNET = 2,  
 Port ID = 2,  
 Port Info = (any valid port information),  
 Connected DNET = 3,  
 Port ID = 2,  
 Port Info = (any valid port information),  
 Connected DNET = 5,  
 Port ID = 1,  
 Port Info = (any valid port information),  
 Connected DNET = 6,  
 Port ID = 2,  
 Port Info = (any valid port information),  
 Connected DNET = 1234,  
 Port ID = 1,  
 Port Info = (any valid port information)

Notes to Tester: The DNET table entries may be in any order.

#### 10.2.2.6.3 Purge Entries in a Routing Table

Purpose: To verify that the IUT will correctly respond to an Initialize-Routing-Table message with a Port ID field of zero.

Test Steps:

1. TRANSMIT PORT A,  
 DESTINATION = IUT,  
 SOURCE = D1A,  
 Initialize-Routing-Table,  
 Number of Ports = 2,  
 Connected DNET = 1234,  
 Port ID = 0,  
 Port Info Length = 0,  
 Connected DNET = 6,  
 Port ID = 0,  
 Port Info Length = 0
2. RECEIVE PORT A,  
 DESTINATION = D1A,  
 SOURCE = IUT,  
 Initialize-Routing-Table-Ack
3. TRANSMIT PORT A,  
 DESTINATION = IUT,  
 SOURCE = D1A,  
 Initialize-Routing-Table,  
 Number of Ports = 0,
4. RECEIVE PORT A,  
 DESTINATION = D1A,  
 SOURCE = IUT,  
 Initialize-Routing-Table-Ack,  
 Number of ports = 4,  
 Connected DNET = 1,  
 Port ID = 1,  
 Port Info = (any valid port information),  
 Connected DNET = 2,  
 Port ID = 2,  
 Port Info = (any valid port information),  
 Connected DNET = 3,  
 Port ID = 2,  
 Port Info = (any valid port information),  
 Connected DNET = 5,



Port ID = 1,  
 Port Info = (any valid port information)

Notes to Tester: The DNET table entries may be in any order.

#### 10.2.2.7 Reject-Message-To-Network

This clause tests some of the possible circumstances where a message should be rejected by the network layer. Rejections caused by busy routers are covered in 10.2.2.4.

##### 10.2.2.7.1 Unknown Network

Purpose: To verify the IUT will reject a message addressed to a device on an unknown and unreachable DNET.

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SOURCE = D1A,  
     DNET = 9,  
     DADR = (any valid MAC address),  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     Who-Is-Router-To-Network,  
     Network Number = 9
3. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Reject Reason = 1, (unknown destination network)  
     DNET = 9

##### 10.2.2.7.2 Unknown Network Layer Message Type

Purpose: To verify that the IUT will reject a network layer message with an unknown message type in the range of message types reserved for use by ASHRAE.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Message Type = (any value in the range reserved for use by ASHRAE that is undefined in the protocol revision claimed by the device)
2. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Rejection Reason = 3 (unknown network layer message type),  
     DNET = (any value)

##### 10.2.2.7.3 Unknown Network Layer Message Type For Someone Else

Purpose: This test case verifies that the IUT will not reject a network layer message with an unknown message type when it is destined elsewhere. This test shall not be run if the IUT does not have a Protocol\_Revision property or its value is less than 4.

Test Steps:

1. TRANSMIT PORT A, DA = IUT, SA = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     Message Type = (any value in the range reserved for use by ASHRAE)
2. RECEIVE Port B, DA = D2C, SA = IUT,  
     SNET = 1,  
     SADR = D1A,  
     Message Type = (value from step 1)
3. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     Message Type = (any value in the range reserved for use by ASHRAE)
4. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any value greater than 1 and less than 255),  
     Message Type = (value from step 3)
5. TRANSMIT PORT A, DA = IUT, SA = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     Message Type = (any value in the range available for vendor proprietary messages),  
     Vendor ID = (any value, when paired with Message Type, that is not supported by the IUT)
6. RECEIVE Port B, DA = D2C, SA = IUT,  
     SNET = 1,  
     SADR = D1A,  
     Message Type = (value from step 5),  
     Vendor ID = (value from step 5)
7. TRANSMIT PORT A, DA = LOCAL BROADCAST, SA = D1A,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     Message Type = (any value in the range available for vendor proprietary messages),  
     Vendor ID = (any value, when paired with Message Type, that is not supported by the IUT)
8. RECEIVE PORT B, DA = LOCAL BROADCAST, SA = IUT,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any value greater than 1 and less than 255),  
     Message Type = (value from step 7),  
     Vendor ID = (value from step 7)

### 10.2.3 Routing of Unicast APDUs

The tests in this clause verify that the IUT correctly routes messages that use unicast destination addresses.

#### 10.2.3.1 Ignore Local Message Traffic

Purpose: To verify that the IUT will ignore local unicast message traffic.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = D1B,  
     SOURCE = D1A,

- BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (any object identifier),  
 'Property Identifier' = (any property of the specified object)
2. CHECK (verify that this message is not forwarded to Network 2)

### 10.2.3.2 Route Message from a Local Device to a Local Device

Purpose: To verify that the IUT can route a maximum sized NPDU unicast message from a local device on Network 1 to a device on Network 2.

Test Concept: A message is sent from Network 1 destined for Network 2 via the IUT, and router functionality is verified by observing the message on Network 2. The sequence is repeated in the opposite direction to verify the IUT can also route messages from Network 2 to Network 1. The messages may have an NPDU length, L, such that L equals the Maximum NPDU length as defined in Table 6-1 for the smaller data link.

Test Steps:

1. TRANSMIT PORT A,  
 DA = IUT,  
 SA = D1A,  
 DNET = 2,  
 DADR = D2C,  
 Hop Count = 255,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = WriteProperty-Request,  
 'Object Identifier' = (O1 = any object identifier),  
 'Property Identifier' = (P1 = any property of the specified object with a CharacterString datatype),  
 'Property Value' = (V = CharacterString with a length such that the NPDU length = L)
2. RECEIVE PORT B,  
 DA = D2C,  
 SA = IUT,  
 SNET = 1,  
 SADR = D1A,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = WriteProperty-Request,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1,  
 'Property Value' = V
3. TRANSMIT PORT B,  
 DA = IUT,  
 SA = D2C,  
 DNET = 1,  
 DADR = D1A,  
 Hop Count = 255,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = WriteProperty-Request,  
 'Object Identifier' = (O1 = any object identifier),  
 'Property Identifier' = (P1 = any property of the specified object with a CharacterString datatype),  
 'Property Value' = (V = CharacterString with a length such that the NPDU length = L)
4. RECEIVE PORT A,  
 DA = D1A,  
 SA = IUT,  
 SNET = 2,  
 SADR = D2C,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = WriteProperty-Request,  
 'Object Identifier' = O1,  
 'Property Identifier' = P1,  
 'Property Value' = V

**10.2.3.3 Route Message from a Local Device to a Router**

Purpose: To verify that the IUT can route a unicast message from a local device on Network 1 to a router on the path to the destination network.

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SOURCE = D1A,  
     DNET = 3,  
     DADR = D3D,  
     Hop Count = 255,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DA = R2-3,  
     SA = IUT,  
     DNET = 3,  
     DADR = D3D,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x:  $0 < x < 255$ ),  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 1),  
     'Property Identifier' = (the property identifier used in step 1)

**10.2.3.4 Route Message from One Router to Another Router**

Purpose: To verify that the IUT can route a unicast message from a device on a remote network to a router on the path to the destination network.

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SA = R1-5,  
     DNET = 3,  
     DADR = D3D,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = 254,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DA = R2-3,  
     SA = IUT,  
     DNET = 3,  
     DADR = D3D,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = (and integer x:  $0 < x < 254$ ),  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 1),  
     'Property Identifier' = (the property identifier used in step 1)

**10.2.3.5 Route Message from a Router to a Local Device**

Purpose: To verify that the IUT can route a unicast message from a peer router to the destination device on a local network.

Test Steps:

1. TRANSMIT PORT A,  
     DA = IUT,  
     SA = R1-5,  
     DNET = 2,  
     DADR = D2C,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = 254,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DA = D2C,  
     SA = IUT,  
     SNET = 5,  
     SADR = D5F,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 1),  
     'Property Identifier' = (the property identifier used in step 1)

**10.2.3.6 Attempt to Locate Downstream Routers**

This clause tests the ability of the IUT to search for routers to an unknown destination network.

Configuration Requirements: The IUT shall be configured to know only about the directly-connected networks.

**10.2.3.6.1 Failed Attempt to Locate Router**

Purpose: To verify that the IUT will attempt to locate a router to an unknown network. Upon failing to locate such a router the IUT will transmit a Reject-Message-To-Network to the source device.

Configuration Requirements: The IUT shall be configured to know only about its directly connected networks.

TOMax: vendor defined time the router waits before sending a Reject-Message-to-Network request.

TOMin: vendor defined minimum time the router waits for a response to the Who-Is-Router-To-Network request.

Notes to Tester: The standard does not provide any guidance on how long a router should wait before declaring that the attempt to locate the next router failed. While there is no explicit minimum time, it is expected that routers wait long enough that the attempt would succeed if the next hop router responded immediately.

Test Steps:

1. TRANSMIT PORT A, DA = IUT,  
     SA = R1-5, DNET = 3,  
     DADR = D3D,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = 254,  
     BACnet-Confirmed-Request-PDU,  
     'Service Choice' = ReadProperty-Request,  
     'Object Identifier' = (any object identifier),  
     'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
     DESTINATION = LOCAL BROADCAST, SOURCE = IUT, Who-Is-Router-To-Network,

Network Number = 3

3. WAIT T<sub>omin</sub>

4. BEFORE T<sub>omax</sub>

RECEIVE PORT A, DA = R1-5,

SOURCE = IUT,

DNET = 5,

DADR = D5F,

Hop Count = 255,

Reject-Message-To-Network,

Rejection Reason = 1 (unknown destination network), DNET = 3

#### 10.2.3.6.2 Successful Attempt to Locate Router

Purpose: To verify that the IUT will attempt to locate a router to an unknown network. When successful it forwards the message to the next router on the path.

Configuration Requirements: The IUT shall be configured to know only about the directly-connected networks.

T<sub>omin</sub>: vendor defined minimum time the router waits for a response to the Who-Is-Router-To-Network request.

Notes to Tester: The standard does not provide any guidance on how long a router should wait before declaring that the attempt to locate the next router failed. While there is no explicit minimum time, it is expected that routers wait long enough that the attempt would succeed if the next hop router responded immediately.

Test Steps:

1. TRANSMIT PORT A, DA = IUT,

SA = R1-5,

DNET = 3,

DADR = D3D,

SNET = 5,

SADR = D5F,

Hop Count = 254,

BACnet-Confirmed-Request-PDU,

'Service Choice' = ReadProperty-Request,

'Object Identifier' = (any object identifier),

'Property Identifier' = (any property of the specified object)

2. RECEIVE PORT B,

DESTINATION = LOCAL BROADCAST, SOURCE = IUT, Who-Is-Router-To-Network,

Network Number = 3

3. WAIT any time less than T<sub>omin</sub>

4. TRANSMIT PORT B,

DESTINATION = LOCAL BROADCAST, SOURCE = R2-3, I-Am-Router-To-Network,

Network Numbers = 3

5. RECEIVE PORT B, SA = R2-3,

SA = IUT, DNET = 3, DADR = D3D,

SNET = 5,

SADR = D5F,

Hop Count = (any integer x: 0 < x < 254), BACnet-Confirmed-Request-PDU,

'Service Choice' = ReadProperty-Request,

'Object Identifier' = (the object identifier used in step 1), 'Property Identifier' = (the property identifier used in step

1)

#### 10.2.4 Routing of Broadcast APDUs

The tests in this clause verify that the IUT correctly routes messages that use broadcast destination addresses.

##### 10.2.4.1 Ignore Local Broadcast Message Traffic

Purpose: To verify that the IUT will ignore local broadcast message traffic that does not contain addressing for a remote network.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = D1A,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is
2. CHECK (verify that this message is not forwarded to Network 2)

#### 10.2.4.2 Global Broadcast from a Local Device

Purpose: To verify that the IUT properly forwards global broadcast messages that originate on a local network.

Test Steps:

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SOURCE = D1A,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is
2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x:  $0 < x < 255$ ),  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is

Notes to Tester: The message described in step 2 shall be transmitted from all ports of the router except for Port 1.

#### 10.2.4.3 Global Broadcast from a Remote Device

Purpose: To verify that the IUT properly forwards global broadcast messages that originate on a remote network.

Test Steps:

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SA = R1-5,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = 254,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is  
     'I-Am Device Identifier' = D5F,
2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 5,  
     SADR = D5F,  
     Hop Count = (any value x:  $0 < x < 254$ ),  
     BACnet-Unconfirmed-Request-PDU,



'Service Choice' = Who-Is

Notes to Tester: The message described in step 2 shall be transmitted from all ports of the router except for Port 1.

#### 10.2.4.4 Remote Broadcast from a Local Device to a Directly-Connected Network

Purpose: To verify that the IUT properly forwards remote broadcast messages that originate on a local network and are directed to another local network.

Test Steps:

1. TRANSMIT PORT B,  
DA = LOCAL BROADCAST,  
SA = D2C,  
DNET = 1,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. RECEIVE PORT A,  
DA = LOCAL BROADCAST,  
SA = IUT,  
SNET = 2,  
SADR = D2C,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

#### 10.2.4.5 Remote Broadcast from a Local Device to a Non-Directly-Connected Network

Purpose: To verify that the IUT properly forwards remote broadcast messages that originate on a local network and are directed to a remote network that is not directly connected.

Test Steps:

1. TRANSMIT PORT B,  
DA = LOCAL BROADCAST,  
SOURCE = D2C,  
DNET = 5,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. RECEIVE PORT A,  
DA = R1-5,  
SA = IUT,  
DNET = 5,  
DLEN = 0,  
SNET = 2,  
SADR = D2C,  
Hop Count = (any integer x:  $0 < x < 255$ ),  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is

#### 10.2.4.6 Remote Broadcast from a Remote Device to a Directly-Connected Network

Purpose: To verify that the IUT properly forwards remote broadcast messages that originate on a remote network and are directed to a directly-connected network.

Test Steps:

1. TRANSMIT PORT B,  
DA = IUT,  
SA = R2-3,

- DNET = 1,
- DLEN = 0,
- SNET = 3,
- SADR = D3D,
- Hop Count = 254,
- BACnet-Unconfirmed-Request-PDU,
- 'Service Choice' = Who-Is
- 2. RECEIVE PORT A,
- DA = LOCAL BROADCAST,
- SA = IUT,
- SNET = 3,
- SADR = D3D,
- BACnet-Unconfirmed-Request-PDU,
- 'Service Choice' = Who-Is

#### 10.2.4.7 Remote Broadcast from a Remote Device to a Remote Network

Purpose: To verify that the IUT properly forwards remote broadcast messages that originate on a remote network and are directed to a remote network.

Test Steps:

- 1. TRANSMIT PORT A,
- DA = IUT,
- SA = R1-5,
- DNET = 3,
- DLEN = 0,
- SNET = 5,
- SADR = D5F,
- Hop Count = 254,
- BACnet-Unconfirmed-Request-PDU,
- 'Service Choice' = Who-Is
- 2. RECEIVE PORT B,
- DA = R2-3,
- SA = IUT,
- DNET = 3,
- DLEN = 0,
- SNET = 5,
- SADR = D5F,
- Hop Count = (any integer  $x$ ,  $0 < x < 254$ ),
- BACnet-Unconfirmed-Request-PDU,
- 'Service Choice' = Who-Is

#### 10.2.4.8 Remote Broadcast that Should Be Ignored

Purpose: To verify that the IUT ignores broadcast messages that are intended for a remote network that is reachable through the same port that the message was received from.

Test Steps:

- 1. TRANSMIT PORT B,
- DA = LOCAL BROADCAST,
- SA = D2C,
- DNET = 3,
- DLEN = 0,
- Hop Count = 255,
- 'Service Choice' = Who-Is
- 2. CHECK (verify that the IUT does not forward this message)

#### 10.2.5 Hop Count Protection

Purpose: To verify that the IUT will discard a message if the Hop Count becomes zero.

Test Steps:

1. TRANSMIT PORT A,  
DA = IUT,  
SA = R1-5,  
DNET = 3,  
DLEN = 0,  
SNET = 5,  
SADR = D5F,  
Hop Count = 1,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
2. CHECK (verify that the IUT does not forward this message)
3. TRANSMIT PORT A,  
DA = IUT,  
SA = R1-5,  
DNET = 3,  
DLEN = 0,  
SNET = 5,  
SADR = D5F,  
Hop Count = 0,  
BACnet-Unconfirmed-Request-PDU,  
'Service Choice' = Who-Is
4. CHECK (verify that the IUT does not forward this message)

#### 10.2.6 Network Layer Priority

Purpose: To verify that the IUT can process and forward messages with all network priorities.

Test Steps:

1. TRANSMIT PORT A,  
DA = IUT,  
SA = D1A,  
Priority = B'00',  
DNET = 2,  
DADR = D2C,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (any object identifier),  
'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
SA = D2C,  
SA = IUT,  
Priority = B'00',  
SNET = 1,  
SDR = D1A,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (the object identifier used in step 1),  
'Property Identifier' = (the property identifier used in step 1)
3. TRANSMIT PORT A,  
DA = IUT,  
SA = D1A,  
Priority = B'01',  
DNET = 2,  
DADR = D2C,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,

- 'Object Identifier' = (any object identifier),  
 'Property Identifier' = (any property of the specified object)
4. RECEIVE PORT B,  
 DA = D2C,  
 SA = IUT,  
 Priority = B'01',  
 SNET = 1,  
 SDR = D1A,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (the object identifier used in step 3),  
 'Property Identifier' = (the property identifier used in step 3)
  5. TRANSMIT PORT A,  
 DA = IUT,  
 SA = D1A,  
 Priority = B'10',  
 DNET = 2,  
 DADR = D2C,  
 Hop Count = 255,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (any object identifier),  
 'Property Identifier' = (any property of the specified object)
  6. RECEIVE PORT B,  
 DA = D2C,  
 SA = IUT,  
 Priority = B'10',  
 SNET = 1,  
 SDR = D1A,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (the object identifier used in step 5),  
 'Property Identifier' = (the property identifier used in step 5)
  7. TRANSMIT PORT A,  
 DA = IUT,  
 SA = D1A,  
 Priority = B'11',  
 DNET = 2,  
 DADR = D2C,  
 Hop Count = 255,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (any object identifier),  
 'Property Identifier' = (any property of the specified object)
  8. RECEIVE PORT B,  
 DA = D2C,  
 SA = IUT,  
 Priority = B'11',  
 SNET = 1,  
 SDR = D1A,  
 BACnet-Confirmed-Request-PDU,  
 'Service Choice' = ReadProperty-Request,  
 'Object Identifier' = (the object identifier used in step 7),  
 'Property Identifier' = (the property identifier used in step 7)

#### 10.2.7 Initiates Network-Number-Is on Startup

Purpose: To verify that a router initiates Network-Number-Is on startup for each port with a known network number.

**Test Concept:** The IUT is reset and the tester verifies that the IUT broadcasts a Network-Number-Is message out each port. The vendor can specify a time, or physically observable event after reset, which marks the time at which IUT has completed its startup sequence, including the sending of the Network-Number-Is messages.

**Configuration Requirements:** The IUT is configured with a network number for each of its enabled ports. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

**Test Steps:**

1. MAKE (the IUT reset)
2. BEFORE the IUT has completed its startup sequence  
     REPEAT X = (for each enabled port) DO {  
         RECEIVE PORT X,  
         DESTINATION = LOCAL BROADCAST,  
         Network-Number-Is,  
         Network Number = (the configured Network Number for port X)  
     }

#### 10.2.8 Routers Execute What-Is-Network-Number

**Purpose:** To verify that a router responds to a What-Is-Network-Number request within 10 seconds.

**Test Concept:** A What-Is-Network-Number is broadcast on the local network and the tester verifies that the IUT responds with a Network-Number-Is message within 10 seconds.

**Configuration Requirements:** The IUT knows its network number, N1. If the IUT claims a protocol revision of less than 11, this test shall be skipped.

**Test Steps:**

1. TRANSMIT What-Is-Network-Number,  
     DESTINATION = LOCAL\_BROADCAST
2. BEFORE 10s + **Internal Processing Fail Time**  
     RECEIVE Network-Number-Is,  
     Network Number = (the configured value),  
     Configured =(any valid value)

#### 10.2.9 Data Attributes Forwarding Test

**Purpose:** To verify that routers which connect multiple network supporting data attributes, correctly route data attributes.

**Test Concept:** With the IUT configured as a router between two networks which supports data attributes (such as BACnet/SC), send to the router a message which needs to be routed to the next network, which contains data attributes. Verify that the message is correctly routed and the data attributes are included in the routed message.

**Configuration Requirements:** The IUT shall be configured as a router between 2 networks which support data attributes. If the IUT does not support this configuration this test shall be skipped.

