
**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*



STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020



COPYRIGHT PROTECTED DOCUMENT

© ISO 2020

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
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

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 documents should be noted. International Standards are drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

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).

This fourth edition cancels and replaces the third edition (ISO 16484-6:2014), which has been technically revised. See the detailed list of changes in pages 724 to 728.

A list of all parts in the ISO 16484 series can be found on the ISO website.

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.

CONTENTS

CLAUSE	PAGE
Foreword.....	iii
1. PURPOSE.....	1
2. SCOPE.....	1
3. DEFINITIONS	1
4. ELECTRONIC PICS FILE FORMAT	1
4.1 Character Encoding.....	1
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.....	4
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
7. OBJECT SUPPORT TESTS	20
7.1 Read Support for Properties in the Test Database.....	20
7.2 Write Support for Properties in the Test Database.....	22
7.3 Object Functionality Tests.....	24
8. APPLICATION SERVICE INITIATION TESTS.....	186
8.1 AcknowledgeAlarm Service Initiation Tests	186
8.2 ConfirmedCOVNotification Service Initiation Tests.....	187
8.3 UnconfirmedCOVNotification Service Initiation Tests	196
8.4 ConfirmedEventNotification Service Initiation Tests	199
8.5 UnconfirmedEventNotification Service Initiation Tests	241
8.6 GetAlarmSummary Service Initiation Tests.....	261
8.7 GetEnrollmentSummary Service Initiation Tests	262
8.8 GetEventInformation Service Initiation Tests.....	263
8.9 LifeSafetyOperation Service Initiation Tests	265
8.10 SubscribeCOV Service Initiation Tests	266
8.11 SubscribeCOVProperty Service Initiation Tests.....	267
8.12 AtomicReadFile Service Initiation Tests.....	268
8.13 AtomicWriteFile Service Initiation Tests	268
8.14 AddListElement Service Initiation Tests	269
8.15 RemoveListElement Service Initiation Tests.....	270
8.16 CreateObject Service Initiation Tests.....	270
8.17 DeleteObject Service Initiation Tests.....	271
8.18 ReadProperty Service Initiation Tests.....	271
8.19 ReadPropertyConditional Service Initiation Tests.....	273
8.20 ReadPropertyMultiple Service Initiation Tests.....	274
8.21 ReadRange Service Initiation Tests	276
8.22 WriteProperty Service Initiation Tests	280
8.23 WritePropertyMultiple Service Initiation Tests.....	282
8.24 DeviceCommunicationControl Service Initiation Tests.....	284
8.25 ConfirmedPrivateTransfer Service Initiation Test.....	286
8.26 UnconfirmedPrivateTransfer Service Initiation Test.....	286

8.27	ReinitializeDevice Service Initiation Tests.....	286
8.28	ConfirmedTextMessage Service Initiation Tests.....	287
8.29	UnconfirmedTextMessage Service Initiation Tests.....	288
8.30	TimeSynchronization Service Initiation Tests.....	289
8.31	UTCTimeSynchronization Service Initiation Tests.....	290
8.32	Who-Has Service Initiation Tests.....	290
8.33	I-Have Service Initiation Tests.....	291
8.34	Who-Is Service Initiation Tests.....	291
8.35	I-Am Service Initiation Tests.....	292
8.36	VT-Open Service Initiation Tests.....	292
8.37	VT-Close Service Initiation Tests.....	293
8.38	VT-Data Service Initiation Tests.....	294
8.39	RequestKey Service Initiation Tests.....	296
8.40	Authenticate Service Initiation Tests.....	297
9.	APPLICATION SERVICE EXECUTION TESTS.....	301
9.1	AcknowledgeAlarm Service Execution Tests.....	301
9.2	ConfirmedCOVNotification Service Execution Tests.....	327
9.3	UnconfirmedCOVNotification Service Execution Tests.....	332
9.4	ConfirmedEventNotification Service Execution Tests.....	334
9.5	UnconfirmedEventNotification Service Execution Tests.....	337
9.6	GetAlarmSummary Service Execution Tests.....	337
9.7	GetEnrollmentSummary Service Execution Tests.....	338
9.8	GetEventInformation Service Execution Tests.....	342
9.9	LifeSafetyOperation Service Execution Test.....	345
9.10	SubscribeCOV Service Execution Tests.....	346
9.11	SubscribeCOVProperty Service Execution Tests.....	354
9.12	AtomicReadFile Service Execution Tests.....	361
9.13	AtomicWriteFile Service Execution Tests.....	368
9.14	AddListElement Service Execution Tests.....	379
9.15	RemoveListElement Service Execution Tests.....	381
9.16	CreateObject Service Execution Tests.....	383
9.17	DeleteObject Service Execution Tests.....	388
9.18	ReadProperty Service Execution Tests.....	389
9.19	ReadPropertyConditional Service Execution Tests.....	391
9.20	ReadPropertyMultiple Service Execution Tests.....	392
9.21	ReadRange Service Execution Tests.....	400
9.22	WriteProperty Service Execution Tests.....	410
9.23	WritePropertyMultiple Service Execution Tests.....	415
9.24	DeviceCommunicationControl Service Execution Test.....	424
9.25	ConfirmedPrivateTransfer Service Execution Tests.....	430
9.26	UnconfirmedPrivateTransfer Service Execution Tests.....	431
9.27	ReinitializeDevice Service Execution Tests.....	431
9.28	ConfirmedTextMessage Service Execution Tests.....	434
9.29	UnconfirmedTextMessage Service Execution Tests.....	435
9.30	TimeSynchronization Service Execution Tests.....	435
9.31	UTCTimeSynchronization Service Execution Tests.....	437
9.32	Who-Has Service Execution Tests.....	437
9.33	Who-Is Service Execution Tests.....	444
9.34	VT-Open Service Execution Tests.....	447
9.35	VT-Close Service Execution Tests.....	449