**Test Steps:**

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SOURCE = D1A,  
     'Data Options' = (DATA\_OPTS: a valid list of options),  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is
2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,

'Data Options' = DATA\_OPTS,  
 DNET = GLOBAL BROADCAST,  
 DLEN = 0,  
 SNET = 1,  
 SADR = D1A,  
 Hop Count = (any integer x:  $0 < x < 255$ ),  
 BACnet-Unconfirmed-Request-PDU,  
 'Service Choice' = Who-Is

#### 10.2.10 Data Attributes Dropping Test

Purpose: To verify that the IUT drops data\_attributes from a received message before routing it to a network which does not support data\_attributes.

Test Concept: With the IUT configured as a router from a network which support data\_attributes (such as BACnet/SC) to a network which does not support data\_attributes (such as BACnet/IP), send to the router a message which needs to be routed to the next network, and which contains data\_attributes. Verify that message is correctly routed and the data\_attributes are silently dropped.

Test Steps:

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SOURCE = D1A,  
     'Data Options' = (DATA\_OPTS: a valid list of options),  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is
2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x:  $0 < x < 255$ ),  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is

#### 10.2.11 Secure Path Test

Purpose: To verify that a BACnet/SC to BACnet/SC router correctly conveys the 'Secure Path' header option on routed messages.

Test Concept: With the IUT configured as a router from BACnet/SC to BACnet/SC or another datalink which supports the 'Secure Path' header option, send to the router a message which needs to be routed to the next network, and which contains the Secure Path header option. Verify that Secure Path option is included in the routed message.

Configuration Requirements: The IUT is connected to 2 networks which support data attributes and are secure. If the IUT does not support this configuration then this test shall be skipped.

Test Steps:

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SOURCE = D1A,  
     'Data Options' = (DATA\_OPTS: Secure Path),  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,

2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,  
     'Data Options' = (DATA\_OPTS: Secure Path),  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x:  $0 < x < 255$ ),  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is

### 10.2.12 Insecure Path Test

**Purpose:** To verify that a non-BACnet/SC to BACnet/SC router does not include the 'Secure Path' header option on routed messages from the non-BACnet/SC network.

**Test Concept:** With the IUT configured as a router from BACnet/SC to a non-BACnet/SC which does not have data options, send to the router a message on the non-BACnet/SC network which needs to be routed to the BACnet/SC network. Verify that the Secure Path option is not included in the routed message.

**Configuration Requirements:** The IUT is connected to 2 networks with PORT A connected to a network type which does not support secure path, and PORT B connected to a network type which does. If the IUT does not support this configuration then this test shall be skipped.

**Test Steps:**

1. TRANSMIT PORT A,  
     DA = LOCAL BROADCAST,  
     SOURCE = D1A,  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     Hop Count = 255,  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is
2. RECEIVE PORT B,  
     DA = LOCAL BROADCAST,  
     SA = IUT,  
     -- 'Data Options' absent  
     DNET = GLOBAL BROADCAST,  
     DLEN = 0,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x:  $0 < x < 255$ ),  
     BACnet-Unconfirmed-Request-PDU,  
     'Service Choice' = Who-Is

### 10.3 Half-Router Functionality Tests

This clause defines the tests necessary to demonstrate BACnet half-router functionality. The tests assume that the half-router has two ports. Port 1 is directly connected to Network 1 and Port 2 is a PTP connection to Network 2. The logical configuration of the internetwork used for these tests is shown in Figure 10-2. The test descriptions in this clause assume that the TD can connect simultaneously to Networks 1 and 2 and mimic all of the other devices, including the peer half-router, HR2-1. The connection to Network 1 is referred to as "Port A" and the connection to Network 2 as "Port B." An acceptable alternative is to construct an internetwork with real devices as indicated. Logical networks 3 and 5 shall use different LAN technologies, both of which are different from networks 1 and 2 in order to ensure that the router can support remote networks with different size MAC addresses. The tests in this clause do not address PTP functionality except that the PTP connection is used. PTP functionality tests are covered in 12.2.



Configuration Requirements: The IUT shall be configured with routing tables indicating that Network 1 is directly connected to Port 1 and that a PTP connection could be, but is not yet, established to Network 2 through Port 2. The routing table shall contain no other entries.

The half-router functionality tests shall be conducted in the order they are defined in this standard. In some cases successful completion of a test case requires that the IUT begin in the final state from the previous test.

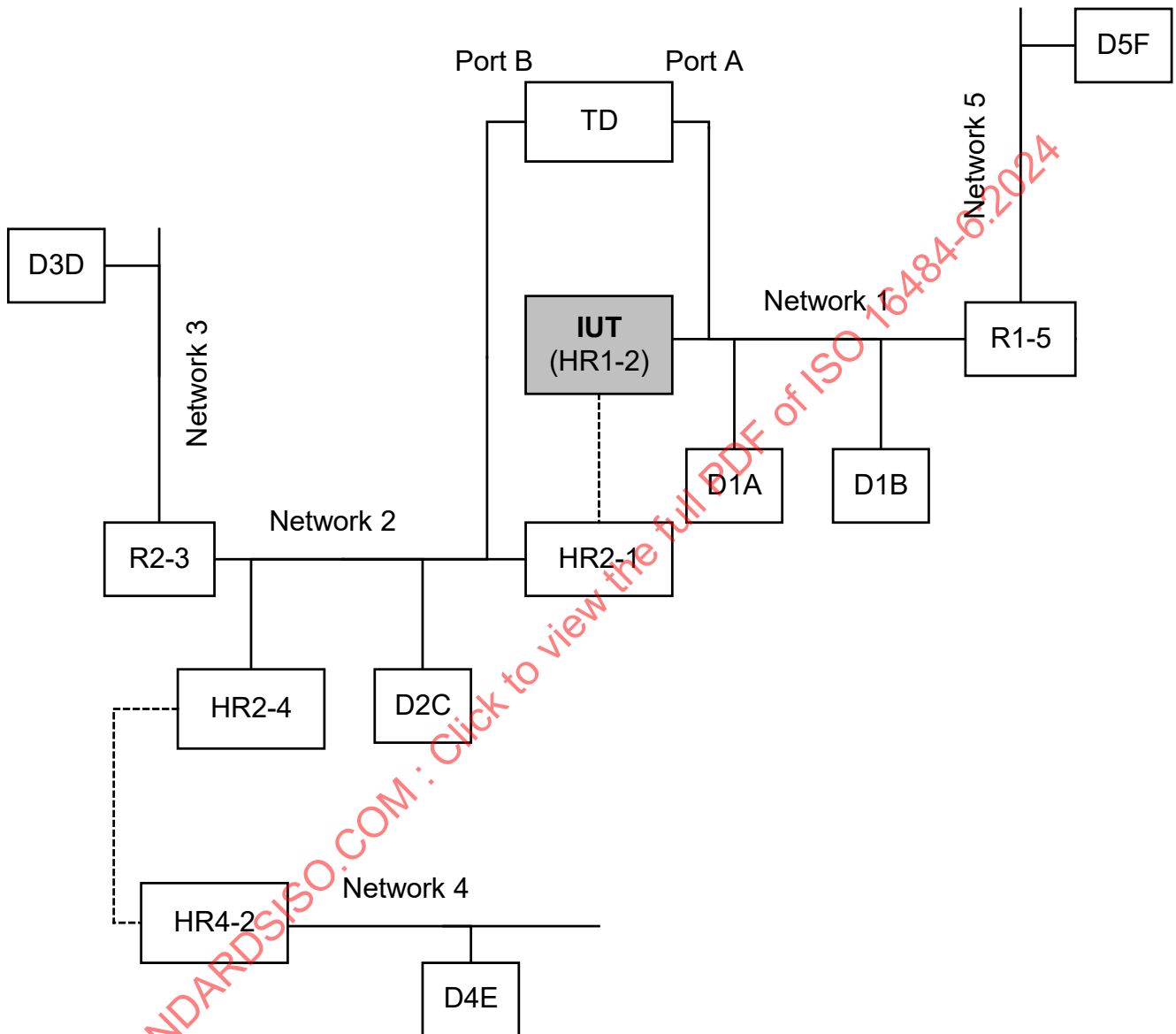


**Figure 10-2.** Logical internetwork configuration for half-router functionality tests

The logical devices included in the internetwork are:

- IUT: implementation under test, a half-router connected to Network 1 which can make a PTP connection to a half-router connected to Network 2
- D1A: device on Network 1
- D1B: device on Network 1
- D2C: device on Network 2
- D3D: device on Network 3
- D4E: device on Network 4
- D5F: device on Network 5
- R1-5: router between networks 1 and 5
- R2-3: router between networks 2 and 3

- HR2-1: half-router connected to Network 2 which can make a PTP connection to a half-router connected to Network 1 (the IUT)
- HR2-4: half-router connected to Network 2 that can make a PTP connection to a half-router H4-2 connected to Network 4.

### 10.3.1 Execute Who-Is-Router-To-Network

This clause defines tests to verify that the IUT correctly responds to Who-Is-Router-To-Network messages before a PTP connection is established.

#### 10.3.1.1 No Specified Network Number

Purpose: To verify that the IUT will not send any messages when it receives a Who-Is-Router-To-Network message with no specified network number.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network
2. CHECK (verify that the IUT does not respond)

#### 10.3.1.2 A Network Number is Specified that can be Reached Through a PTP Connection

Purpose: To verify that the IUT will transmit an I-Could-Be-Router-To-Network message when it receives a Who-Is-Router-To-Network message specifying a network number that is reachable through a PTP connection.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = D1A,  
Who-Is-Router-To-Network,  
Network Number = 2
2. RECEIVE PORT A,  
DESTINATION = D1A,  
SOURCE = IUT,  
I-Could-Be-Router-To-Network,  
Network Number = 2,  
Performance Index = (any valid performance index)

### 10.3.2 Reject Messages if no Connection is Established

Purpose: To verify that the IUT will transmit a Reject-Message-To-Network message if it is asked to route a message to a network to which no connection is established.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = IUT,  
SOURCE = D1A,  
DNET = 2,  
DADR = D2C,  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (any BACnet object),  
'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT A,  
DESTINATION = D1A,  
SOURCE = IUT,  
Reject-Message-To-Network,

Reject Reason = 1 (unknown destination network),  
 DNET = 2

### 10.3.3 Initiating Half-Router Procedure for Connection Establishment

Purpose: To verify that the IUT follows correct procedures when initiating a connection to a peer half-router.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Establish-Connection-To-Network,  
     DNET = 2,  
     Termination Time Value = 60
2. BEFORE (60 seconds)  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     I-Am-Router-To-Network,  
     Network Numbers = 2
3. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet object),  
     'Property Identifier' = (any property of the specified object)
4. RECEIVE PORT B,  
     DESTINATION = D2C,  
     SOURCE = IUT,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x: 0 < x < 255),  
     ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 3),  
     'Property Identifier' = (the property identifier used in step 3)

### 10.3.4 Automatic Disconnection Due to Expiration of the Activity Timer

Purpose: To verify that the IUT will terminate a connection with another half-router if it is not called upon to route any messages within the time interval specified by Termination Time Value.

Test Steps:

1. WAIT (90 seconds)
2. TRANSMIT PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = D1A,  
     Who-Is-Router-To-Network
3. CHECK (verify that the IUT does not respond)
4. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet object),  
     'Property Identifier' = (any property of the specified object)

5. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Reject Reason = 1 (unknown destination network),  
     DNET = 2

### 10.3.5 Answering Half-Router Procedure for Connection Establishment

Purpose: To verify that the IUT follows correct procedures to establish a connection initiated by a peer half-router.

Test Steps:

1. MAKE (the TD or a real half-router HR2-1 initiate a PTP connection to the IUT with a Termination Time Value of 0)
2. BEFORE (60 seconds)  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     I-Am-Router-To-Network,  
     Network Numbers = 2
3. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet object),  
     'Property Identifier' = (any property of the specified object)
4. RECEIVE PORT B,  
     DESTINATION = D2C,  
     SOURCE = IUT,  
     SNET = 1,  
     SADR = D1A,  
     Hop Count = (any integer x: 0 < x < 255),  
     ReadProperty-Request,  
     'Object Identifier' = (the object identifier used in step 3),  
     'Property Identifier' = (the property identifier used in step 3)

### 10.3.6 Periodic Broadcast of I-Am-Router-To-Network Messages

Purpose: To verify that the IUT will broadcast an I-Am-Router-To-Network message at five-minute intervals while the IUT has a connection established to another half-router.

Test Steps:

1. BEFORE (5 minutes) {  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     I-Am-Router-To-Network,  
     Network Numbers = 2  
     RECEIVE PORT B,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = HR2-1,  
     I-Am-Router-To-Network,  
     Network Numbers = 1}
2. BEFORE (5 minutes) {  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,

- I-Am-Router-To-Network,  
 Network Numbers = 2  
 RECEIVE PORT B,  
 DESTINATION = LOCAL BROADCAST,  
 SOURCE = HR2-1,  
 I-Am-Router-To-Network,  
 Network Numbers = 1}
3. BEFORE (5 minutes) {  
 RECEIVE PORT A,  
 DESTINATION = LOCAL BROADCAST,  
 SOURCE = IUT,  
 I-Am-Router-To-Network,  
 Network Numbers = 2  
 RECEIVE PORT B,  
 DESTINATION = LOCAL BROADCAST,  
 SOURCE = HR2-1,  
 I-Am-Router-To-Network,  
 Network Numbers = 1}

Notes to Tester: The I-Am-Router-To-Network messages can be received in either order. The time interval between groups of I-Am-Router-To-Network messages shall be approximately 5 minutes.

### 10.3.7 Disconnect-Connection-To-Network

Purpose: To verify that the IUT will follow correct procedures to terminate a connection with a peer half-router after receiving a Disconnect-Connection-To-Network message.

Test Steps:

1. TRANSMIT PORT A,  
 DESTINATION = IUT,  
 SOURCE = D1A,  
 Disconnect-Connection-To-Network,  
 DNET = 2
2. WAIT (60 seconds)
3. TRANSMIT PORT A,  
 DESTINATION = LOCAL BROADCAST,  
 SOURCE = D1A,  
 Who-Is-Router-To-Network
4. CHECK (verify that the IUT does not respond)
5. TRANSMIT PORT A,  
 DESTINATION = IUT,  
 SOURCE = D1A,  
 DNET = 2,  
 DADR = D2C,  
 Hop Count = 255,  
 ReadProperty-Request,  
 'Object Identifier' = (any BACnet object),  
 'Property Identifier' = (any property of the specified object)
6. RECEIVE PORT A,  
 DESTINATION = D1A,  
 SOURCE = IUT,  
 Reject-Message-To-Network,  
 Reject Reason = 1 (unknown destination network),  
 DNET = 2

### 10.3.8 Recovering from Duplicate Network Connections

Purpose: To verify that the IUT will terminate a connection to another half-router if an I-Am-Router-To-Network message is received that contains a DNET to one of the half-router's directly connected networks.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Establish-Connection-To-Network,  
     DNET = 2,  
     Termination Time Value = 60
2. BEFORE (60 seconds)  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,  
     I-Am-Router-To-Network,  
     Network Numbers = 2
3. TRANSMIT PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = R1-5,  
     I-Am-Router-To-Network,  
     Network Numbers = 2,5
4. WAIT (60 seconds)
5. TRANSMIT PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = D1A,  
     Who-Is-Router-To-Network
6. CHECK (verify that the IUT does not respond)
7. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     DNET = 2,  
     DADR = D2C,  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any BACnet object),  
     'Property Identifier' = (any property of the specified object)
8. RECEIVE PORT A,  
     DESTINATION = D1A,  
     SOURCE = IUT,  
     Reject-Message-To-Network,  
     Reject Reason = 1 (unknown destination network),  
     DNET = 2

### 10.3.9 Normal Routing Functions

Purpose: To verify that, in conjunction with its half-router peer, the IUT performs as a normal BACnet router after a PTP connection is established.

Test Concept: A PTP connection is established with the peer half-router and all of the tests prescribed in 10.2 are conducted considering the two half-routers to be a single router functioning as the IUT for those tests.

Test Steps:

1. TRANSMIT PORT A,  
     DESTINATION = IUT,  
     SOURCE = D1A,  
     Establish-Connection-To-Network,  
     DNET = 2,  
     Termination Time Value = 0
2. BEFORE (60 seconds)  
     RECEIVE PORT A,  
     DESTINATION = LOCAL BROADCAST,  
     SOURCE = IUT,

I-Am-Router-To-Network,  
Network Numbers = 2

3. MAKE (conduct all of the tests prescribed in 10.2)

Notes to Tester: The IUT shall respond to each test case as defined in 10.2.

#### 10.4 B/IP PAD Tests

This clause defines the tests necessary to demonstrate BACnet/Internet Protocol packet-assembler-disassembler (B/IP PAD) functionality, as defined in Annex H of the BACnet Standard. The logical configuration of the internetwork used for these tests is shown in Figure 10-3. The test descriptions assume that the TD can connect simultaneously to Network 1 and Network 2, mimic all of the other devices, and monitor the IP traffic between B/IP PADs. The connection to Network 1 is referred to as "Port A" and the connection to Network 2 as "Port B." An acceptable alternative is to construct an internetwork with real devices as indicated. Logical networks 3 and 5 shall use different LAN technologies, both of which are different from networks 1 and 2 in order to ensure that the router can support remote networks with different size MAC addresses

Configuration Requirements: The IUT shall be configured with routing tables indicating that Network 1 is directly connected to Port 1 and that the IP connection is connected to Port 2. The IUT may use the same network interface for both ports but they shall be distinct in the routing table. The IP connection to B/IP PAD2-1 shall be established prior to the start of the testing. The details of how to initialize the IUT are a local matter. It may require the use of a domain name server, which is not shown in Figure 10-3.

Test Steps: All of the tests defined in 10.2 shall be conducted.

Notes to Tester: Passing results are the same as defined in 10.2 with the addition that the messages between peer B/IP routers shall be monitored to ensure that they are UDP messages with the source and destination ports set to X'BAC0'.



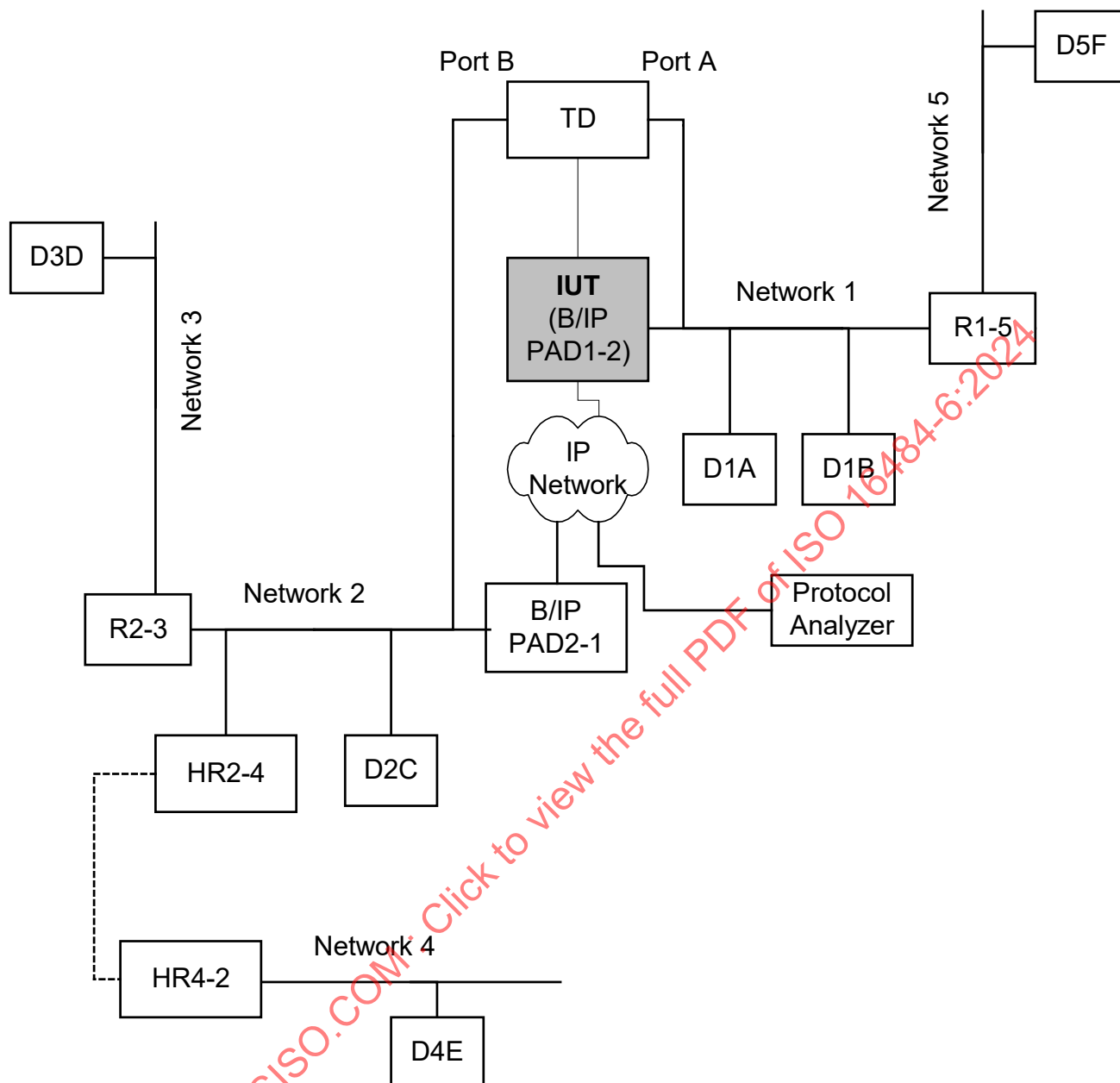


**Figure 10-3.** Logical internetwork configuration for B/IP router functionality tests

The logical devices included in the internetwork are:

- IUT: implementation under test, a B/IP router connected to Network 1 that is configured to recognize a peer B/IP router connected to Network 2
- D1A: device on Network 1
- D1B: device on Network 1
- D2C: device on Network 2
- D3D: device on Network 3
- D4E: device on Network 4
- D5F: device on Network 5
- R1-5: router between networks 1 and 5
- R2-3: router between networks 2 and 3
- HR2-4: half-router connected to Network 2 that can make a PTP connection to a half-router H4-2 connected to Network 4.

## 10.5 Initiating Network Layer Messages

This clause defines the tests necessary to demonstrate the ability to initiate BACnet network layer messages that are used to locate routers, manage router tables, and establish or terminate PTP connections. These tests are not restricted to BACnet routers. They shall be used to test any BACnet device that performs these functions.

### 10.5.1 Locating Routers

#### 10.5.1.1 Who-Is-Router-To-Network – General Query

Purpose: To verify that the IUT can transmit a correctly formatted Who-Is-Router-To-Network message that does not specify any network number.

Test Steps:

1. MAKE (the IUT initiate a Who-Is-Router-To-Network query with no DNET specified)
2. RECEIVE  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Who-Is-Router-To-Network

#### 10.5.1.2 Who-Is-Router-To-Network - Specific Network Number

Purpose: To verify that the IUT can transmit correctly formatted Who-Is-Router-To-Network message that specifies a network number.

Test Steps:

1. MAKE (the IUT initiate a Who-Is-Router-To-Network query with a DNET specified)
2. RECEIVE  
DESTINATION = LOCAL BROADCAST,  
SOURCE = IUT,  
Who-Is-Router-To-Network,  
DNET =(any valid DNET)

### 10.5.2 Managing Router Tables

#### 10.5.2.1 Query A Routing Table

**Purpose:** To verify that the IUT can transmit a correctly formatted Initialize-Routing-Table message that contains no port mappings.

Test Steps:

1. MAKE (the IUT initiate an Initialize-Routing-Table message with Number of Ports = 0)
2. RECEIVE  
DESTINATION = TD,  
SOURCE = IUT,  
Initialize-Routing-Table,  
Number of Ports = 0

#### 10.5.2.2 Change a Routing Table

Purpose: To verify that the IUT can transmit a correctly formatted Initialize-Routing-Table message that contains at least one port mapping.

Test Steps:

1. MAKE (the IUT initiate an Initialize-Routing-Table message with Number of Ports > 0)
2. RECEIVE  
DESTINATION = TD,  
SOURCE = IUT,  
Initialize-Routing-Table,

Number of Ports = (any integer > 0),  
 Connected DNET = (any valid DNET),  
 Port ID = (any valid port ID),  
 Port Info = (any valid port information)

Notes to Tester: The fields Connected DNET, Port ID, and Port Info shall be repeated the number of times indicated in the Number of Ports parameter.

### 10.5.2.3 Query A Router's Known Routes

Purpose: To verify that the IUT can query a router to determine which routes are accessible through it.

Test Concept: Make the IUT query the router, to determine their routes and verify that the IUT conveys the information to the user.

Configuration: The TD is configured as a router which does not support Network Port objects (Protocol Revision < 17).

Test Steps:

1. MAKE (the IUT initiate an Who-Is-Router-To-Network message)
2. RECEIVE Who-Is-Router-To-Network  
     DESTINATION = TD | LOCAL\_BROADCAST
3. TRANSMIT I-Am-Router-To-Network,  
     Network Numbers = (L: a valid list of networks)
4. CHECK (the IUT presents the router's known routes)

## 10.5.3 Initiating and Terminating PTP Connections

### 10.5.3.1 Establish-Connection-To-Network

Purpose: To verify that the IUT can transmit a correctly formatted Establish-Connection-To-Network message.

Test Steps:

1. MAKE (the IUT initiate an Establish-Connection-To-Network message)
2. RECEIVE  
     DESTINATION = TD,  
     SOURCE = IUT,  
     Establish-Connection-To-Network,  
     DNET = (any valid DNET),  
     Termination Time Value = (any integer >= 0)

### 10.5.3.2 Disconnect-Connection-To-Network

Purpose: To verify that the IUT can transmit a correctly formatted Disconnect-Connection-To-network message.

Test Steps:

1. MAKE (the IUT initiate an Disconnect-Connection-To-Network message)
2. RECEIVE  
     DESTINATION = TD,  
     SOURCE = IUT,  
     Disconnect-Connection-To-Network,  
     DNET = (any valid DNET)

## 10.6 Non-Router Functionality Tests

This subclause defines the tests necessary to demonstrate the portion of BACnet Network Layer functionality that is unique to non-router nodes. These tests verify that the node correctly ignores messages that are reserved for routers. If the IUT can only be configured as a BACnet router, then these tests shall be omitted.

The tests assume that the IUT is located on network DNET1 and the TD appears to be a router to network DNET2. The value DNET3 is assigned a unique network number. If the IUT can initiate requests, it will be configured to send those

requests to a device (D2A) on network DNET2. The IUT will also be expected to respond to requests from device D2A. The test descriptions assume that the TD will mimic device D2A.

Note to Tester: Clauses 6.5.1 and 6.5.3 indicate that there are only two ways for a non-router to transmit a request (on the local network and destined for a remote network), neither of which include network layer source routing information. If the IUT emits any NPDU with SNET/SADR fields during the tests in this subclause, then it shall fail.

Note to Tester: Clauses 6.6 and 6.6.3.3 define BACnet routers and the network layer services reserved for routers. If the IUT emits any I-Am-Router-To-Network or I-Could-Be-Router-To-Network NPDUs during the tests in this subclause (including during test 10.6.2), then it shall fail.

#### 10.6.1 Ignore Remote packets

Purpose: This test case verifies that the non-router IUT will quietly accept and discard packets destined for remote networks.

Test Concept: The TD transmits both broadcast and directed requests to the IUT with DNET (not equal to x'FFFF') and DADR in the Network Layer header. The IUT is required to silently drop the requests because it is not a router.

Test Steps:

1. TRANSMIT
  - DA = BROADCAST,
  - SA = TD,
  - DNET = DNET3,
  - DADR = BROADCAST,
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = Who-Is
2. WAIT **Internal Processing Fail Time**
3. CHECK (that the IUT did not send an I-Am)
4. TRANSMIT
  - DA = IUT,
  - SA = TD,
  - DNET = DNET3,
  - DADR = IUT,
  - Hop Count = 255,
  - BACnet-Confirmed-Request-PDU,
  - 'Service Choice' = ReadProperty-Request,
  - 'Object Identifier' = (O2, any BACnet standard object in IUT),
  - 'Property Identifier' = (P2, any required property of the specified object)
5. WAIT **Internal Processing Fail Time**
6. CHECK (that the IUT did not send a response to the ReadProperty)

Notes to Tester: Ensure that the packets transmitted in Step 1 and Step 4 will actually reach the IUT by transmitting them on the local network of the IUT. It is impossible to perform this test through a router.

#### 10.6.2 Ignore Who-Is-Router-To-Network

Purpose: This test case verifies that the non-router IUT will quietly accept and discard the Who-Is-Router-To-Network service.

Test Concept: The TD transmits both the general query and the specific network number query form of the Who-Is-Router-To-Network service. The IUT is required to silently drop the requests because it is not a router.

Test Steps:

1. TRANSMIT
  - DA = BROADCAST,
  - SA = TD,
  - Who-Is-Router-To-Network,
  - DNET = DNET2

2. **WAIT Internal Processing Fail Time**
3. **CHECK** (that the IUT did not send any packets in response to the Who-Is-Router-To-Network)
4. **TRANSMIT**  
     DA = BROADCAST,  
     SA = TD,  
     Who-Is-Router-To-Network
5. **WAIT Internal Processing Fail Time**
6. **CHECK** (that the IUT did not send any packets in response to the Who-Is-Router-To-Network)

### 10.6.3 Ignore Router Commands

Purpose: This test case verifies that the non-router IUT will either quietly accept and discard network layer router services or respond with a Reject-Message-To-Network.

Test Concept: The TD transmits the Initialize-Routing-Table, Establish-Connection-To-Network, and Disconnect-Connection-To-Network services.

Test Steps:

1. **TRANSMIT**  
     DA = IUT,  
     SA = TD,  
     Initialize-Routing-Table  
     Number of Ports = 0
2. **WAIT Internal Processing Fail Time**
3. (**CHECK** (that the IUT did not send any packets in response to the Initialize-Routing-Table)) |  
     (**RECEIVE**  
         DESTINATION = TD,  
         SOURCE = UT,  
         Reject-Message-To-Network  
         DNET = (any valid value)  
         Rejection Reason = 0 (other) | 3 (unknown))
4. **TRANSMIT**  
     DA = IUT,  
     SA = TD,  
     Establish-Connection-To-Network  
     DNET = DNET3  
     Termination Time Value = 0
5. **WAIT Internal Processing Fail Time**
6. (**CHECK**(that the IUT did not send any packets in response to the Establish-Connection-To-Network)) |  
     (**RECEIVE**  
         DESTINATION = TD,  
         SOURCE = IUT,  
         Reject-Message-To-Network  
         DNET = (any valid value)  
         Rejection Reason = 0 (other) | 3 (unknown))
7. **TRANSMIT**  
     DA = IUT,  
     SA = TD,  
     Disconnect-Connection-To-Network  
     DNET = NET3
8. **WAIT Internal Processing Fail Time**
9. (**CHECK**(that the IUT did not send any packets in response to the Disconnect-Connection-To-Network)) |  
     (**RECEIVE**  
         DESTINATION = TD,  
         SOURCE = IUT,  
         Reject-Message-To-Network  
         DNET = (any valid value)  
         Rejection Reason = 0 (other) | 3 (unknown))

### 10.7 Route Binding Tests

This subclause defines the tests necessary to demonstrate the portion of BACnet Network Layer functionality that is used to determine network routes. These tests are generic and are meant to be applied to both router and non-router nodes.

The tests assume that the IUT is located on network DNET1 and the TD appears to be a router to network DNET2. The value DNET3 is assigned a unique network number. If the IUT can initiate requests, it will be configured to send those requests to a device (D2A) on network DNET2. The IUT will also be expected to respond to requests from device D2A. The test descriptions assume that the TD will mimic device D2A.

Note: Clauses 6.5.1 and 6.5.3 indicate that there are only two ways for a non-router to transmit a request (on the local network and destined for a remote network), neither of which includes network layer source routing information. If the IUT is configured as a non-router and it emits any NPDU with SNET/SADR fields during the tests in this subclause, then it shall fail.

Note: Clauses 6.6 and 6.6.3.3 define BACnet routers and the network layer services reserved for routers. If the IUT is configured as a non-router and it emits any I-Am-Router-To-Network or I-Could-Be-Router-To-Network NPDUs during the tests in this subclause (including during test 10.7.2), then it shall fail.

#### 10.7.1 Static Router Binding

Purpose: To verify that the IUT can initiate requests to a remote network when the IUT has been statically configured with the MAC address of the router to that remote network.

Test Concept: The IUT transmits a request to a device on the remote network without performing any form of dynamic router binding. If the IUT does not support static router binding or if the IUT cannot initiate a request, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another service can be substituted.

Configuration Requirements: The IUT is configured with the TD's MAC address as the router for network DNET2.

Test Steps:

1. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)

2. RECEIVE

DA = TD,  
SA = IUT,  
DNET = DNET2,  
DADR = D2A,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (O1, any BACnet standard object in D2A),  
'Property Identifier' = (P1, any required property of the specified object)

3. TRANSMIT

DA = IUT,  
SA = TD,  
SNET = DNET2,  
SADR = D2A,  
BACnet-ComplexACK-PDU,  
'Service ACK Choice' = ReadProperty-ACK,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value)

#### 10.7.2 Router Binding via Application Layer Services

Purpose: To verify that the IUT can initiate requests to a remote network and respond to requests from a remote network after the IUT uses the Who-Is and I-Am Application Layer services to discover the MAC address of the router to that remote network.

Test Concept: The IUT broadcasts a Who-Is request to discover device D2A and notes the MAC address of the intervening router in the corresponding I-Am reply. The TD transmits a request to a device on the remote network and

responds to a request from the remote network without performing any further form of dynamic router binding. If the IUT does not support application layer router binding, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted. The IUT may use the deviceInstanceRange form of Who-Is.

Clause 6.5.3 specifically mentions router binding via Who-Is and does not mention router binding by initiating other application layer services (such as Who-Has) or by lurking and noting the router MAC addresses for incoming application layer requests. For this reason the test only allows for router binding via Who-Is.

Test Steps:

1. MAKE (IUT transmit Who-Is to discover the device on the remote network)
2. RECEIVE
  - DA = BROADCAST,
  - SA = IUT,
  - DNET = GLOBAL BROADCAST,
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = Who-Is
  - | (DA = BROADCAST,
  - SA = IUT,
  - DNET = DNET2,
  - DADR= BROADCAST or D2A,
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = Who-Is)
3. TRANSMIT
  - DA = BROADCAST,
  - SA = TD,
  - SNET = DNET2,
  - SADR = D2A,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = I-Am,
  - 'I Am Device Identifier' = (device object, instance number of D2A),
  - 'Max APDU Length Accepted ' = (any valid value),
  - 'Segmentation Supported' = (any valid value),
  - 'Vendor ID ' = (any valid value)
4. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)
5. RECEIVE
  - DA = TD,
  - SA = IUT,
  - DNET = DNET2,
  - DADR= D2A,
  - Hop Count = 255,
  - BACnet-Confirmed-Request-PDU,
  - 'Service Choice' = ReadProperty-Request,
  - 'Object Identifier' = (O1, any BACnet standard object in D2A),
  - 'Property Identifier' = (P1, any required property of the specified object)
6. TRANSMIT
  - DA = IUT,
  - SA = TD,
  - SNET = DNET2,
  - SADR = D2A,
  - BACnet-ComplexACK-PDU,
  - 'Service ACK Choice' = ReadProperty-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value)



### 10.7.3 Router Binding via Who-Is-Router-To-Network

**Purpose:** To verify that the IUT can initiate requests to a remote network after the IUT uses the Who-Is-Router-To-Network Network Layer service to discover the MAC address of the router to that remote network.

**Test Concept:** The IUT broadcasts a Who-Is-Router-To-Network request to discover the router to the desired network. The TD transmits a request to a device on the remote network without performing any further form of dynamic router binding. If the IUT does not support Who-Is-Router-To-Network router binding, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted. The IUT may use either the general query or specific network number query form of the Who-Is-Router-To-Network service.

Note that Clause 6.5.3 specifically mentions router binding via Who-Is-Router-To-Network and does not mention router binding by lurking and noting unsolicited I-Am-Router-To-Network messages.

**Test Steps:**

1. MAKE (IUT transmit Who-Is-Router-To-Network to discover the router to DNET2)
2. RECEIVE
  - DA = BROADCAST,
  - SA = IUT,
  - Who-Is-Router-To-Network,
  - | (DA = BROADCAST,
  - SA = IUT,
  - Who-Is-Router-To-Network,
  - DNET = DNET2)
3. TRANSMIT
  - DESTINATION = BROADCAST,
  - SOURCE = TD,
  - I-Am-Router-To-Network,
  - Network Numbers = DNET2
4. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)
5. RECEIVE
  - DA = TD,
  - SA = IUT,
  - DNET = DNET2,
  - DADR = D2A,
  - Hop Count = 255,
  - BACnet-Confirmed-Request-PDU,
  - 'Service Choice' = ReadProperty-Request,
  - 'Object Identifier' = (O1, any BACnet standard object in D2A),
  - 'Property Identifier' = (P1, any required property of the specified object)
6. TRANSMIT
  - DA = IUT,
  - SA = TD,
  - SNET = DNET2,
  - SADR = D2A,
  - BACnet-ComplexACK-PDU,
  - 'Service ACK Choice' = ReadProperty-ACK,
  - 'Object Identifier' = O1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any valid value)

### 10.7.4 Router Binding via Broadcast

**Purpose:** To verify that the IUT can initiate requests to a remote network when the IUT uses an initial broadcast to discover the MAC address of the router to that remote network.

**Test Concept:** The IUT broadcasts a request to a device on the remote network and notes the MAC address of the intervening router in the reply. If the IUT does not support router binding via broadcast, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted.

Test Steps:

1. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)

2. RECEIVE

DA = BROADCAST,  
SA = IUT,  
DNET = DNET2,  
DADR = D2A,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (O1, any BACnet standard object in D2A),  
'Property Identifier' = (P1, any required property of the specified object)

3. TRANSMIT

DA = IUT,  
SA = TD,  
SNET = DNET2,  
SADR = D2A,  
BACnet-ComplexACK-PDU,  
'Service ACK Choice' = ReadProperty-ACK,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value)

4. MAKE (IUT transmit a ReadProperty request to the D2A device on the remote network)

5. RECEIVE

DA = TD,  
SA = IUT,  
DNET = DNET2,  
DADR = D2A,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (O1, any BACnet standard object in D2A),  
'Property Identifier' = (P1, any required property of the specified object)

6. TRANSMIT

DA = IUT,  
SA = TD,  
SNET = DNET2,  
SADR = D2A,  
BACnet-ComplexACK-PDU,  
'Service ACK Choice' = ReadProperty-ACK,  
'Object Identifier' = O1,  
'Property Identifier' = P1,  
'Property Value' = (any valid value)

#### 10.7.5 Reuse Router Binding Found By Broadcast

**Purpose:** This test verifies that a device which uses a local broadcast of a confirmed-request PDU (and thus in which the data-expecting-reply (DER) of the NPDU control octet is set) to route to a device on a remote network uses the address information from the response for subsequent requests to that device.

**Test Concept:** Configure the IUT so that it will use a local network broadcast of a confirmed-request PDU (i.e., the data-expecting-reply of the NPCI octet is set) in order to locate a router on the local network towards the specified remote device. TD impersonates a local router, responding with a PDU that appears to have been routed from RD. The IUT communicates with RD again but does not use a local network broadcast because it does not need to, having retained the address binding to TD which is impersonating the local router.

**Configuration Requirements:** The IUT shall be configured to request information from a device on a remote network, RD, using a BACnetAddress (network number and MAC address), and to use a MAC broadcast of a confirmed request PDU in order to locate a router to that network, on the local network, in order to communicate to that device. If the IUT cannot be configured in this fashion, then this test shall be skipped.

Test Steps:

1. MAKE (IUT initiate a request to RD)
2. RECEIVE
 

|             |                         |
|-------------|-------------------------|
| SA =        | IUT,                    |
| DA =        | MAC BROADCAST,          |
| DNET =      | (network number of RD), |
| DADR =      | (MAC address of RD),    |
| APDU Type = | Confirmed-Request       |
3. TRANSMIT
 

|             |                                                  |
|-------------|--------------------------------------------------|
| SA =        | TD,                                              |
| DA =        | IUT,                                             |
| SNET =      | (network number of RD),                          |
| SADR =      | (MAC address of RD),                             |
| APDU Type = | SimpleACK   Complex-ACK   Error-PDU   Reject-PDU |
4. MAKE (IUT initiate a request to RD)
5. RECEIVE
 

|             |                          |
|-------------|--------------------------|
| SA =        | IUT,                     |
| DA =        | TD,                      |
| DNET =      | (network address of RD), |
| DADR =      | (MAC address of RD),     |
| APDU Type = | Confirmed-Request        |
6. TRANSMIT BACnet-SimpleACK-PDU

### 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.

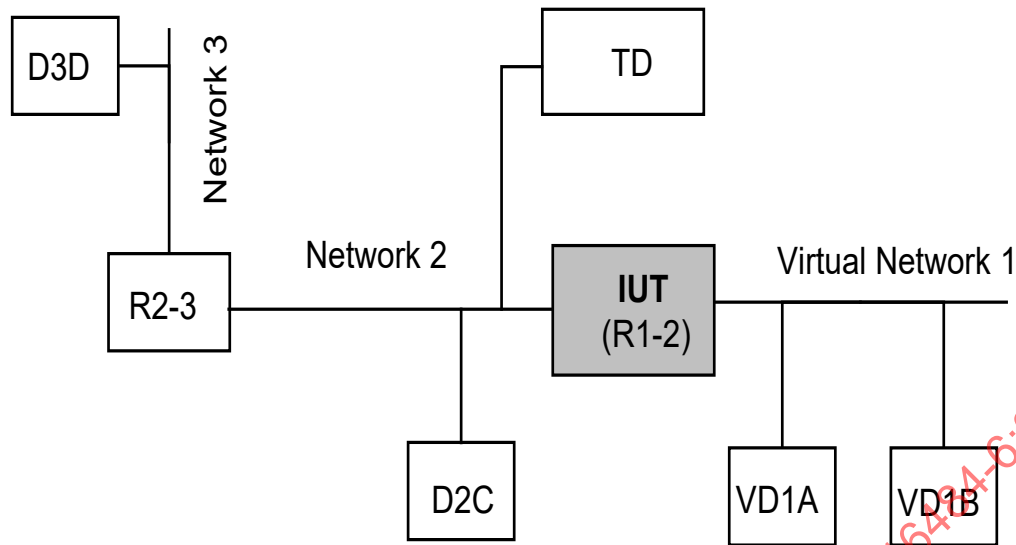


**Figure 10-4.** Logical internetwork configuration for virtual routing functionality tests.

### 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.

Notes to Tester: The destination device (VD1A) can be any virtual device in the IUT.