9.36	VT-Data Service Execution Tests	450
9.37	RequestKey Service Execution Test.....	450
9.38	Authenticate Service Execution Tests.....	452
9.39	General Testing of Service Execution.....	456
10.	NETWORK LAYER PROTOCOL TESTS.....	458
10.1	General Network Layer Tests.....	458
10.2	Router Functionality Tests	459
10.3	Half-Router Functionality Tests	483
10.4	B/IP PAD Tests	490
10.5	Initiating Network Layer Messages	492
10.6	Non-Router Functionality Tests	494
10.7	Route Binding Tests	496
10.8	Virtual Routing Functionality Tests.....	501
11.	LOGICAL LINK LAYER PROTOCOL TESTS.....	520
11.1	UI Command and Response	520
11.2	XID Command and Response.....	520
11.3	TEST Command and Response	521
12.	DATA LINK LAYER PROTOCOLS TESTS	523
12.1	MS/TP State Machine Tests.....	523
12.2	PTP State Machine Tests	587
13.	SPECIAL FUNCTIONALITY TESTS.....	626
13.1	Segmentation	626
13.2	Time Master	635
13.3	Character Sets	640
13.4	Malformed PDUs	640
13.5	Slave Proxy Tests.....	642
13.6	Automatic Network Mapping	644
13.7	Automatic Device Mapping.....	645
13.8	Backup and Restore Procedure Tests	645
13.9	Application State Machine Tests	657
13.10	Workstation Scheduling Tests.....	658
14.	BACnet/IP Functionality Tests.....	676
14.1	Non-BBMD B/IP Device	676
14.2	BBMD B/IP Device with a Server Application	678
14.3	Broadcast Distribution Table Operations	682
14.4	Foreign Device Table Operations (Negative Tests).....	686
14.5	BACnet Broadcast Management (No Foreign Device Table, No Applications)	687
14.6	Foreign Device Management	689
14.7	Broadcast Management (BBMD, Foreign Devices, Local Application)	693
14.8	Registering as a Foreign Device.....	701
14.9	Initiating BVLL Service Requests Conveying an NPDU.....	702
15.	Reporting Test Results	704
	ANNEX A – EXAMPLE EPICS (INFORMATIVE).....	705
	HISTORY OF REVISIONS.....	722

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 135-2016 also apply to this addendum.

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

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

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

3.4 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

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: □↓
}↓
```

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  
  other properties...  
}
```

-- Writable when Out_Of_Service is TRUE

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

5. EPICS CONSISTENCY TESTS

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

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

- (a) All object types required by the specified BIBBs shall be indicated as supported in the Standard Object Types Supported section of the EPICS.
- (b) A minimum of one instance of each object type required by the specified BIBBs shall be included in the test database.
- (c) The Protocol_Object_Types_Supported property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.
- (d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.
- (e) The 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.
- (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, for each object included in the test database, the Property_List property shall have one entry for each property present, including non-standard properties, but excluding 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

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:

<object identifier> ::= '(' <object type> ',' <instance number> ')'

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.

<statement> ::= <simple statement> | <compound statement>

<compound statement> ::= '{' <statement>... '}'

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 135-2008, 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 135-2008, 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

6.3 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.

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.

6.4 BACnet References

All references to BACnet clauses in this standard refer to ANSI/ASHRAE 135-2008, 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.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

Purpose: Verifies that a correct response is returned when each property of each object is read using BACnet ReadProperty and ReadPropertyMultiple services. The test is performed once using ReadProperty and once using ReadPropertyMultiple, if supported. When verifying array properties, whole arrays shall be read without using an ARRAY INDEX, where possible.