Test Steps:

1. TRANSMIT,
  - DA = LOCAL BROADCAST,

SA = TD,

DNET = 1,

DADR = VD1A,

Hop Count = 255,

ReadProperty-Request,

'Object Identifier' = (the object identifier of any object in the target device),

'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)

2. RECEIVE,

DA = TD,

SA = IUT,

SNET = 1,

SADR = VD1A,

ReadProperty-ACK,

'Object Identifier' = (the object identifier used in step 1),

'Property Identifier' = (the property identifier used in step 1),

'Property Value' = (the contents of the specified property)

3. TRANSMIT,

DA = IUT,

SA = TD,

DNET = 1,

DADR = VD1A,

Hop Count = 255,

ReadProperty-Request,

'Object Identifier' = (the object identifier of any object in the target device),

'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented, but not the same property as in step 1)

4. RECEIVE,

DA = TD,

SA = IUT,

SNET = 1,

SADR = VD1A,

ReadProperty-ACK,

'Object Identifier' = (the object identifier used in step 3),

'Property Identifier' = (the property identifier used in step 3),

'Property Value' = (the contents of the specified property)

### 10.8.3.2 Route Request Message from a Virtual Device to a Local Device

Purpose: To verify that the IUT can route a unicast request message from a virtual device to a local device.

Test Concept: Make one of the virtual devices generate a unicast request, and verify that the NPCI is correctly formed. This test shall be skipped if none of the IUT's virtual devices can issue a confirmed or unconfirmed request in a unicast message.

Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a unicast message to a particular target device on Network 2.

Notes to Tester: During the test, the TD shall answer any requests that the IUT generates while attempting to locate the route to the target device.

Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a unicast message to a device on a local network. Depending on the capabilities of the IUT, this may involve sending a message from the target device to the IUT (unicast or broadcast), writing the network address of the target device to an object property in the IUT, writing the Device ID of the target device to an object property in the IUT, writing the Device Name of the target device to an object property in the IUT, or configuring the IUT using a proprietary method. The IUT may need to broadcast a Who-Is or Who-Has request in order to discover the network address of the target device if the network address is unknown.



Test Steps:

1. 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 any virtual device on Network 1),  
     BACnet-Confirmed-Request-PDU or BACnet-Unconfirmed-Request-PDU

#### 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.

Notes to Tester: Choose a virtual device on Network 1 for this test.

Test Steps:

1. TRANSMIT,  
     DA = LOCAL BROADCAST,  
     SA = TD,  
     DNET = 59001,  
     DADR = (the MAC address of the selected virtual device),  
     Hop Count = 255,  
     ReadProperty-Request,  
     'Object Identifier' = (any object identifier of an object in the virtual device),  
     'Property Identifier' = (any property of the specified object)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT did not transmit I-Am-Router-To-Network (Network Numbers = 59001...) or Reject-Message-To-Network (Network Number = 59001) or any message in response to the Read Property request on Network 2)

#### 10.8.3.5.2 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)

**10.8.3.6 Silently Drop Messages to a Virtual Device that is Offline**

Purpose: To verify that the IUT does not return any message in response to an NPDU with a destination that is offline.

Test Concept: The non-BACnet device is verified to be online and recognized by the IUT. It is then made to go offline, and the IUT is made to recognize that the device is offline. A property, P1, from Object1 which is derived from the data in a virtual device is read from the IUT. Verify that when a virtual device is off-line, that the IUT sends no response to messages that are directed to that off-line device.

Configuration Requirements: The IUT acting as a virtual router shall be configured so that a virtual device VD1A, which can sometimes be online, is initially online for this test. If no virtual device can become off-line, then this test shall be skipped.

Test Steps:

1. CHECK (any vendor-specified indication, that the virtual device is online)
2. MAKE (the virtual device containing Object1 go offline)
3. MAKE (the IUT notice that the virtual device is offline)
4. TRANSMIT ReadProperty-Request,  
DESTINATION = VD1A,  
'Object Identifier' = Object1,  
'Property Identifier' = P1
5. CHECK (that no responsive message is returned from IUT)
6. TRANSMIT  
DESTINATION = VD1A,  
Message Type = (any valid value)
7. CHECK (that no responsive message is returned from IUT)

**10.8.4 Routing of Broadcast APDUs to Virtual Devices****10.8.4.1 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.

Test Steps:

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.

#### Test Steps:

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.

#### Test Steps:

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.

Test Steps:

## 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.

Test Steps:

## 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.

#### Test Steps:

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.

#### Test Steps:

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.

Test Steps:

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.

Test Steps:

#### 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:

Test Steps:

#### 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:

Test Steps:

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.

Configuration Requirements: The IUT shall be configured or otherwise stimulated so that one of its virtual devices will send a remote broadcast message to Network 2.

Notes to Tester: This test should be run repeatedly in order to exercise all ways that the IUT can be configured or stimulated to send a broadcast message to a local (physical) network. Depending on the capabilities of the IUT, this may involve sending a message from a device on the target network to the IUT (unicast or broadcast), writing a broadcast address to an object property in the IUT, or configuring the IUT using a proprietary method.

Test Steps:

1. MAKE (the virtual device generate a remote broadcast message to the local network of the IUT)
2. RECEIVE,
  - DA = LOCAL BROADCAST,
  - SA = IUT,
  - SNET = 1,
  - SADR = (MAC address of a virtual device on Network 1),
  - BACnet-Unconfirmed-Request-PDU

#### 10.8.4.8 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

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.

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.

#### 10.8.7.4 Who-Is Specifying Unknown Device Ids

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

1. TRANSMIT,
  - DA = IUT,
  - SA = TD,
  - DNET = 1,

- DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
Who-Is-Request,  
'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)  
'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)
- CHECK (verify that the IUT does not transmit an I-Am-Request-PDU)

#### 10.8.7.5 Who-Has Specifying Unknown Device Ids

Purpose: To verify that the IUT will not respond to discovery for devices that it does not contain.

Test Steps:

- TRANSMIT,  
DA = IUT,  
SA = TD,  
DNET = 1,  
DLEN = 0,  
Hop Count = 255,  
BACnet-Unconfirmed-Request-PDU,  
Who-Has-Request,  
'Device Instance Range Low Limit' = (Low Limit of instance range excluding all virtual devices)  
'Device Instance Range High Limit' = (High Limit of instance range excluding all virtual devices)  
'Object Identifier' = (Device object identifier of VD1B)
- CHECK (verify that the IUT does not transmit an I-Have-Request-PDU)

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2024



## 11. LOGICAL LINK LAYER PROTOCOL TESTS

This clause defines the tests necessary to demonstrate the proper operation of the ISO/IEC 8802-2 Logical Link Control (Type 1) layers specified by BACnet for use with ISO/IEC 8802-3 ("Ethernet") and ARCNET data link layers. These tests are based on ISO/IEC 8802-2 (1994), and the reference clauses refer to this document.

The following definitions are made for Destination Service Access Point (DSAP) and Source Service Access Point (SSAP) values.

| DSAP Type:        | Octet Value: |
|-------------------|--------------|
| BACnet_Individual | X'82'        |
| BACnet_Group      | X'83'        |
| Null_Individual   | X'00'        |
| Global_Group      | X'FF'        |

| SSAP Type:      | Octet Value: |
|-----------------|--------------|
| BACnet_Command  | X'82'        |
| BACnet_Response | X'82'        |
| Null_Command    | X'00'        |
| Null_Response   | X'01'        |

### 11.1 UI Command and Response

Purpose: All BACnet messages are conveyed by LLC Class I services using the Unnumbered Information (UI) command. This test verifies that the correct DSAP and SSAP values are used.

Test Steps:

- TRANSMIT ReadProperty-Request,  
 DSAP = BACnet\_Individual,  
 SSAP = BACnet\_Command,  
 CTL = X'03',  
 'Object Identifier' = (the device's Device object),  
 'Property Identifier' = Object\_Identifier
- BEFORE **Acknowledgement Fail Time**  
 RECEIVE BACnet-ComplexACK-PDU,  
 DSAP = BACnet\_Individual,  
 SSAP = BACnet\_Command,  
 CTL = X'03',  
 'Property Value' = (the device's Device object)

### 11.2 XID Command and Response

Purpose: To verify that the LLC correctly responds to Exchange Identification (XID) requests.

Test Steps:

- REPEAT tSSAP = (BACnet\_Command, Null\_Command) DO {  
 REPEAT tDSAP = (BACnet\_Individual, BACnet\_Group,  
 Global\_Group, Null\_Individual) DO {  
 REPEAT tCTL = (X'AF', X'BF') DO {  
 TRANSMIT XID-Request,  
 DSAP = tDSAP,  
 SSAP = tSSAP,  
 CTL = tCTL,  
 XID-Information = X'810100'  
 BEFORE **Acknowledgement Fail Time**  
 RECEIVE XID-Response,  
 DSAP = (BACnet\_Individual | Null\_Individual),  
 SSAP = (BACnet\_Response | Null\_Response),

```

 CTL = tCTL,
 XID-Information = X'810100'
 }
}

```

### 11.3 TEST Command and Response

Purpose: To verify that the LLC correctly responds to TEST requests.

In the following test arbitrary data octets may be appended to the TEST-Request PDU; as many octets as can be transferred in a single PDU across the data link can be included. If the IUT returns data in the TEST-Response PDU, it shall return the same data as transmitted in the TEST-Request PDU.

Test Steps:

```

1. REPEAT tSSAP = (BACnet_Command, Null_Command) DO {
 REPEAT tDSAP = (BACnet_Individual, BACnet_Group,
 Global_Group, Null_Individual) DO {
 REPEAT tCTL = (X'E3', X'F3') DO {
 TRANSMIT TEST-Request,
 DSAP = tDSAP,
 SSAP = tSSAP,
 CTL = tCTL
 BEFORE Acknowledgement Fail Time
 RECEIVE TEST-Response,
 DSAP = (IF (tSSAP is Null_Command) THEN
 Null_Individual
 ELSE
 BACnet_Individual)
 SSAP = (IF (tDSAP is Null_Individual) THEN
 Null_Response
 ELSE
 BACnet_Response)
 CTL = tCTL
 }
 }
}

```

## 12. DATA LINK LAYER PROTOCOLS TESTS

### 12.1 MS/TP State Machine Tests

#### 12.1.1 MS/TP Master Tests

The tests defined in this clause shall be used to verify that a BACnet MS/TP master device properly implements the Receive Frame Finite State Machine defined in BACnet Clause 9.5.4, the SendFrame procedure of BACnet Clause 9.5.5, and the Master Node Finite State Machine of BACnet Clause 9.5.6. The state machine diagrams in BACnet Figures 9-3 and 9-4 may be helpful in interpreting the tests.

Some state machine transitions may be caused by one of several different conditions. In order to differentiate the tests being performed, the multiple conditions are labeled consecutively with the letters 'a', 'b', etc.

These tests shall be performed at every baud rate supported by the IUT, with only the IUT and TD present on the MS/TP LAN. These tests must be run completely in sequence, without pauses between tests, in order to ensure the proper machine states of the master device. During all tests, replies from the IUT must be returned before the time specified by  $T_{reply\_delay}$ , there must be no inter-frame gaps greater than the interval specified by  $T_{frame\_gap}$ , and after receiving a Token or Poll For Master frame the IUT must begin transmitting the next frame within the time specified by  $T_{usage\_delay}$ .

##### 12.1.1.1 Test Setup

The IUT shall be set to MAC address 2 with  $N_{max\_master}$  set to 127 and  $N_{max\_info\_frames}$  set to 2, if possible (if not, the transition DONE\_WITH\_TOKEN:SendAnotherFrame is untestable). The TD shall be set to MAC address 3.

The TD must know the device instance of the device being tested. It also must know of an unconfirmed service supported by the device, if any.

##### 12.1.1.2 Startup Tests

These tests verify the basic operation of the IUT.

##### 12.1.1.2.1 SendFrame Test

Purpose: To verify the SendFrame procedure. The following state machine transitions are verified by this test:

Master Node:

|               |                  |
|---------------|------------------|
| INITIALIZING: | DoneInitializing |
| IDLE:         | ReceivedPFM      |

Receive Frame:

|             |                                                             |
|-------------|-------------------------------------------------------------|
| IDLE:       | Preamble1                                                   |
| PREAMBLE:   | Preamble2                                                   |
| HEADER:     | FrameType, Destination, Source, Length1, Length2, HeaderCRC |
| HEADER CRC: | Data (destination address = TS)                             |
| DATA:       | DataOctet, CRC1, CRC2                                       |
| DATA CRC:   | GoodCRC                                                     |

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
  - TRANSMIT
    - Poll For Master
  - IF (IUT is turned off) THEN
    - MAKE (IUT turned on or otherwise started)
3. RECEIVE
  - Reply To Poll For Master

Notes to Tester: In this test, IUT initialization is defined to be complete when the IUT emits a Reply To Poll For Master frame.

**12.1.1.2.2 Confirmed Service Request Transitions**

Purpose: To verify that the IUT can receive and understand properly formed frames, and create a properly formed frame in response. The following state machine transitions are verified by this test:

Master Node:

|                      |                          |
|----------------------|--------------------------|
| INITIALIZING:        | DoneInitializing         |
| IDLE:                | ReceivedDataNeedingReply |
| ANSWER DATA REQUEST: | Reply                    |

Receive Frame:

|             |                                                             |
|-------------|-------------------------------------------------------------|
| IDLE:       | Preamble1                                                   |
| PREAMBLE:   | Preamble2                                                   |
| HEADER:     | FrameType, Destination, Source, Length1, Length2, HeaderCRC |
| HEADER CRC: | Data (destination address = TS)                             |
| DATA:       | DataOctet, CRC1, CRC2                                       |
| DATA CRC:   | GoodCRC                                                     |

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
Test\_Request,  
'Length' = (value from 1 to 50),  
'Data' = ('Length' number of octets)
4. RECEIVE  
Test\_Response

**12.1.1.3 State Machine Transition Tests for Error Transitions**

This clause defines the test cases necessary to demonstrate correct operation of the Receive Frame State Machine under circumstances where confirmed and unconfirmed service requests are received and data link errors are encountered by the state machine.

**12.1.1.3.1 Error Tests with no Response**

This clause defines the test cases where the data link errors cause the frame to be discarded and no response is issued.

**12.1.1.3.1.1 Bad Data CRC**

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with an incorrect Data CRC. Let X be the instance number of the Device Object for the IUT. The following Receive Frame State Machine transition is verified by this test:

|           |        |
|-----------|--------|
| DATA CRC: | BadCRC |
|-----------|--------|

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
Test\_Request
4. RECEIVE  
Test\_Response
5. TRANSMIT  
ReadProperty-Request,  
'Frame Type' = BACnet Data Expecting Reply,  
'Object Identifier' = (Device,X),  
'Property Identifier' = Object\_Identifier,  
'Data CRC' = (any incorrect value)
6. BEFORE (Acknowledgement Fail Time) {  
IF (ReadProperty-ACK received) THEN  
ERROR "Incorrect MS/TP Frame Data CRC undetected."

7. TRANSMIT  
     ReadProperty-Request,  
     'Frame Type' = BACnet Data Expecting Reply,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = Object\_Identifier,  
     'Service Request' = ReadProperty-Request
8. RECEIVE  
     ReadProperty-ACK,  
     'Frame Type' = BACnet Data Not Expecting Reply,  
     'Object Identifier' = (Device, X),  
     'Property Identifier' = Object\_Identifier,  
     'Property Value' = (Device, X)

#### 12.1.1.3.1.2 Data Timeout

Purpose: To verify that the Receive Frame State Machine detects and rejects frames that timeout during data field transmission. The following Receive Frame State Machine transition is verified by this test:

DATA: Timeout

Test Steps: During the transmission of step 5, the transmission shall be paused for a time greater than  $T_{\text{frame\_abort}}$  sometime after the Header CRC octet has been transmitted but before the final Data CRC octet is transmitted.

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
     Test\_Request
4. RECEIVE  
     Test\_Response
5. TRANSMIT  
     Test\_Request,  
     'Length' = (x where  $1 \leq x \leq 50$ ),  
     'Data' = (x number of octets)
6. BEFORE ( $T_{\text{reply\_delay}}$ ) {  
     IF (anything received) THEN  
         ERROR "MS/TP Framing error: undetected."  
     }
7. TRANSMIT  
     Test\_Request,  
     'Length' = (x where  $1 \leq x \leq 50$ ),  
     'Data' = (x number of octets)
8. RECEIVE  
     Test\_Response,  
     'Data' = (the same data transmitted in step 7)

#### 12.1.1.3.1.3 Data Framing Error

Purpose: To verify that the Receive Frame State Machine detects and rejects frames in which a transmission framing error occurs in the data. The following Receive Frame State Machine transition is verified by this test:

DATA: Error

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
     Test\_Request
4. RECEIVE  
     Test\_Response

5. TRANSMIT  
     Test\_Request,  
     'Length' = (x where  $1 \leq x \leq 50$ ),  
     'Data' = (x number of octets)
6. BEFORE (T<sub>reply\_delay</sub>) {  
     IF (anything received) THEN  
         ERROR "MS/TP Framing error undetected."  
     }
7. TRANSMIT  
     Test\_Request,  
     'Length' = (x where  $1 \leq x \leq 50$ ),  
     'Data' = (x number of octets)
8. RECEIVE  
     Test\_Response

Notes to Tester: During the transmission of step 5, one octet in the Data field shall be transmitted with a logical zero in the stop bit position.

#### 12.1.1.3.1.4 Bad Header CRC

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with an incorrect Header CRC. The following Receive Frame State Machine transition is verified by this test:

HEADER CRC:           BadCRC

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
     Test\_Request
4. RECEIVE  
     Test\_Response
5. TRANSMIT  
     Test\_Request  
     'Header CRC' = (any incorrect value)
6. BEFORE (T<sub>reply\_delay</sub>) {  
     IF (anything received) THEN  
         ERROR "MS/TP Frame Header CRC error undetected."  
     }
7. TRANSMIT  
     Test\_Request
8. RECEIVE  
     Test\_Response

#### 12.1.1.3.1.5 Not For Us

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with a destination address different from the IUT address. The following Receive Frame State Machine transition is verified by this test:

HEADER CRC:           NotForUs

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT  
     Test\_Request
4. RECEIVE  
     Test\_Response
5. TRANSMIT  
     DA = (MAC address not used by TD or IUT),

- ```

Test_Request
6. BEFORE (Treply_delay) {
    IF (anything received) THEN
        ERROR "Device responded to MAC address not its own."
    }
7. TRANSMIT
Test_Request
8. RECEIVE
Test_Response

```

12.1.1.3.1.6 Header Framing Error

Purpose: To verify that the Receive Frame State Machine detects and rejects frames in which a transmission framing error occurs in the Header. The following Receive Frame State Machine transition is verified by this test:

HEADER: Error

Test Steps:

- ```

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test_Request
6. BEFORE (Treply_delay) {
 IF (anything received) THEN
 ERROR "MS/TP Frame Header framing error undetected."
 }
7. TRANSMIT
Test_Request
8. RECEIVE
Test_Response

```

Notes to Tester: During the transmission of step 5, one octet in the Header shall be transmitted with a logical zero in the stop bit position.

#### 12.1.1.3.1.7 Header Timeout

Purpose: To verify that the Receive Frame State Machine detects and rejects frames that timeout during header field transmission. The following Receive Frame State Machine transition is verified by this test:

HEADER: Timeout

Test Steps:

- ```

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test_Request
6. BEFORE (Treply_delay) {
    IF (anything received) THEN
        ERROR "MS/TP Frame Header timeout undetected."
    }
7. TRANSMIT
Test_Request

```


8. RECEIVE

Test_Response

Notes to Tester: During the transmission of step 5, the transmission shall be halted after the second Preamble octet is transmitted and before the Header CRC octet is transmitted.

12.1.1.3.1.8 Not Preamble

Purpose: To verify that the Receive Frame State Machine correctly rejects incorrectly formed preambles. The following Receive Frame State Machine transition is verified by this test:

HEADER: NotPreamble

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
X'55', (any value other than X'55' and X'FF')
6. TRANSMIT
Test_Request
7. RECEIVE
Test_Response

12.1.1.3.1.9 Eat An Error

Purpose: To verify that the Receive Frame State Machine correctly rejects an initial preamble octet for which a ReceiveError occurred. The following Receive Frame State Machine transition is verified by this test:

IDLE EatAnError

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test_Request
6. BEFORE (T_{reply delay}) {
IF (anything received) THEN
ERROR "MS/TP Preamble receive error undetected."
}
7. TRANSMIT
Test_Request
8. RECEIVE
Test_Response

Notes to Tester: During the transmission of step 5, the X'55' octet in the preamble shall be transmitted with a logical zero in the stop bit position.

12.1.1.3.1.10 Eat An Octet

Purpose: To verify that the Receive Frame State Machine correctly rejects an initial preamble octet that does not have the value X'55'. The following Receive Frame State Machine transition is verified by this test:

IDLE: EatAnOctet

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test Request,
'Preamble' = (any value other than X'55), X'FF'
6. BEFORE (T_{reply_delay}) {
IF (anything received) THEN
ERROR "MS/TP Frame incorrect Preamble undetected."
}
7. TRANSMIT
Test_Request
8. RECEIVE
Test_Response

12.1.1.3.1.11 Frame Too Long

Purpose: To verify the Receive Frame state machine error check for a frame too large for the IUT. This tests the following Receive Frame State machine transition:

HEADER CRC: FrameTooLong

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test Request,
'Length' = 502,
'Data' = (502 octets)
6. BEFORE (T_{reply_delay}) {
IF (anything received) THEN
ERROR "MS/TP Frame Header framing error undetected."
}
7. TRANSMIT
Test_Request,
'Length' = 0
8. RECEIVE
Test_Reply

12.1.1.3.2 Tests with Response

This clause defines the test cases where data link errors are corrected or which cause a response to be issued.

12.1.1.3.2.1 Repeated Preamble1

Purpose: To verify the Receive Frame state machine check for a repeated first preamble octet. The following Receive Frame State Machine transition is verified by this test:

PREAMBLE: RepeatedPreamble1

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
X'55'
6. TRANSMIT
Test_Request
7. RECEIVE
Test_Response

12.1.1.3.2.2 Test Request Empty Frame

Purpose: To verify acceptance of an empty Test_Request frame with reply via Test_Response. This verifies the Receive Frame State Machine transition:

HEADER CRC: NoData

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request,
'Length' = 0
4. RECEIVE
Test_Response,
'Length' = 0

12.1.1.3.2.3 Test Request With Data

Purpose: To verify acceptance of a Test_Request frame with data and reply via Test_Response.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request,
'Length' = (x where $1 \leq x \leq 50$),
'Data' = (x number of octets)
4. RECEIVE
Test_Response,
'Length' = (0 or value in step 3),
'Data' = (empty or octets transmitted in step 3)

12.1.1.4 State Machine Transition Tests for Token Operations

This clause defines the test cases necessary to demonstrate correct operation of the Master Node Finite State Machine in receiving and passing the token, and in generating a new token when the previous is lost.

12.1.1.4.1 Token Passed to IUT

Purpose: To verify that the IUT correctly receives the token, uses it, and begins its search for the next master. Depending upon the implementation and setup of the IUT, there are three possibilities in the sequence of verified Master Node Finite State Machine transitions:

IDLE:	ReceivedToken (tested in all cases)
USE_TOKEN:	NothingToSend (tested if no data frames are sent)
USE_TOKEN:	SendNoWait (tested if a frame not expecting reply is sent)

DONE_WITH_TOKEN: SendAnotherFrame (tested if multiple frames are sent)
 USE_TOKEN: SendAndWait (tested if a frame expecting reply is sent)
 WAIT_FOR_REPLY: ReplyTimeout (tested if there is no response to a frame expecting reply)
 DONE_WITH_TOKEN: SendMaintenancePFM (tested in all cases)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Poll For Master
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 Reply To Poll For Master
4. TRANSMIT
 Token
5. WHILE (received frame not a Poll For Master) DO {
 }
6. RECEIVE
 DA = IUT+1,
 Poll For Master

12.1.1.4.2 Token Passed by IUT

Purpose: To verify that the IUT correctly responds to a Reply To Poll For Master by passing the token. This tests the following Master Node Finite State Machine transition:

POLL_FOR_MASTER: ReceivedReplyToPFM

Depending upon the implementation and setup of the IUT, there are three possibilities in these sequences of verified Master Node Finite State Machine transitions:

In all cases:

PASS_TOKEN: SawTokenUser
 IDLE: ReceivedToken

Case1: Nothing is sent.

USE_TOKEN: NothingToSend

Case 2: A frame not expecting a reply is to be sent:

USE_TOKEN: SendNoWait

This could repeat once by the following transition that returns to the USE_TOKEN state.

DONE_WITH_TOKEN: SendAnotherFrame

Case 3: A frame expecting a reply is sent:

USE_TOKEN: SendAndWait

WAIT_FOR_REPLY: InvalidFrame

In all cases:

DONE_WITH_TOKEN: SendToken (b)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Poll For Master
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)

```

    }
3. RECEIVE
   Reply To Poll For Master
4. TRANSMIT
   Token
5. WHILE (received other than Poll For Master frame) DO {
   IF (frame is BACnet Data Expecting Reply) THEN
     TRANSMIT
       SA = (DA of received frame),
       BACnet Data Not Expecting Reply,
       'Header CRC' = (any incorrect value)
   }
6. RECEIVE
   DA = IUT+1,
   Poll For Master
7. TRANSMIT
   SA = IUT+1,
   DA = IUT,
   Reply To Poll For Master
8. RECEIVE
   DA = IUT+1,
   Token
9. TRANSMIT
   DA = (MAC address other than DA and IUT),
   Test_Request
10. BEFORE (Tno_token – 1 milliseconds) {
   IF (anything received) THEN
     ERROR "Passed token use undetected by IUT."
   }

```

12.1.1.4.3 Token Dropped After Passing

Purpose: To verify the correct operation of the IUT when the token is dropped after being passed to another device. This tests the transitions:

PASS_TOKEN:	RetrySendToken
PASS_TOKEN:	FindNewSuccessor
POLL_FOR_MASTER:	SendNextPFM (a)

Test Steps:

```

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
   TRANSMIT
     Poll For Master
   IF (IUT is turned off) THEN
     MAKE (IUT turned on or otherwise started)
   }
3. RECEIVE
   Reply To Poll For Master
4. TRANSMIT
   Token
5. WHILE (received frame other than Poll For Master) DO {
   IF (frame is BACnet Data Expecting Reply) THEN
     TRANSMIT
       SA = (DA of received frame),
       BACnet Data Not Expecting Reply,
       'Header CRC' = (any incorrect value)
   }
6. RECEIVE
   DA = IUT+1,

```

- Poll For Master
- 7. TRANSMIT
 - SA = IUT+1,
 - DA = IUT,
 - Reply To Poll For Master
- 8. RECEIVE
 - DA = IUT+1,
 - Token
- 9. WAIT $T_{usage\_timeout}$
- 10. RECEIVE
 - DA = IUT+1,
 - Token
- 11. WAIT $T_{usage\_timeout}$
- 12. RECEIVE
 - DA = IUT+2,
 - Poll For Master
- 13. WAIT $T_{usage\_timeout}$
- 14. RECEIVE
 - DA = IUT+3,
 - Poll For Master

12.1.1.4.4 Poll For Master - Invalid Frame

Purpose: To verify the correct operation of the IUT when an invalid frame is received in response to a transmitted Poll For Master frame. This tests the transition:

POLL_FOR_MASTER: SendNextPFM (b)

Test Steps:

- 1. MAKE (IUT turned off or otherwise reset)
- 2. WHILE (IUT not initialized) DO {
 - TRANSMIT
 - Poll For Master
 - IF (IUT is turned off) THEN
 - MAKE (IUT turned on or otherwise started)
- 3. RECEIVE
 - Reply To Poll For Master
- 4. TRANSMIT
 - Token
- 5. WHILE (received frame other than Poll For Master) DO {
 - IF (frame is BACnet Data Expecting Reply) THEN
 - TRANSMIT
 - SA = (DA of received frame),
 - BACnet Data Not Expecting Reply,
 - 'Header CRC' = (any incorrect value)
- 6. RECEIVE
 - DA = IUT+1,
 - Poll For Master
- 7. TRANSMIT
 - SA = IUT+1,
 - DA = IUT,
 - Reply To Poll For Master
 - 'Header CRC' = (any invalid value)
- 8. RECEIVE
 - DA = IUT+2,
 - Poll For Master

12.1.1.4.5 Token Received and Passed

Purpose: To verify the passing of a token to a master with a MAC address other than one count higher than the IUT MAC address. Depending upon the implementation and setup of the IUT, there are three possibilities in the sequence of verified Master Node Finite State Machine transitions:

In all cases:

PASS_TOKEN:	SawTokenUser
IDLE:	ReceivedToken

Case1: Nothing is sent.

USE_TOKEN:	NothingToSend

Case 2: A frame not expecting a reply is to be sent:

USE_TOKEN:	SendNoWait

This could repeat once by the following transition that returns to the USE_TOKEN state.

DONE_WITH_TOKEN:	SendAnotherFrame

Case 3: A frame expecting a reply is sent:

USE_TOKEN:	SendAndWait

WAIT_FOR_REPLY:	ReceivedPostpone

This could repeat once by the following transition that returns to the USE_TOKEN state.

DONE_WITH_TOKEN:	SendAnotherFrame

In all cases:

DONE_WITH_TOKEN:	SendToken (a)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 - TRANSMIT
 - SA = 6,
 - Poll For Master
 - IF (IUT is turned off) THEN
 - MAKE (IUT turned on or otherwise started)
3. RECEIVE
 - DA = 6,
 - Reply To Poll For Master
4. TRANSMIT
 - SA = 6,
 - Token
5. WHILE (received other than Token frame) DO {
 - IF (frame is BACnet Data Expecting Reply) THEN
 - TRANSMIT
 - SA = (DA of received frame),
 - BACnet Data Not Expecting Reply,
 - 'Header CRC' = (any incorrect value)
6. RECEIVE
 - DA = 6,
 - Token

12.1.1.4.6 Done Polling - No Reply

Purpose: To verify the passing of a token upon the completion of a cycle of polling for masters when there was no reply for the most recent Poll For Master frame. Depending upon the implementation and setup of the IUT, there are three possibilities in the sequence of verified Master Node Finite State Machine transitions, with two other choices at the end of the testing loop:

In all cases:

PASS_TOKEN:	SawTokenUser
IDLE:	ReceivedToken

Case1: Nothing is sent.

USE_TOKEN:	NothingToSend

Case 2: A frame not expecting a reply is to be sent:

USE_TOKEN:	SendNoWait

This could repeat once by the following transition that returns to the USE_TOKEN state:

DONE_WITH_TOKEN:	SendAnotherFrame

Case 3: A frame expecting a reply is sent:

USE_TOKEN:	SendAndWait

WAIT_FOR_REPLY:	ReceivedReply

This could repeat once by the following transition that returns to the USE_TOKEN state:

DONE_WITH_TOKEN:	SendAnotherFrame

In all cases, either:

DONE_WITH_TOKEN:	SendToken (a)

Or:

DONE_WITH_TOKEN:	SendMaintenancePFM

POLL_FOR_MASTER:	DoneWithPFM (a)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 - TRANSMIT
 - SA = 6,
 - Test Request
 - IF (IUT is turned off) THEN
 - MAKE (IUT turned on or otherwise started)
3. RECEIVE
 - DA = 6,
 - Test Response
4. REPEAT I = (1 to 49) DO {
 - TRANSMIT
 - DA = 127,
 - SA = 6,
 - Poll For Master
 - WAIT $T_{usage\_timeout}$
 - TRANSMIT
 - SA = 6,
 - Token
 - WHILE (received other than Token frame) DO {
 - IF (frame is BACnet Data Expecting Reply) THEN
 - TRANSMIT
 - SA = (DA of received frame),
 - BACnet Data Not Expecting Reply,
 - 'Header CRC' = (any incorrect value)

```

        RECEIVE
        DA = 6,
        Token
    }
5.  TRANSMIT
    DA = 127,
    SA = 6,
    Poll For Master
6.  WAIT Tusage_timeout
7.  TRANSMIT
    SA = 6,
    Token
8.  WHILE (received other than Token frame) DO {
    IF (frame is BACnet Data Expecting Reply) THEN
        TRANSMIT
        SA = (DA of received frame),
        BACnet Data Not Expecting Reply,
        'Header CRC' = (any incorrect value)
    }
9.  RECEIVE
    DA = 3,
    Poll For Master
10. WAIT Tusage_timeout
11. RECEIVE
    DA = 6,
    Token
    
```

12.1.1.4.7 Done Polling - Invalid Reply

Purpose: To verify the passing of a token upon the completion of a cycle of polling for masters when there was an invalid reply for the most recent Poll For Master frame. The sequence of verified transitions is the same as in 12.1.1.4.6 except for the final transition:

POLL_FOR_MASTER: DoneWithPFM(b)

Test Steps:

```

1.  MAKE (IUT turned off or otherwise reset)
2.  WHILE (IUT not initialized) DO {
    TRANSMIT
    SA = 6,
    Test Request
    IF (IUT is turned off) THEN
        MAKE (IUT turned on or otherwise started)
    }
3.  RECEIVE
    DA = 6,
    Test Response
4.  REPEAT I = (1 to 49) DO {
    TRANSMIT
    DA = 127,
    SA = 6,
    Poll For Master
    WAIT Tusage_timeout
    TRANSMIT
    SA = 6,
    Token
    WHILE (received other than Token frame) DO {
    IF (frame is BACnet Data Expecting Reply) THEN
        TRANSMIT
        SA = (DA of received frame),
    }
    }
    
```

BACnet Data Not Expecting Reply,
'Header CRC' = (any incorrect value)

```

    }
    RECEIVE
      DA = 6,
      Token
    }
5. TRANSMIT
  DA = 127,
  SA = 6,
  Poll For Master
6. WAIT Tusage_timeout
7. TRANSMIT
  SA = 6,
  Token
8. WHILE (received other than Token frame) DO {
  IF (frame is BACnet Data Expecting Reply) THEN
    TRANSMIT
      SA = (DA of received frame),
      BACnet Data Not Expecting Reply,
      'Header CRC' = (any incorrect value)
  }
9. RECEIVE
  DA = 3,
  Poll For Master
10. WAIT Tusage_timeout
11. RECEIVE
  DA = 6,
  Token
12. REPEAT I = (1 to 49) DO {
  TRANSMIT
    DA = 127,
    SA = 6,
    Poll For Master
  WAIT Tusage_timeout
  TRANSMIT
    SA = 6,
    Token
  WHILE (received other than Token frame) DO {
    IF (frame is BACnet Data Expecting Reply) THEN
      TRANSMIT
        SA = (DA of received frame),
        BACnet Data Not Expecting Reply,
        'Header CRC' = (any incorrect value)
    }
  RECEIVE
    DA = 6,
    Token
  }
13. TRANSMIT
  DA = 127,
  SA = 6,
  Poll For Master
14. WAIT Tusage_timeout
15. TRANSMIT
  SA = 6,
  Token
16. WHILE (received other than Token frame) DO {
  IF (frame is BACnet Data Expecting Reply) THEN
    TRANSMIT

```

```

        SA = (DA of received frame),
        BACnet Data Not Expecting Reply,
        'Header CRC' = (any incorrect value)
    }
17. RECEIVE
    DA = 4,
    Poll For Master
18. TRANSMIT
    SA=4,
    Reply To Poll For Master,
    'Header CRC' = (any invalid value)
19. RECEIVE
    DA = 6,
    Token

```

12.1.1.4.8 Reset Poll For Master

Purpose: To verify the ResetMaintenancePFM transition that takes place when the MAC address one less than the next known master's MAC address has been polled. The sequence of verified transitions is the same as in 12.1.1.4.7 except that the final transition is:

DONE_WITH_TOKEN: ResetMaintenancePFM

Test Steps:

```

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
    TRANSMIT
    SA = 6,
    Test Request
    IF (IUT is turned off) THEN
        MAKE (IUT turned on or otherwise started)
    }
3. RECEIVE
    DA = 6,
    Test Response
4. REPEAT M = (3 to 6) DO {
    REPEAT I = (1 to 49) DO {
        TRANSMIT
        DA = 127,
        SA = 6,
        Poll For Master
        WAIT Tusage_timeout
        TRANSMIT
        SA = 6,
        Token
        WHILE (received other than Token frame) DO {
            IF (frame is BACnet Data Expecting Reply) THEN
                TRANSMIT
                SA = (DA of received frame),
                BACnet Data Not Expecting Reply,
                'Header CRC' = (any incorrect value)
            }
        }
        RECEIVE
        DA = 6,
        Token
    }
    TRANSMIT
    DA = 127,
    SA = 6,
    Poll For Master

```

```

WAIT Tusage_timeout
TRANSMIT
  SA = 6,
  Token
WHILE (received other than Token frame) DO {
  IF (frame is BACnet Data Expecting Reply) THEN
    TRANSMIT
      SA = (DA of received frame),
      BACnet Data Not Expecting Reply,
      'Header CRC' = (any incorrect value)
  }
IF (M = 6) THEN
  RECEIVE
    DA = 3,
    Poll For Master
ELSE
  RECEIVE
    DA = M,
    Poll For Master
}

```

12.1.1.4.9 Next Master Disappeared

Purpose: To verify that the IUT correctly resumes polling for masters at the MAC address one greater than the last known "next master's" MAC address when that master does not receive or use the token passed to it. This tests the transitions:

PASS_TOKEN:	RetrySendToken
PASS_TOKEN:	FindNewSuccessor

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {


```

TRANSMIT
  SA = 6,
  Poll For Master
IF (IUT is turned off) THEN
  MAKE (IUT turned on or otherwise started)
      
```
3. RECEIVE


```

DA = 6,
Reply To Poll For Master
      
```
4. TRANSMIT


```

SA = 6,
Token
      
```
5. WHILE (received other than Token frame) DO {


```

IF (frame is BACnet Data Expecting Reply) THEN
  TRANSMIT
    SA = (DA of received frame),
    BACnet Data Not Expecting Reply,
    'Header CRC' = (any incorrect value)
      
```
6. RECEIVE


```

DA = 6,
Token
      
```
7. WAIT T_{usage_timeout}
8. RECEIVE


```

DA = 7,
Poll For Master
      
```

12.1.1.4.10 Reply To Poll For Master Frame - Incorrect Destination

Purpose: To verify that the IUT correctly transitions to the IDLE state when a Reply To Poll For Master frame with the wrong Destination Address is received during a Poll For Master. This tests the transitions:

POLL_FOR_MASTER: ReceivedUnexpectedFrame (a)
IDLE: ReceivedPFM

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 SA = 6,
 Poll For Master
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 DA = 6,
 Reply To Poll For Master
4. TRANSMIT
 SA = 6,
 Token
5. WHILE (received other than Token frame) DO {
 IF (frame is BACnet Data Expecting Reply) THEN
 TRANSMIT
 SA = (DA of received frame),
 BACnet Data Not Expecting Reply,
 'Header CRC' = (any incorrect value)
 }
6. RECEIVE
 DA = 6,
 Token
7. WAIT $T_{usage\_timeout}$
8. RECEIVE
 DA = 7,
 Poll For Master
9. TRANSMIT
 DA = (value less than 128 and other than IUT, 6 or 7),
 Reply To Poll For Master
10. TRANSMIT
 SA = 0,
 Poll For Master
11. RECEIVE
 DA = 0,
 Reply To Poll For Master

12.1.1.4.11 Generate Token

Purpose: To verify that the IUT generates a token when it has been lost. This tests the transitions:

IDLE: LostToken
NO_TOKEN: GenerateToken

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 SA = 6,
 Poll For Master
 IF (IUT is turned off) THEN

- ```

 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 DA = 6,
 Reply To Poll For Master
4. RECEIVE
 DA = IUT+1,
 Poll For Master

```

#### 12.1.1.4.12 Poll For Master - Incorrect Response

Purpose: To verify that the IUT correctly transitions to the IDLE state when a frame other than Reply To Poll For Master is received during a Poll For Master. This tests the transitions:

POLL\_FOR\_MASTER: ReceivedUnexpectedFrame (b)  
 IDLE: Received PFM

Test Steps:

- ```

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
   TRANSMIT
   SA = 6,
   Poll For Master
   IF (IUT is turned off) THEN
     MAKE (IUT turned on or otherwise started)
   }
3. RECEIVE
   DA = 6,
   Reply To Poll For Master
4. RECEIVE
   Poll For Master
5. TRANSMIT
   Test Request
6. TRANSMIT
   SA = 0,
   Poll For Master
7. RECEIVE
   DA = 0,
   Reply To Poll For Master

```

12.1.1.4.13 SawFrame

Purpose: To verify that after a token has been dropped and a new one created by another device, the IUT observes the creation of the token.

Test Steps:

- ```

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 SA = 6,
 Poll For Master
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 DA = 6,
 Reply To Poll For Master
4. WAIT (Tno_token + Tslot)
5. TRANSMIT
 DA = 127,

```



- ```

        SA = 1,
        Poll For Master
6.  BEFORE (Tno_token + Tslot) {
        IF (anything received) THEN
            ERROR "Token incorrectly generated by IUT."
        }
7.  TRANSMIT
        SA = 0,
        Poll For Master
8.  RECEIVE
        DA = 0,
        Reply To Poll For Master
    
```

12.1.1.5 Tests to Verify Answer Data Request

This clause describes two tests that verify the proper operations of the transitions associated with the ANSWER_DATA_REQUEST state. Since the choice between Reply and Deferred Reply is a matter internal to the IUT, it may not be possible to ensure a complete test. If the choice is reliably determined by external factors, such as the choice of property to be read, these tests shall be performed twice, once for immediate replies and once for postponed, to verify all transitions. Let X be the instance number of the Device Object for the IUT.

Only one of these two tests needs to be passed.

12.1.1.5.1 Answer Data Request

Purpose: To verify a correct response direct to a BACnet Data Expecting Reply frame. The transitions verified are:

IDLE:	ReceivedDataNeedingReply
ANSWER_DATA_REQUEST:	Reply

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {


```

                TRANSMIT
                Test Request
                IF (IUT is turned off) THEN
                    MAKE (IUT turned on or otherwise started)
            
```
3. RECEIVE


```

                Test Response
            
```
4. TRANSMIT


```

                ReadProperty-Request,
                'Frame Type' = BACnet Data Expecting Reply,
                'ObjectIdentifier' = (Device, X),
                'PropertyIdentifier' = Object_Identifier
            
```
5. RECEIVE


```

                ReadProperty-ACK,
                'Frame Type' = BACnet Data Not Expecting Reply,
                'ObjectIdentifier' = (Device, X),
                'PropertyIdentifier' = Object_Identifier,
                'Data' = (Device, X)
            
```

12.1.1.5.2 Deferred Reply

This test is only performed if the response from the IUT in 12.1.1.5.1 was a Reply Postponed frame.

Purpose: To verify the actual response if a Reply Postponed frame is received. Transitions tested:

IDLE:	ReceivedDataNeedingReply
ANSWER_DATA_REQUEST:	Deferred Reply

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 - TRANSMIT
 - Test Request
 - IF (IUT is turned off) THEN
 - MAKE (IUT turned on or otherwise started)
3. RECEIVE
 - Test Response
4. TRANSMIT
 - ReadProperty-Request,
 - 'Frame Type' = BACnet Data Expecting Reply,
 - 'ObjectIdentifier' = (Device, X),
 - 'PropertyIdentifier' = Object_Identifier
5. RECEIVE
 - Reply Postponed
6. WHILE (received frame other than BACnet Data Not Expecting Reply) DO {
 - IF (frame type is Poll For Master) THEN
 - TRANSMIT
 - Reply To Poll For Master
 - ELSE
 - IF (frame type is Token) THEN
 - TRANSMIT
 - Token
7. RECEIVE
 - ReadProperty-ACK,
 - 'Frame Type' = BACnet Data Not Expecting Reply,
 - 'ObjectIdentifier' = (Device, X),
 - 'PropertyIdentifier' = Object_Identifier,
 - 'Data' = (Device, X)

12.1.1.6 Miscellaneous Non-Response Tests

The tests described in the clause shall be used to verify various elemental operations that do not result in a response from the IUT. Verification is performed by a test demonstrating that the Master Node Finite State Machine is still in the IDLE state. Let X be the instance number of the Device Object for the IUT.

12.1.1.6.1 Received Data No Reply

Purpose: To verify that the Master Node Finite State Machine in the IDLE state properly handles a BACnet Data Not Expecting Reply frame. Transitions verified:

IDLE:	ReceivedDataNoReply
IDLE:	Received PFM

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 - TRANSMIT
 - Test Request
 - IF (IUT is turned off) THEN
 - MAKE (IUT turned on or otherwise started)
3. RECEIVE
 - Test Response
4. TRANSMIT
 - TimeSynchronization-Request,
 - 'Frame Type' = BACnet Data Not Expecting Reply,

5. BEFORE (T_{reply_delay}) {
 IF (anything received) THEN
 ERROR "Response issued by IUT to BACnet Data Not Expecting Reply."
 }
6. TRANSMIT
 Poll For Master
7. RECEIVE
 Reply To Poll For Master

12.1.1.6.2 Received Invalid Frame

Purpose: To verify that the Master Node Finite State Machine in the IDLE state properly handles an invalid frame.
 Transitions verified:

IDLE:	ReceivedInvalidFrame
IDLE:	Received PFM

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Test Request
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 Test Response
4. TRANSMIT
 Poll For Master,
 'Header CRC' = (any incorrect value)
5. BEFORE (T_{reply_delay}) {
 IF (frame received) THEN
 ERROR "Invalid Frame accepted by IUT."
 }
6. TRANSMIT
 Poll For Master
7. RECEIVE
 Reply To Poll For Master

12.1.1.6.3 Unwanted Frame Tests

The tests described by this clause verify that the IUT in its IDLE state rejects frames not addressed to it, or frames that were illegally broadcast. Clause 12.1.1.6.3.1 verifies a transition that should never occur since the Receive Frame State Machine rejects messages for other devices in its NotForUs transition. Transitions verified:

IDLE:	ReceivedUnwantedFrame
IDLE:	Received PFM

12.1.1.6.3.1 Not Our Address

Purpose: To verify that the IUT in the IDLE state properly handles a frame addressed to another device.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Test Request
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }

3. RECEIVE
Test Response
4. TRANSMIT
DA = (value less than 128 and other than IUT or TD),
Poll For Master
5. BEFORE (T_{reply_delay}) {
IF (frame received) THEN
ERROR "IUT accepted frame addressed to other device."
}
6. TRANSMIT
Poll For Master
7. RECEIVE
Reply To Poll For Master

12.1.1.6.3.2 Broadcast Token Frame

Purpose: To verify that the IUT in the IDLE state properly handles a broadcast Token frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Test Request
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Test Response
4. TRANSMIT
DA = LOCAL BROADCAST,
Token
5. BEFORE (T_{reply_delay}) {
IF (frame received) THEN
ERROR "Broadcast Token frame accepted by IUT."
}
6. TRANSMIT
Poll For Master
7. RECEIVE
Reply To Poll For Master

12.1.1.6.3.3 Broadcast BACnet Data Expecting Reply Frame

Purpose: To verify that the IUT in the IDLE state properly handles a broadcast BACnet Data Expecting Reply frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Test Request
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Test Response
4. TRANSMIT
DA = LOCAL BROADCAST,
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'ObjectIdentifier' = (Device, X),
'PropertyIdentifier' = Object_Identifier

5. BEFORE (T_{reply_delay}) {
 IF (frame received) THEN
 ERROR "Broadcast BACnet Data Expecting Reply frame accepted by IUT."
 }
6. TRANSMIT
 Poll For Master
7. RECEIVE
 Reply To Poll For Master

12.1.1.6.3.4 Broadcast Test Request Frame

Purpose: To verify that the IUT in the IDLE state properly handles a broadcast Test Request frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Test Request
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 Test Response
4. TRANSMIT
 DA = LOCAL BROADCAST,
 Test Request
5. BEFORE (T_{reply_delay}) {
 IF (frame received) THEN
 ERROR "Broadcast Test Request frame accepted by IUT."
 }
6. TRANSMIT
 Poll For Master
7. RECEIVE
 Reply To Poll For Master

12.1.1.7 Sole Master Tests

These tests verify the ability of the IUT to properly recognize itself to be the only master on the MS/TP LAN, and to identify when another master enters.

12.1.1.7.1 Drop Token

Purpose: To verify that the IUT recognizes that the token has been lost and generates another. This tests the transitions:

IDLE:	LostToken
NO_TOKEN:	GenerateToken

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
 TRANSMIT
 Poll For Master
 IF (IUT is turned off) THEN
 MAKE (IUT turned on or otherwise started)
 }
3. RECEIVE
 Reply To Poll For Master
4. BEFORE ($T_{no\_token} + (2 * T_{slot})$) {
 IF (frame received) THEN
 ERROR "Lost Token detected too early by IUT."
 }

5. BEFORE (T_{slot})
RECEIVE
DA = IUT+1,
Poll For Master

12.1.1.7.2 Poll For Next Master

Purpose: To verify that the IUT holds the token and is conducting a poll to find another master. This verifies the transition:

POLL_FOR_MASTER: SendNextPFM

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Poll For Master
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Reply To Poll For Master
4. RECEIVE
DA = IUT+1,
Poll For Master
5. BEFORE ($T_{usage\_timeout}$) {
IF (frame received) THEN
ERROR "IUT didn't wait long enough for Reply To Poll For Master."
}
6. RECEIVE
DA = IUT+2,
Poll For Master

12.1.1.7.3 More Polls

Purpose: To verify that the IUT checks all remaining MAC addresses in its polling to find another master. In the final step this causes the transition:

POLL_FOR_MASTER: DeclareSoleMaster (a)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Poll For Master

IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Reply To Poll For Master
4. RECEIVE
DA = IUT+1,
Poll For Master
5. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {
BEFORE ($T_{usage\_timeout}$) {
IF (frame received) THEN
ERROR "IUT didn't wait long enough for Reply To Poll For Master."
}
RECEIVE
DA = X,

```

    Poll For Master
  }

```

12.1.1.7.4 Declare Sole Master (a)

Purpose: To verify that the IUT has declared itself the sole master but is still conducting a poll to find another master. This verifies the transitions:

```

    DONE_WITH_TOKEN:      Solemaster (a)
    DONE_WITH_TOKEN:      SendMaintenancePFM

```

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {


```

        TRANSMIT
        Poll For Master
        IF (IUT is turned off) THEN
          MAKE (IUT turned on or otherwise started)
      
```
3. RECEIVE


```

        Reply To Poll For Master
      
```
4. RECEIVE


```

        DA = IUT+1,
        Poll For Master
      
```
5. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {


```

        BEFORE (Tusage_timeout) {
          IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
          }
        RECEIVE
        DA = X,
        Poll For Master
      
```
6. BEFORE (T_{usage_timeout}) {


```

        IF (frame received) THEN
          ERROR "IUT didn't wait long enough for Reply To Poll For Master."
      
```
7. RECEIVE


```

        DA = IUT+1,
        Poll For Master
      
```

12.1.1.7.5 New Master Enters

Purpose: To verify that the IUT recognizes the presence of another master.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {


```

        TRANSMIT
        Poll For Master
        IF (IUT is turned off) THEN
          MAKE (IUT turned on or otherwise started)
      
```
3. RECEIVE


```

        Reply To Poll For Master
      
```
4. RECEIVE


```

        DA = IUT+1,
        Poll For Master
      
```
5. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {


```

        BEFORE (Tusage_timeout) {

```



```

        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
    RECEIVE
        DA = X,
        Poll For Master
    }
6. BEFORE (Tusage_timeout) {
    IF (frame received) THEN
        ERROR "IUT didn't wait long enough for Reply To Poll For Master."
    }
7. RECEIVE
    DA = IUT+1,
    Poll For Master
8. TRANSMIT
    SA = IUT+1,
    Reply To Poll For Master
9. RECEIVE
    DA = IUT+1,
    Token

```

12.1.1.7.6 Poll For Next Master

Purpose: To verify that the IUT holds the Token and is conducting a poll to find another master. This verifies the transition:

POLL_FOR_MASTER: SendNextPFM

Test Steps:

```

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
    TRANSMIT
        Poll For Master
    IF (IUT is turned off) THEN
        MAKE (IUT turned on or otherwise started)
    }
3. RECEIVE
    Reply To Poll For Master
4. RECEIVE
    DA = IUT+1,
    Poll For Master
5. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {
    BEFORE (Tusage_timeout) {
        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
    RECEIVE
        DA = X,
        Poll For Master
    }
6. BEFORE (Tusage_timeout) {
    IF (frame received) THEN
        ERROR "IUT didn't wait long enough for Reply To Poll For Master."
    }
7. RECEIVE
    DA = IUT+1,
    Poll For Master
8. TRANSMIT
    SA = IUT+1,
    Reply To Poll For Master

```

9. RECEIVE
DA = IUT+1
Token
10. BEFORE ($T_{usage\_timeout}$) {
IF (frame received) THEN
ERROR "IUT didn't wait long enough for Reply To Poll For Master."
}
11. RECEIVE
DA = IUT+2,
Poll For Master

12.1.1.7.7 DeclareSoleMaster (b)

Purpose: To verify that the IUT undergoes the transitions necessary for it to declare itself the sole master when it receives an invalid frame in response to the transmitted Poll For Master Frame. The transitions tested are:

POLL_FOR_MASTER:	SendNextPFM
POLL_FOR_MASTER:	DeclareSoleMaster (b)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Poll For Master
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Reply To Poll For Master
4. RECEIVE
DA = IUT+1,
Poll For Master
5. TRANSMIT
Reply To Poll For Master
SA = IUT+1,
'Header CRC' = (any incorrect value)
6. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {
RECEIVE
DA = X,
Poll For Master
TRANSMIT
SA = X,
Reply To Poll For Master,
'Header CRC' = (any incorrect value)
}

12.1.1.7.8 SoleMaster (b)

Purpose: To verify that the IUT has declared itself to be the sole master and is continuing to poll for other masters. Depending upon the implementation and setup of the IUT, there are three possibilities in the sequence of verified Master Node Finite State Machine transitions, with two other choices at the end of the testing loop:

1. Nothing is sent.
USE_TOKEN: NothingToSend
2. A frame not expecting a reply is to be sent:
USE_TOKEN: SendNoWait
This could repeat once by the following transition that returns to the beginning of this step:
DONE_WITH_TOKEN: SendAnotherFrame
3. A frame expecting a reply is sent:
USE_TOKEN: SendAndWait

In all cases:

WAIT_FOR_REPLY: ReceivedReply

The first 49 times:

DONE_WITH_TOKEN: SoleMaster

The 50th time:

DONE_WITH_TOKEN: SendMaintenancePFM

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Poll For Master
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Reply To Poll For Master
4. RECEIVE
DA = IUT+1,
Poll For Master
5. TRANSMIT
Reply To Poll For Master,
SA = IUT+1,
'Header CRC' = (any incorrect value)
6. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {
RECEIVE
DA = X,
Poll For Master
TRANSMIT
SA = X,
Reply To Poll For Master,
'Header CRC' = (any incorrect value)
}
7. RECEIVE
DA = IUT+1,
Poll For Master

12.1.1.7.9 Get Token

Purpose: To verify that the IUT properly responds to the entry of another master. The following transition is verified:

POLL_FOR_MASTER: ReceivedReplyToPFM

1. MAKE (IUT turned off or otherwise reset)
2. WHILE (IUT not initialized) DO {
TRANSMIT
Poll For Master
IF (IUT is turned off) THEN
MAKE (IUT turned on or otherwise started)
}
3. RECEIVE
Reply To Poll For Master
4. RECEIVE
DA = IUT+1,
Poll For Master
5. TRANSMIT
Reply To Poll For Master,
SA = IUT+1,
'Header CRC' = (any incorrect value)

6. REPEAT X=(IUT+2 to 127, 0 to IUT-1) {
 - RECEIVE
 - DA = X,
 - Poll For Master
 - TRANSMIT
 - SA = X,
 - Reply To Poll For Master,
 - 'Header CRC' = (any incorrect value)
7. RECEIVE
 - DA = IUT+1,
 - Poll For Master
8. TRANSMIT
 - SA = IUT+1,
 - Reply To Poll For Master
9. RECEIVE
 - DA = IUT+1,
 - Token

12.1.1.8 Multiple Tokens Detected During Confirmed Service Request

This clause defines tests that are only performed if the IUT is able to periodically initiate a confirmed service request such as a ReadProperty request. Each test waits until the IUT generates a token and uses it, then the TD transmits an invalid frame type (indicating the presence of another token), causing the IUT to re-enter its IDLE state, which is then tested. These test the conditionals of the transition:

WAIT_FOR_REPLY: ReceivedUnexpectedFrame(a,b)

The IUT should be set up to transmit a repeated confirmed service request (i.e., ReadProperty request) with a destination MAC address of 3 or higher.

Let X be the instance number of the Device Object for the IUT.

12.1.1.8.1 Different Destination

Purpose: To detect a second token when a message to a different MAC address appears out of turn.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
 - Test_Request
4. RECEIVE
 - Test_Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
 -
7. TRANSMIT
 - DA = (value less than 128 and other than IUT or TD),
 - BACnet Data Not Expecting Reply
8. WAIT T_{reply_timeout}
9. TRANSMIT
 - SA = 0,
 - DA = IUT,
 - Poll For Master
10. RECEIVE
 - DA = 0,
 - SA = IUT,
 - Reply To Poll For Master

12.1.1.8.2 Broadcast

Purpose: To detect a second token when a broadcast message appears out of turn.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
 }
7. TRANSMIT
 DA = LOCAL BROADCAST,
 BACnet Data Not Expecting Reply
8. WAIT $T_{reply\_timeout}$
9. TRANSMIT
 SA = 0,
 DA = IUT,
 Poll For Master
10. RECEIVE
 DA = 0,
 SA = IUT,
 Reply To Poll For Master

12.1.1.8.3 Token

Purpose: To detect a second token when it gets passed to the IUT already holding a token.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
 }
7. TRANSMIT
 Token
8. WAIT $T_{reply\_timeout}$
9. TRANSMIT
 SA = 0,
 DA = IUT,
 Poll For Master
10. RECEIVE
 DA = 0,
 SA = IUT,
 Reply To Poll For Master

12.1.1.8.4 Poll For Master

Purpose: To detect a second token when the IUT with a token is polled.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)

3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
 }
7. TRANSMIT
 SA = 0,
 DA = IUT,
 Poll For Master
8. WAIT $T_{reply\_timeout}$
9. TRANSMIT
 SA = 0,
 DA = IUT,
 Poll For Master
10. RECEIVE
 DA = 0,
 SA = IUT,
 Reply To Poll For Master

12.1.1.8.5 Reply To Poll For Master

Purpose: To detect a protocol problem when an incorrect reply is returned.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
 }
7. TRANSMIT
 SA = 0,
 DA = IUT,
 Reply To Poll For Master
8. WAIT $T_{reply\_timeout}$
9. TRANSMIT
 SA = 0,
 DA = IUT,
 Poll For Master
10. RECEIVE
 DA = 0,
 SA = IUT,
 Reply To Poll For Master

12.1.1.8.6 Test Request

Purpose: To detect a second token when the IUT with a token receives a Test_Request frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response

5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {
}
7. TRANSMIT
SA = 0,
DA = IUT,
Test Request
8. WAIT $T_{\text{reply\_timeout}}$
9. TRANSMIT
SA = 0,
DA = IUT,
Poll For Master
10. RECEIVE
DA = 0,
SA = IUT,
Reply To Poll For Master

12.1.1.8.7 BACnet Data Expecting Reply

Purpose: To detect a second token when the IUT with a token is polled.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test Request
4. RECEIVE
Test Response
5. MAKE (IUT to generate a confirmed service request)
6. WHILE (BACnet Data Expecting Reply frame not received) DO {}
7. TRANSMIT
SA = 0,
DA = IUT,
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'ObjectIdentifier' = (Device, X),
'PropertyIdentifier' = Object_Identifier
8. WAIT $T_{\text{reply\_timeout}}$
9. TRANSMIT
SA = 0,
DA = IUT,
Poll For Master
10. RECEIVE
DA = 0,
SA = IUT,
Reply To Poll For Master

12.1.1.9 Token Usage Tests

This clause defines tests that are only performed if the IUT is able to periodically initiate a confirmed service request such as a ReadProperty request, or to initiate an unconfirmed service request such as an I-Am request either periodically or in response to another request.

If it can, the IUT should be set up to transmit a repeated confirmed service request (i.e., ReadProperty request) with a destination MAC address of 3 or higher. It should also be set up to transmit unconfirmed service requests.

If the Max_Info_Frames property of the Device object of the IUT is alterable, it should initially be set to 1 and the Number_of_APDU_Retries shall have a value of 2 or greater.

12.1.1.9.1 Unconfirmed Request

This test shall be performed only if the IUT supports the Who-Is or Who-Has unconfirmed services, responding with the I-Am and I-Have services, accordingly.

Purpose: To verify the correct initiation of unconfirmed service requests by the Master Node Finite State Machine. This verifies the transitions:

IDLE	LostToken
NO_TOKEN	GenerateToken
POLL_FOR_MASTER	SendNextPFM
IDLE	ReceivedDataNoReply
USE_TOKEN	SendNoWait

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. WHILE (no Token frame received) DO {
 - IF (Poll For Master frame received with DA set to TD) THEN
 - TRANSMIT
 - Reply To Poll For Master
4. TRANSMIT
 - DA = LOCAL BROADCAST,
 - Who-Is-Request
5. TRANSMIT
 - Token
6. RECEIVE
 - I-Am-Request

Notes to Tester: In these steps, if the Who-Is service is not supported, substitute the Who-Has and I-Have services.

12.1.1.9.2 Confirmed Request With Reply

This test shall be performed only if the IUT is able to initiate confirmed service requests.

Purpose: To verify the correct initiation of confirmed service requests by the Master Node Finite State Machine. This verifies the transitions:

USE_TOKEN:	SendAndWait
WAIT_FOR_REPLY:	ReceivedReply

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. MAKE (IUT initiate ReadProperty-Request to TD)
4. WHILE (ReadProperty-Request with DA set to TD not received) DO {
 - IF (Poll For Master frame received with DA set to TD) THEN
 - TRANSMIT
 - Reply To Poll For Master
 - IF (Token frame received with DA set to TD not received) THEN
 - TRANSMIT
 - Token
5. RECEIVE
 - ReadProperty-Request,
 - 'Frame Type' = BACnet Data Expecting Reply,
6. TRANSMIT
 - ReadProperty-ACK

```

7. BEFORE (APDU_Timeout) {
    IF (Poll For Master frame received with DA set to TD) THEN
        TRANSMIT
        Reply To Poll For Master
    IF (Token frame received with DA set to TD not received) THEN
        TRANSMIT
        Token
    IF (ReadProperty-Request identical to step 5 received) THEN
        ERROR "Confirmed Service ACK not understood by IUT."
}

```

Notes to Tester: The ReadProperty-ACK of step 6 shall be an appropriate and correct response to the request of step 5.

12.1.1.9.3 Confirmed Request - No Reply

This test shall be performed only if the IUT is able to initiate confirmed service requests.

Purpose: To verify the correct termination of a confirmed service request by the Master Node Finite State Machine when no reply is received. This verifies the transitions:

USE_TOKEN:	SendAndWait
WAIT_FOR_REPLY:	ReplyTimeout

Test Steps:

```

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. MAKE (IUT initiate ReadProperty-Request to TD)
4. REPEAT X = (1 to IUT APDU_Retry_Count) DO {
    BEFORE (APDU_Timeout) {
        IF (ReadProperty-Request with DA set to TD received) THEN
            ERROR "Retry too soon."
        IF (Poll For Master frame received with DA set to TD) THEN
            TRANSMIT
            Reply To Poll For Master
        IF (Token frame received with DA set to TD not received) THEN
            TRANSMIT
            Token
    }
    RECEIVE
    ReadProperty-Request
}
5. BEFORE (APDU_Timeout) {
    IF (ReadProperty-Request with DA set to TD received) THEN
        ERROR "Incorrectly terminated service request."
    IF (Poll For Master frame received with DA set to TD) THEN
        TRANSMIT
        Reply To Poll For Master
    IF (Token frame received with DA set to TD not received) THEN
        TRANSMIT
        Token
}

```

12.1.1.9.4 Confirmed Request - Invalid Reply

This test shall be performed only if the IUT is able to initiate confirmed service requests.

Purpose: To verify the correct handling of an invalid reply frame by the Master Node Finite State Machine. This verifies the transitions:

USE_TOKEN:	SendAndWait
WAIT_FOR_REPLY:	InvalidFrame

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. MAKE (IUT initiate ReadProperty-Request to TD)
4. REPEAT X = (1 to IUT APDU_Retries) DO {
 - BEFORE (APDU_Timeout) {
 - IF (ReadProperty-Request with DA set to TD received) THEN
 - ERROR "Retry too soon."
 - IF (Poll For Master frame received with DA set to TD) THEN
 - TRANSMIT
 - Reply To Poll For Master
 - IF (Token frame received with DA set to TD not received) THEN
 - TRANSMIT
 - Token
 - RECEIVE
 - ReadProperty-Request
 - TRANSMIT
 - ReadProperty-ACK,
 - 'Frame Type' = BACnet Data Expecting Reply,
 - 'Data CRC' = (any incorrect value)
5. BEFORE (APDU_Timeout) {
 - IF (ReadProperty-Request with DA set to TD received) THEN
 - ERROR "Incorrectly terminated service request."
 - IF (Poll For Master frame received with DA set to TD) THEN
 - TRANSMIT
 - Reply To Poll For Master
 - IF (Token frame received with DA set to TD not received) THEN
 - TRANSMIT
 - Token

12.1.1.9.5 Confirmed Request With Reply Postponed

This test shall be performed only if the IUT is able to initiate confirmed service requests.

Purpose: To verify the correct termination of postponed replies by the Master Node Finite State Machine. This verifies the transitions:

USE_TOKEN:	SendAndWait
WAIT_FOR_REPLY:	ReceivedPostponed

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. MAKE (IUT initiate ReadProperty-Request to TD)
4. REPEAT X = (1 to IUT APDU_Retries) DO {
 - BEFORE (APDU_Timeout) {
 - IF (ReadProperty-Request with DA set to TD received) THEN
 - ERROR "Retry too soon."
 - IF (Poll For Master frame received with DA set to TD) THEN
 - TRANSMIT
 - Reply To Poll For Master
 - IF (Token frame received with DA set to TD not received) THEN
 - TRANSMIT
 - Token

```

RECEIVE
  ReadProperty-Request
TRANSMIT
  Reply Postponed
}
5. BEFORE (APDU_Timeout) {
  IF (ReadProperty-Request with DA set to TD received) THEN
    ERROR "Incorrectly terminated service request."
  IF (Poll For Master frame received with DA set to TD) THEN
    TRANSMIT
      Reply To Poll For Master
  IF (Token frame received with DA set to TD not received) THEN
    TRANSMIT
      Token
}

```

12.1.1.9.6 Max Info Frame Check

Purpose: This check verifies that the MS/TP Master Node State Machine does not issue more than Nmax_info_frames information frames between the time the IUT receives a Token and either the time it passes the Token or it initiates a Poll For Master. Unlike tests, checks are not constructed of test steps, but rather conditions that must hold true through the complete testing process. As such, checks are periodically verified during or after the execution of tests.

Configuration Recommendations: If the Max_Info_Frames property of the Device object is configurable, it is recommended that this property be set to its minimum setting for the performance of at least some tests involving the MS/TP port being tested.

Check conditions: Monitor the MS/TP LAN during operations where the IUT would be expected to issue a number of information frames; if the IUT emits more information frames than:

- the configured value for Max_Info_Frames in the interval between receiving and passing the Token (with multiple masters on the LAN), or
- the configured value for Max_Info_Frames in the interval between receiving the Token and issuing PFM (with multiple masters on the LAN), or
- the configured value for Max_Info_Frames in the interval between any two consecutive Poll For Master frames except the interval between the issuance of a Poll For Master to (TS-1) modulo Max_Master and a Poll For Master to (TS+1) modulo Max_Master, (with the IUT as the only master on the LAN), or
- 52 times the configured value for Max_Info_Frames in the interval between a Poll For Master frame issued to (TS-1) modulo Max_Master, and the subsequent Poll For Master frame to (TS+1) modulo Max_Master (with the IUT as the only master on the LAN),

then the IUT shall fail this check.

Note to Tester: The value 52 is used in d) because an error in the MS/TP state machine originally defined in Standard 135-1995 caused the Token to be passed 52 times between Poll For Master cycles, instead of 50 times.

12.1.2 MS/TP Slave Tests

The tests defined in this clause shall be used to verify that a BACnet MS/TP slave device properly implements the Receive Frame Finite State Machine, the SendFrame procedure, and the Slave Node Finite State Machine. One state transition, ANSWER-DATA REQUEST: CannotReply, is untestable.

Some transitions of the Slave Node Finite State Machine are impossible to verify due to the existence of parallel transitions and the inability of a TD to invoke a specific transition. The IUT shall be considered to be in conformance with BACnet if any of the parallel transitions are made as observed in these tests.

These tests shall be performed at every baud rate supported by the IUT with only the IUT and TD present on the MS/TP LAN.

12.1.2.1 State Machine Transition Tests for Normal Confirmed and Unconfirmed Service Requests

This clause defines the test cases necessary to demonstrate correct operation of the Receive Frame state machine and the Slave Node state machine under circumstances where confirmed and unconfirmed service requests are received and processed without errors. Let X be the instance number of the Device Object for the IUT.

12.1.2.1.1 Confirmed Service Request Transitions

Purpose: To verify that the IUT can receive and understand properly formed frames, and create a properly formed frame in response. The following state machine transitions are verified by this test:

Slave Node:

INITIALIZING:	DoneInitializing
IDLE:	ReceivedDataNeedingReply
ANSWER DATA REQUEST:	Reply

Receive Frame:

IDLE:	Preamble1
PREAMBLE:	Preamble2
HEADER:	FrameType, Destination, Source, Length1, Length2, HeaderCRC
HEADER CRC:	Data (destination address = TS)
DATA:	DataOctet, CRC1, CRC2
DATA CRC:	GoodCRC

This also tests MS/TP frame type 05, BACnet Data Expecting Reply.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
6. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X)

Notes to Tester: In the replies returned in this test, there shall be no inter-frame gaps greater than T_{frame_gap} .

12.1.2.1.2 Directed BACnet Data Not Expecting Reply

Purpose: To verify that there is no reply to directly addressed frames of type BACnet Data Not Expecting Reply and that the Slave Node Finite State Machine remains in the IDLE state. The following Slave Node Finite State Machine transition is verified by this test:

IDLE:	ReceivedDataNoReply

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
(Unconfirmed service supported by IUT),
'Frame Type' = BACnet Data Expecting Reply,

6. TRANSMIT
 ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
7. BEFORE (T_{reply_delay})
 RECEIVE
 ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
 'Property Value' = (Device, X)

Notes to Tester: If the IUT does not support any unconfirmed services this test shall not be performed and the IUT shall be considered to be in conformance to the portion of BACnet tested by this clause.

12.1.2.1.3 Broadcast BACnet Data Not Expecting Reply

Purpose: To verify that there is no reply to broadcast frames of type BACnet Data Not Expecting Reply and that the Slave Node Finite State Machine remains in the IDLE state. The following Slave Node Finite State Machine transition is verified by this test:

HEADER_CRC: Data (destination = 255)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
 Test_Request
4. RECEIVE
 Test_Response
5. TRANSMIT
 DA = LOCAL BROADCAST,
 (Unconfirmed service supported by IUT),
 'Frame Type' = BACnet Data Not Expecting Reply,
6. TRANSMIT
 ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
7. BEFORE (T_{reply_delay})
 RECEIVE
 ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
 'Property Value' = (Device, X)

Notes to Tester: If the IUT does not support any broadcast unconfirmed services this test shall not be performed and the IUT shall be considered to be in conformance to the portion of BACnet tested by this clause.

12.1.2.2 State Machine Transition Tests for Error Transitions

This clause defines the test cases necessary to demonstrate correct operation of the Receive Frame State Machine and the Slave Node Finite State Machine under circumstances where confirmed and unconfirmed service requests are received and data link errors are encountered by the state machine.

12.1.2.2.1 Error Tests with no Response

This clause defines the test cases where the data link errors cause the frame to be discarded and no response is issued. These test cases test the Slave Node Finite State Machine transition:

IDLE: ReceivedInvalidFrame

12.1.2.2.1.1 Bad Data CRC

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with an incorrect Data CRC. The following Receive Frame State Machine transition is verified by this test:

DATA_CRC: BadCRC

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
 'Data CRC' = (any incorrect value),
6. BEFORE (**Acknowledgement Fail Time**) {
 IF (ReadProperty-ACK received) THEN
 ERROR "Incorrect MS/TP Frame Data CRC undetected."
 }
7. TRANSMIT
ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
8. RECEIVE
ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
 'Property Value' = (Device, X)

12.1.2.2.1.2 Data Timeout

Purpose: To verify that the Receive Frame State Machine detects and rejects frames that timeout during data field transmission. The following Receive Frame State Machine transition is verified by this test:

DATA: Timeout

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
6. WAIT $T_{\text{frame\_abort}}$
7. TRANSMIT
ReadProperty-Request,

'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type,

8. RECEIVE

ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type,
 'Property Value' = DEVICE

Notes to Tester: In step 5 of this test, transmission shall be halted after the Header CRC octet has been transmitted but before the final Data CRC octet is transmitted.

12.1.2.2.1.3 Data Framing Error

Purpose: To verify that the Receive Frame State Machine detects and rejects frames in which a transmission framing error occurs in the data. The following Receive Frame State Machine transition is verified by this test:

DATA: Error

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
8. RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Property Value' = DEVICE

Notes to Tester: In step 5 of this test, one octet in the Data field shall be transmitted with a logical zero in the stop bit position.

12.1.2.2.1.4 Bad Header CRC

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with an incorrect Header CRC. The following Receive Frame State Machine transition is verified by this test:

HEADER_CRC: BadCRC

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request

4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
'Header CRC' = (any incorrect value),
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
8. RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Property Value' = DEVICE

12.1.2.2.1.5 Not For Us

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with a destination address different from the IUT address. The following Receive Frame State Machine transition is verified by this test:

HEADER_CRC: NotForUs

Because the Receive Frame State Machine first checks the destination address, the following Slave Node State Machine transition never occurs, though it would otherwise be tested by this test:

IDLE ReceivedUnwantedFrame (a)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
DA = (value less than 128 and other than IUT and TD),
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Service Request' = ReadProperty-Request
8. RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Property Value' = DEVICE

12.1.2.2.1.6 Header Framing Error

Purpose: To verify that the Receive Frame State Machine detects and rejects frames in which a transmission framing error occurs in the Header. The following Receive Frame State Machine transition is verified by this test:

HEADER: Error

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
8. RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Property Value' = DEVICE

Notes to Tester: In the transmission of step 5, one octet in the Header shall be transmitted with a logical zero in the stop bit position.

12.1.2.2.1.7 Header Timeout

Purpose: To verify that the Receive Frame State Machine detects and rejects frames that timeout during header field transmission. The following Receive Frame State Machine transition is verified by this test:

HEADER: Timeout

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,

8. RECEIVE

ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type,
 'Property Value' = DEVICE

Notes to Tester: During the transmission of step 5, the transmission shall be halted after the second Preamble octet is transmitted and before the Header CRC octet is transmitted.

12.1.2.2.1.8 Not Preamble

Purpose: To verify that the Receive Frame State Machine correctly rejects incorrectly formed preambles. The following Receive Frame State Machine transition is verified by this test:

HEADER: NotPreamble

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT

ReadProperty-Request,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type
4. RECEIVE

ReadProperty-ACK,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type,
 'Property Value' = DEVICE
5. TRANSMIT

X'55'
6. TRANSMIT

(octet other than X'55' and X'FF')
7. TRANSMIT

ReadProperty-Request,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type
8. RECEIVE

ReadProperty-ACK,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Type,
 'Property Value' = DEVICE

12.1.2.2.1.9 Eat An Error

Purpose: To verify that the Receive Frame State Machine correctly rejects an initial preamble octet for which a ReceiveError occurred. The following Receive Frame State Machine transition is verified by this test:

IDLE: EatAnError

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT

Test_Request
4. RECEIVE

Test_Response
5. TRANSMIT

ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,

- 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
6. WAIT T_{reply_delay}
 7. TRANSMIT
 - ReadProperty-Request,
 - 'Frame Type' = BACnet Data Expecting Reply,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type,
 8. RECEIVE
 - ReadProperty-ACK,
 - 'Frame Type' = BACnet Data Not Expecting Reply,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type,
 - 'Property Value' = DEVICE

Notes to Tester: In the transmission of step 5, the X'55' octet in the preamble shall be transmitted with a logical zero in the stop bit position.

12.1.2.2.1.10 Eat An Octet

Purpose: To verify that the Receive Frame State Machine correctly rejects an initial preamble octet that does not have the value X'55'. The following Receive Frame State Machine transition is verified by this test:

IDLE: EatAnOctet

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
 - Test_Request
4. RECEIVE
 - Test_Response
5. TRANSMIT
 - ReadProperty-Request,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type
6. RECEIVE
 - ReadProperty-ACK,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type,
 - 'Property Value' = DEVICE
7. TRANSMIT
 - (octet other than X'55')
8. TRANSMIT
 - ReadProperty-Request,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type
9. RECEIVE
 - ReadProperty-ACK,
 - 'Object Identifier' = (Device, X),
 - 'Property Identifier' = Object_Type,
 - 'Property Value' = DEVICE

12.1.2.2.1.11 Frame Too Long

Purpose: To verify the Receive Frame State Machine error check for a frame too large for the IUT. This tests the following Receive Frame State Machine Transition:

HEADER_CRC: FrameTooLong

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Test_Request,
'Length' = (x > 501),
'Data' = (x number of octets)
6. WAIT T_{reply_delay}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
8. RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Type,
'Property Value' = DEVICE

12.1.2.2.1.12 Illegally Broadcast Frame

Purpose: To verify the Slave Node Finite State Machine's rejection of an illegally broadcast frame. This tests the following Slave Node Finite State Machine transition:

IDLE: ReceivedUnwantedFrame (b)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
DA = LOCAL BROADCAST,
'Frame Type' = BACnet Data Expecting Reply
(Any Confirmed service supported by IUT),
6. BEFORE (T_{reply_delay}) {
IF (receive a frame) THEN
ERROR "Response to broadcast confirmed service request."
}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
8. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X)

12.1.2.2.1.13 Illegally Broadcast Test_Request Frame

Purpose: To verify the Slave Node Finite State Machine's handling of an illegally broadcast Test Request frame. This tests the following Slave Node Finite State Machine transition:

IDLE: ReceivedUnwantedFrame (b)

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
DA = LOCAL BROADCAST,
Test_Request
6. BEFORE (T_{reply_delay}) {
IF (receive a frame) THEN
ERROR "Response to broadcast Test_Request."
}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
8. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X),

12.1.2.2.1.14 Unwanted Token Frame

Purpose: To verify the Slave Node Finite State Machine's rejection of an unwanted Token frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Token
6. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
7. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X)

12.1.2.2.1.15 Unwanted Poll For Master Frame

Purpose: To verify the Slave Node Finite State Machine's rejection of an unwanted Poll For Master frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Poll For Master
6. BEFORE (T_{reply_delay}) {
IF (receive a frame) THEN
ERROR "Slave device responded to Poll For Master."
}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
8. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X)

12.1.2.2.1.16 Unwanted Reply to Poll For Master Frame

Purpose: To verify the Slave Node Finite State Machine's rejection of an unwanted Reply to Poll For Master frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request
4. RECEIVE
Test_Response
5. TRANSMIT
Reply To Poll For Master
6. BEFORE (T_{reply_delay}) {
IF (receive a frame) THEN
ERROR "Slave device responded to Reply To Poll For Master."
}
7. TRANSMIT
ReadProperty-Request,
'Frame Type' = BACnet Data Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier
8. BEFORE (T_{reply_delay})
RECEIVE
ReadProperty-ACK,
'Frame Type' = BACnet Data Not Expecting Reply,
'Object Identifier' = (Device, X),
'Property Identifier' = Object_Identifier,
'Property Value' = (Device, X)

12.1.2.2.1.17 Unwanted Reply Postponed Frame

Purpose: To verify the Slave Node Finite State Machine's rejection of an unwanted Reply Postponed frame.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
 Test_Request
4. RECEIVE
 Test_Response
5. TRANSMIT
 Reply Postponed
6. BEFORE (T_{reply_delay}) {
 IF (receive a frame) THEN
 ERROR "Slave device responded to Reply Postponed."
 }
7. TRANSMIT
 ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
8. BEFORE (T_{reply_delay})
 RECEIVE
 ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,
 'Property Value' = (Device, X)

12.1.2.2.2 Tests with Response

This clause defines the test cases where data link errors are corrected or which cause a response to be issued.

12.1.2.2.2.1 Repeated Preamble1

Purpose: To verify the Receive Frame State Machine check for a repeated first preamble octet. The following Receive Frame State Machine transition is verified by this test:

PREAMBLE: RepeatedPreamble1

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
 Test_Request
4. RECEIVE
 Test_Response
5. TRANSMIT
 X'55'
6. TRANSMIT
 ReadProperty-Request,
 'Frame Type' = BACnet Data Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier
7. BEFORE (T_{reply_delay})
 RECEIVE
 ReadProperty-ACK,
 'Frame Type' = BACnet Data Not Expecting Reply,
 'Object Identifier' = (Device, X),
 'Property Identifier' = Object_Identifier,

'Property Value' = (Device, X)

12.1.2.2.2.2 Test Request Empty Frame

Purpose: To verify acceptance of an empty Test_Request frame with reply via Test_Response. This verifies the Slave Node Finite State Machine transitions:

IDLE: ReceivedDataNeedingReply
ANSWER DATA REQUEST: Reply

and the Receive Frame State Machine transition:

HEADER CRC: NoData

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request,
'Length' = 0
4. BEFORE (T_{reply_delay})
RECEIVE
Test_Response,
'Length' = 0

Notes to Tester: In the response in step 4, there shall be no inter-frame gaps greater than T_{frame_gap}.

12.1.2.2.2.3 Test Request With Data

Purpose: To verify acceptance of a Test_Request frame with data and reply via Test_Response.

Configuration Requirements: The Test_Request frame transmitted in step 3 shall have a length less than or equal to (the value of the IUT Device object's Max_APDU_Length_Accepted property) + 21. In the response in step 4, there shall be no inter-frame gaps greater than T_{frame_gap}.

Test Steps:

1. MAKE (IUT turned off or otherwise reset)
2. MAKE (IUT turned on or otherwise started)
3. TRANSMIT
Test_Request,
'Length' = (x > 0),
'Data' = (x number of octets)
4. BEFORE (T_{reply_delay})
RECEIVE
Test_Response

12.1.3 MS/TP Data Link Layer Tests (Alternate)

This clause defines tests that can be applied without a specialized MS/TP testing tool.

12.1.3.1 Test Environment

12.1.3.1.1 Test Setup

In these tests the TD is installed on the non-MS/TP side of a router and therefore the tests do not cover strict conformance to the MS/TP data link layer.

These tests require the use of an MS/TP router and an MS/TP master device.

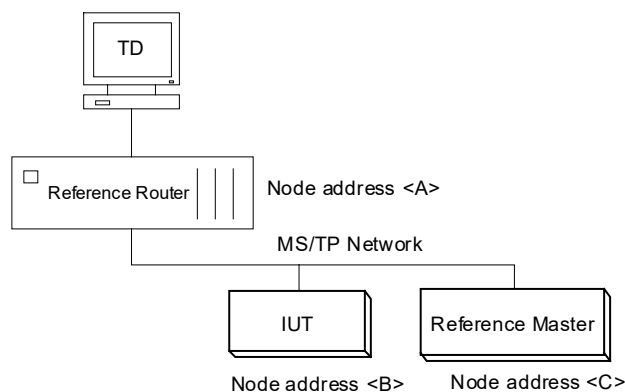


Figure 12-1. General MS/TP Test Network Setup.

The MS/TP node addresses are not critical, but must meet these requirements:

- $\langle A \rangle = 0$
- $\langle B \rangle = \langle A \rangle + 2$ or higher (this address changes during testing)
- $\langle C \rangle = \langle B \rangle + 2$ or higher

12.1.3.1.2 Serial Analyzer

Some of the tests specify the use of a serial analyzer for measuring characteristics of communication signals on the MS/TP cabling.

When measuring the silence time between MS/TP frames, the desired measurement is the elapsed time from the last bit transmitted of the first frame to the first bit transmitted in the following frame. When using a serial analyzer that applies a time stamp to each octet, taking the difference between the time stamps introduces an error equal to the transmission time of one octet because the difference between the time stamps is actually the elapsed time from the last bit of the last octet of the first frame to the last bit (not the first bit) of the first octet in the following frame.

The tests require that the serial analyzer used is accurate to the nearest millisecond and thus the tests assume that all measurements are ± 1 millisecond. As a result of these measurement inaccuracies, all measurements of silence time between frames where the silence time is required to be less than a specified amount are allowed to be as much as 2 milliseconds greater than the specified limit, and all measurements of silence time between frames that must be greater than a specified amount are allowed to be as much as 1 millisecond less than the specified limit. The following terms are defined to represent these measurement tolerances:

$$T_{\text{pos\_err}} = 2 \text{ milliseconds}$$

$$T_{\text{neg\_err}} = 1 \text{ millisecond}$$

12.1.3.1.3 Other Test Equipment

Some of the tests require the verification of inter-bit timings that are not normally available from serial analyzers. The tester is responsible for choosing appropriate testing equipment for these measurement tasks (logic analyzer, oscilloscope, etc.).

12.1.3.2 Verify $T_{\text{postdrive}}$

Purpose: Verify that the time between the transmission of the last bit in a frame and the time that the IUT stops driving its EIA-485 transmitter is 15 bit times or less.

Configuration Requirements: Connect an oscilloscope to the MS/TP network.

Test Steps:

1. Elicit the transmission of any frame type from the IUT. For IUTs that are master nodes, any Token frame or Poll For Master frame is satisfactory. For IUTs that are slave nodes, send any request to the slave node that elicits a response frame from the slave.

2. Measure the time interval from the trailing edge of the last stop bit transmitted by the IUT to the time that the EIA-485 voltage levels returns to idle. If the IUT employs a “padding” octet of X'FF' as the last octet of every frame, then the time shall be measured from the trailing edge of the stop bit of the octet that precedes the X'FF' “pad” octet.
3. Fail the IUT if the time interval measured in step 2 is greater than the time intervals shown below for each baud rate.

9600 baud:	fail if interval is greater than 1,562 microseconds
19200 baud:	fail if interval is greater than 781 microseconds
38400 baud:	fail if interval is greater than 391 microseconds
57600 baud:	fail if interval is greater than 260 microseconds
76800 baud:	fail if interval is greater than 195 microseconds
115200 baud:	fail if interval is greater than 130 microseconds
x baud:	fail if interval is greater than (15/x) seconds

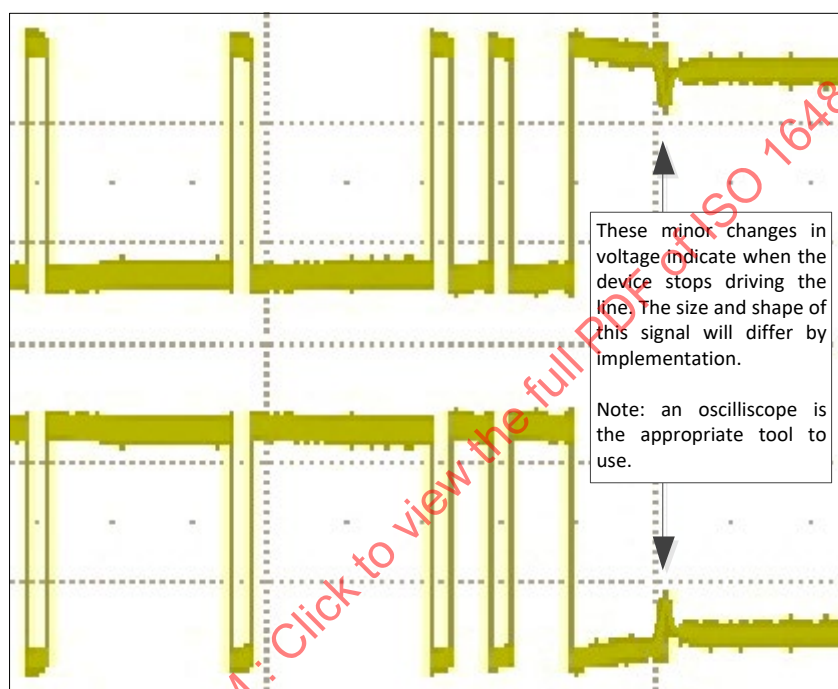


Figure 12-2: Example oscilloscope trace showing effect of $T_{\text{postdrive}}$

Notes to Tester: Note that temporarily using non-standard termination or bias can exaggerate the "minor changes in voltage" indicated in Figure 12-2.

12.1.3.3 Verify $T_{\text{frame\_gap}}$

Purpose: Verify that the maximum idle time between octets when transmitting a frame is 20 bit times or less.

Configuration Requirements: Run the IUT and a Reference Master (or Router) on the same MS/TP network.

Test Steps:

1. Elicit the transmission of any frame from the IUT.
2. Measure the longest EIA-485 idle time that appears between octets within the frame transmitted by the IUT. If there is no idle time between octets, pass the IUT.
3. Fail the IUT if the time measured in step 2 is greater than the time intervals shown below for each baud rate.

9600 baud:	fail if interval is greater than 2,083 microseconds
19200 baud:	fail if interval is greater than 1,042 microseconds
38400 baud:	fail if interval is greater than 521 microseconds
57600 baud:	fail if interval is greater than 347 microseconds
76800 baud:	fail if interval is greater than 261 microseconds

115200 baud:	fail if interval is greater than 173 microseconds
x baud:	fail if interval is greater than (20/x) seconds

12.1.3.4 Verify $T_{\text{turnaround}}$

Purpose: Verify that the IUT waits at least 40 bit times before enabling its EIA-485 transmitter after the reception of the last octet of a frame.

Test Steps:

1. Elicit the transmission of any frame type from the IUT. For IUTs that are master nodes, any frame sent after the IUT receives a Token frame or a Poll For Master frame shall be satisfactory. For IUTs that are slave nodes, send any request to the slave node that elicits a response frame from the slave.
2. Measure the time interval from the trailing edge of the last stop bit in the frame transmitted by the reference master, to the point where the EIA-485 voltage becomes driven by the IUT at the beginning of the frame transmitted by the IUT. If the reference master employs a “padding” octet of X'FF' as the last octet of every frame, then the time shall be measured starting from the trailing edge of the stop bit of the octet that precedes the X'FF' “pad” octet in the frame transmitted by the reference master.
3. Fail the IUT if the time measured in step 2 is less than the time intervals shown below for each baud rate.

9600 baud:	fail if interval is less than 4,167 microseconds
19200 baud:	fail if interval is less than 2,083 microseconds
38400 baud:	fail if interval is less than 1,042 microseconds
57600 baud:	fail if interval is less than 694 microseconds
76800 baud:	fail if interval is less than 521 microseconds
115200 baud:	fail if interval is less than 347 microseconds
x baud:	fail if interval is less than (40/x) seconds

12.1.3.5 Verify $T_{\text{reply\_delay}}$

Purpose: Verify that the time between a DER frame sent to the IUT and the first octet of a reply frame or Reply Postponed frame from the IUT is no longer than 250 milliseconds.

Note to Tester: The 250 millisecond time limit can be stressed by making a DER request which requires a large packet (a packet size approaching the maximum length supported by the device) to be sent in response. For example, a ReadPropertyMultiple request, if supported, for several properties or for “all” from the device object could require the IUT to construct and issue a large packet in response.

Test Steps:

1. MAKE (IUT turned on or otherwise started)
2. VERIFY (any supported property) = (any valid value)
3. CHECK (Did the IUT transmit a type 6 or type 7 frame immediately after the type 5 frame?)
4. CHECK (Is the time difference between the last octet of the type 5 frame and the first octet of the type 6 or 7 frame sent by the IUT less than 250 milliseconds?)

12.1.3.6 Verify $T_{\text{usage\_delay}}$ After a Token w/ Serial Analyzer

Purpose: This test verifies that the IUT begins using the Token within 15 milliseconds of receiving the last octet of a Token frame.

Test Steps:

1. MAKE (IUT turned on or otherwise started)
2. MAKE (MS/TP master device turned on or otherwise started)
3. CHECK (Is the time difference between the last octet of any type 0 frame (Token) sent by the master device and the first octet transmitted by the IUT less than 15 milliseconds?)

12.1.3.7 Verify $T_{\text{usage\_delay}}$ After a Poll For Master w/ Serial Analyzer

Purpose: This test verifies that the IUT begins using the Token within 15 milliseconds of receiving the last octet of a Poll For Master frame.

Test Steps:

1. MAKE (IUT turned on or otherwise started)
2. CHECK (Is the IUT transmitting type 1 frames (Poll For Master)?)
3. MAKE (second MS/TP master device turned on or otherwise started)
4. WAIT (until the second MS/TP master device sends a type 1 frame (Poll For Master) to the IUT)
5. CHECK (Did the IUT respond with a type 2 frame (Reply To Poll For Master)?)
6. CHECK (Is the time difference between the last octet of any type 1 frame sent to the IUT and the first octet transmitted by the IUT less than 15 milliseconds?)

12.1.3.8 Verify N_{poll} w/ Serial Analyzer

Purpose: Verify that the number of Token frames that the IUT sends between Poll For Master cycles is between 50 and 52.

Configuration Requirements: Separate the addresses of the master devices enough so that both devices shall transmit Poll For Master frames. The reason for this is that the MS/TP master nodes shall not begin a Poll For Master cycle if the next device is the device + 1.

Test Steps:

1. MAKE (IUT turned on)
2. WAIT (several seconds)
3. MAKE (serial analyzer turned on to capture traffic)
4. WAIT (10 seconds)
5. MAKE (Pause the serial analyzer.)
6. CHECK (The number of Token frames that the IUT sends between Poll For Master cycles is between 50 and 52.)

12.1.3.9 Verify $T_{usage\_timeout}$ w/ Serial Analyzer

Purpose: Verify that the IUT waits at least 20 milliseconds but no longer than 100 milliseconds for another master node to begin using the Token or reply to a Poll For Master frame.

Configuration Requirements: Set up the devices such that the IUT is addressed at 1 greater than the address of the other master device.

Test Steps:

1. MAKE (Power on both devices.)
2. WAIT (several seconds)
3. VERIFY (Has Token passing been established between the devices?)
4. MAKE (Power off the other master device, but not the IUT.)
5. WAIT (10 seconds)
6. MAKE (Stop the data capture.)
7. CHECK (Did the IUT send a Token (type 0) frame to the other master, and, when the other master did not use the Token (because it was powered off), did the IUT follow the Token (type 0) frame with one Token (type 0) frame (Token retry) followed by a series of Poll For Master (type 1) frames?)
8. CHECK (Is the time difference between the last octet of the Token (type 0) frame sent by the IUT and the first octet of the immediately following the Poll For Master (type 1) frame transmitted by the IUT greater than 20 milliseconds - T_{neg_err} and less than 100 milliseconds + T_{pos_err} ?)
9. CHECK (Is the time gap (last character to first character) between any two Poll For Master (type 1) frames sent by the IUT greater than 20 milliseconds - T_{neg_err} , but less than 100 milliseconds + T_{pos_err} ?)

12.1.3.10 Max_Master Test

Purpose: Verify that Max_Master is writable, or that it is 127.

Configuration Requirements: The IUT shall be configured with an MS/TP MAC address less than 127.

Test Steps:

1. VERIFY Max_Master = (any valid value, V1)

2. IF Max_Master is writable THEN
WRITE Max_Master = (any other valid value, with V2 greater than or equal to the IUT's address)
3. ELSE
VERIFY Max_Master = 127

12.1.3.11 Max_Info_Frames Test

Purpose: Verify that Max_Info_Frames is configurable, or that it is 1.

Test Steps:

1. VERIFY Max_Info_Frames = (any valid value)
2. IF Max_Info_Frames is writable THEN
WRITE Max_Info_Frames = (any valid value, V)
VERIFY Max_Info_Frames = V
ELSE IF Max_Info_Frames is configurable THEN
MAKE (Max_Info_Frames equal any valid value, V)
VERIFY Max_Info_Frames = V
ELSE
VERIFY Max_Info_Frames = 1

12.1.3.12 Master Node Data Frame Test

Purpose: Verify that the IUT can properly receive and transmit simple MS/TP data frames.

Test Steps:

1. If the IUT supports DM-DDB-B, perform test 135.1, Clause 9.33.2.2 (Who-Is, General Inquiry, Remote Broadcast) on the IUT and verify the I-Am response. (This tests for proper reception of MS/TP broadcasts and transmission of MS/TP broadcasts.)
2. Perform a ReadProperty of any property (135.1, Clause 9.18.1) to verify correct reception and transmission of unicast messages.

12.1.3.13 Poll For Master w/ Serial Analyzer

Purpose: This tests that a master node performs the Poll For Master sequence properly.

Configuration Requirements: Configure the reference router at address <A>, the reference master at address <C>, and the IUT at address where <A> is not zero, is equal to <A>+2, and is less than <C>-1. Start with the IUT powered off.

Test Steps:

1. MAKE (Power on the IUT.)
2. CHECK (Verify that the IUT responds to Poll For Master requests from <A>.)
3. CHECK (Verify that the IUT periodically transmits Poll For Master frames to nodes + 1 through <C> - 1 and that it only transmits one Poll For Master request to node <C>.).
4. MAKE (Change the IUT's address to <C> - 1 and power cycle the device.)
5. CHECK (Verify that the IUT sends one Poll For Master frame to node <C> and then ceases to transmit any Poll For Master frames.)
6. MAKE (Power off the reference master at address <C>.)
7. MAKE (Power off the IUT.)
8. MAKE (Power on the IUT so that the IUT and reference router are the only nodes on the network.)
9. CHECK (Verify that the IUT periodically transmits Poll For Master frames to nodes + 1 through its own Max_Master setting and nodes zero through <A> - 1 and that it only sends one Poll For Master to node <A>.)
10. MAKE (Power off the IUT. Set its address equal to its own Max_Master setting. Power on the IUT.)
11. CHECK (Verify that the IUT periodically transmits Poll For Master frames to nodes zero through node <A> - 1 and that it only sends one Poll For Master to node <A>.)

12.1.3.14 Slave Node Data Frame Test

Purpose: This test verifies that the IUT can properly receive and transmit simple MS/TP data frames.

Test Steps:

1. If the IUT supports DM-TS-B, send a Remote Broadcast TimeSynchronization service and verify the time change in the IUT Device Object. (This tests for proper reception of MS/TP broadcasts in a slave device.)
2. Perform a ReadProperty of any property (135.1, Clause 9.15.1) to verify correct reception and transmission of unicast messages.

12.1.3.15 Sole Master Test

Purpose: This test verifies that the IUT properly initiates Poll For Master frames when it is the only node installed on the network.

Configuration Requirements: Power off all of the nodes on the MS/TP network, including the IUT. The node address of the IUT can be set to any address 0 through 127, but it shall be less than or equal to its own Max_Master property. The IUT node address shall be referred to as address .

Test Steps:

1. MAKE (Power on the IUT.)
2. WAIT (until the IUT finishes initializing)
3. CHECK (that the IUT starts transmitting Poll For Master frames, starting with address +1, continues through address Max_Master, and then addresses 0 through -1)
4. CHECK (that the Poll For Master cycle repeats)
5. CHECK (that the IUT sends no more than 50 * Max_Info_Frames between successive Poll For Master cycles)

Notes to Tester: This test shall be skipped if the device requires that there be MS/TP activity from which it will determine the baud rate to use.

In step 5, it is acceptable for devices claiming a Protocol_Revision less than 5 to send 52 sets of Max_Info_Frames between Poll For Master cycles.

12.1.3.16 MS/TP Network Startup Tests (IUT power on Variation)

Purpose: Verify that the IUT can join a preexisting MS/TP network when the IUT is introduced into the MS/TP network from a power-on scenario.

Test Concept: A network of reference masters is constructed and is turned on with the IUT remaining off. Once the network achieves normal network operation, the IUT is connected to the network and powered on. The network is monitored to verify that the IUT successfully joins the network within a reasonable time period.

This test shall be performed both with a single reference master and with multiple reference masters.

Configuration Requirements: The test starts with an MS/TP network comprised of one or more reference master devices that has achieved normal network operation. If the IUT does not autobaud, then it shall be configured with the same baud rate of the operating network. The IUT shall be configured with a valid MAC address (0-127) which is not in use by any of the other devices on the network and is less than the Max_Master value in use by the reference masters. The IUT shall be configured with the same Max_Master in use by the other master devices.

Test Steps:

1. MAKE (Power on or otherwise start the IUT.)
2. CHECK (the IUT does not generate any packets until it receives a Poll For Master frame)
3. CHECK (Verify that the IUT correctly joins the MS/TP network by answering a Poll For Master destined for its MAC address, accepting a Token, and generating Poll For Master frames and subsequently passing the Token.)

Passing Result: Note that the IUT may take a considerable amount of time before accepting Poll For Master frames. The duration of the CHECK in step 3 shall be a sufficient duration for the IUT as defined by the vendor.

12.1.3.17 MS/TP Network Startup Tests (IUT's wire connected)

Purpose: To verify that the IUT can, after powering on and declaring Sole Master, successfully join an established MS/TP network.

Test Concept: A network of reference masters is constructed and is turned on with the IUT disconnected. Once the network achieves normal network operation, the IUT is powered on. Once the IUT reaches the Sole Master state, the IUT is connected to the network. The network is monitored to verify that the IUT successfully joins the network within a reasonable time period.

This test shall be performed both with a single other master device and with multiple master devices.

Configuration Requirements: The test starts with an MS/TP network comprised of one or more reference master devices that has achieved normal network operation. If the IUT does not autobaud, then it shall be configured with the baud rate of the operating network. The IUT shall be configured with a valid MAC address (0-127) which is not in use by any of the other devices on the network and is less than the Max_Master value in use by the other masters. The IUT shall be configured with the same Max_Master as is used by the other master devices.

If the IUT lurks until it detects traffic after power on, then this test shall be skipped. For this test, Time_to_join in step 5 is 60 Seconds. The exact time for a device to join is dependent on implementation and the sequence of events that follow the joining of two live networks. The time shall be the sum of the time duration of collisions, the time to start a Poll For Master cycle, and the time to poll and receive a response from the other master. (This total time cannot be calculated because the standard does not specify how often the master node state machine must be run—only that timers must have a 5ms resolution. Moreover, it is possible that the master node could be sending out data frames in between each Poll For Master.)

Test Steps:

1. MAKE (Power on or otherwise start the IUT without the MS/TP wire connected to the IUT.)
2. WAIT (a vendor specified time (for device startup and/or auto-baud completion))
3. CHECK (Verify that the IUT generates Poll For Master frames.)
4. MAKE (Connect the IUT to the MS/TP network.)
5. CHECK (Did the IUT join the MS/TP network within Time_to_join seconds?)

12.1.3.18 MS/TP Network Startup Tests (IUT's wire disconnected)

Purpose: To verify that the IUT can, after disconnecting from a network and declaring Sole Master, successfully rejoin an established MS/TP network.

Test Concept: A network of master devices is constructed and is turned on with the IUT connected. Once the network achieves normal network operation, the IUT is disconnected from the network. Once the IUT reaches the Sole Master state, the IUT is re-connected to the network. The network is monitored to verify that the IUT successfully rejoins the network within a reasonable time period.

This test shall be performed both with a single other master device and with multiple other master devices.

Configuration Requirements: The test starts with an MS/TP network comprised of one or more other master devices and the IUT that has achieved normal network operation. If the IUT does not autobaud, then it shall be configured with the baud rate of the operating network. The IUT shall be configured with a valid MAC address (0-127) which is not in use by any of the other devices on the network and is less than the Max_Master value in use by the other master devices on the network.

The time, Time_to_join, in step 6 is 60 Seconds. The exact time is dependant on implementation and the sequence of events that follow the joining of two live networks. The time shall be the sum of the time duration of collisions, the time to start a Poll For Master cycle, and the time to poll and receive a response from the other master. (This total time cannot be calculated as the standard does not specify how often the master node state machine must be run—only that timers must have a 5ms resolution. Moreover, it is possible that the master node could be sending out data frames in between each Poll For Master.)

Note that if the IUT possesses the Token before step 2, the wait time in step 3 shall be significantly less than the time indicated. The time between step 2 and step 5 shall only be long enough to perform step 4 because some devices may revert back to lurking if separated from the network for too long.