Test Steps:

```

1. REPEAT X = (all objects in the IUT's database) DO {
    REPEAT Y = (all properties in object X) DO {
        IF (Y is defined by the standard as not accessible via the ReadProperty service) THEN
            TRANSMIT ReadProperty-Request,
                'Object Identifier' = X,
                'Property Identifier' = Y
            IF (Protocol_Revision >= 7) THEN
                RECEIVE BACnet-Error-PDU,
                    Error Class = PROPERTY,
                    Error Code = READ_ACCESS_DENIED
            ELSE
                RECEIVE BACnet-Error-PDU,
                    Error Class = PROPERTY,
                    Error Code = READ_ACCESS_DENIED | OTHER
                | (BACnet-Error-PDU,
                    Error Class = OBJECT,
                    Error Code = OTHER)
                | (BACnet-Error-PDU,
                    Error Class = SERVICES,
                    Error Code = SERVICE_REQUEST_DENIED | OTHER)
            ELSE IF (Y is an array and its value is too large to return given the IUT's APDU and segmentation
                limitations) THEN
                TRANSMIT ReadProperty-Request,
                    'Object Identifier' = X,
                    'Property Identifier' = Y
                RECEIVE BACnet-Abort-PDU,
                    '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,
                    'Property Identifier' = Y,
                    'Property Array Index' = 0,
                    'Property Value' = (N: the number of array elements in Y as indicated in the 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)
    }
}

```

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.

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.

Test Steps:

```

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

```

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

7.1.3 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.

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)
}

```

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.

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.

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.

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

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {
 REPEAT Y = (all writable properties in object X) DO {
 REPEAT Z = (a set of values meeting the range requirements of 7.2.1 and

```

        any additional restrictions placed on the allowable property values as specified 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. If the IUT does not support the WriteProperty service, then this test shall be skipped.

Test Steps:

```

1. REPEAT X = (a tester selected set of objects) DO {
    REPEAT Y = (all read-only properties in object X) DO {
        IF (the property is not an array) THEN
            READ Z = X
            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 element value meeting the range requirements of 7.2.1
excluding
                                those in Z)
                RECEIVE BACnet-Error-PDU,
                    Error Class = PROPERTY,
                    Error Code = WRITE_ACCESS_DENIED
            ELSE
                READ LEN = X, Array_Index = 0
                IF (LEN > 0) THEN
                    READ Z = X, Array Index = 1
                    TRANSMIT WriteProperty-Request,
                        'Object Identifier' = X,
                        'Property Identifier' = Y,
                        'Property Value' = Z

```

```

        'Priority 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)
        'Priority Array Index' =      1
    RECEIVE BACnet-Error-PDU,
        Error Class =                PROPERTY,
        Error Code =                 WRITE_ACCESS_DENIED

    IF (the IUT supports AddListElement and the property is an array of lists) THEN
        TRANSMIT AddListElement-Request,
            'Object Identifier' =      X,
            'Property Identifier' =     Y,
            'Priority Array Index' =    1
            List of Elements' =        (any elements value meeting the range requirements of
                                      7.2.1 excluding those in Z)

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

```

Notes to Tester: 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.

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

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

BACnet Reference Clauses: 12.1.7, 12.1.9, 12.1.10, 12.2.7, 12.2.9, 12.2.10, 12.3.7, 12.3.9, 12.3.10, 12.4.6, 12.4.8, 12.4.9, 12.6.7, 12.6.9, 12.6.10, 12.7.7, 12.7.9, 12.7.10, 12.8.6, 12.8.8, 12.8.9, 12.15.8, 12.15.10, 12.15.11, 12.16.8, 12.16.10, 12.16.11, 12.17.6, 12.17.8, 12.17.9, 12.18.7, 12.18.9, 12.18.10, 12.19.7, 12.19.9, 12.19.10, 12.20.6, 12.20.8, 12.20.9, 12.23.7, 12.23.9, and 12.23.10.

Purpose: This test case verifies that `Present_Value` is 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. This test applies to Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Multi-state Input, Multi-state Output, Multi-state Value, Loop and Pulse Converter objects.

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 4 shall be omitted.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

Test Steps:

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

Notes to Tester: If the object being tested is commandable and there is an internal process writing to the Present_Value property, then each WriteProperty request shall contain a priority sufficient to override the internal process. After step 4 the priority array slot shall be relinquished.

7.3.1.2 Relinquish Default Test

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

BACnet Reference Clauses: 12.3.16, 12.4.12, 12.7.22, 12.8.20, 12.19.14, and 12.20.13.

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

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

BACnet Reference Clause: 19.2.

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 = Plow
20. REPEAT Z = (each index 1 through 5) DO {
    VERIFY Priority_Array = NULL, ARRAY INDEX = Z
  }

```

7.3.1.4 Minimum_Off_Time

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

BACnet Reference Clauses: 12.7.19, 12.8.17, 19.2.3, and Annex I.

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

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

BACnet Reference Clauses: 12.7.20, 12.8.18, 19.2.3, and Annex I.

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 Override of Minimum Time

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

BACnet Reference Clause: 12.7.19, 12.7.20, 12.8.17, 12.8.18, 19.2.

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 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. 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 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. BEFORE Minimum_On_Time
WRITE Present_Value = INACTIVE, PRIORITY = (any value numerically lower than 6 (higher priority))
5. VERIFY Present_Value = INACTIVE
6. VERIFY Priority_Array = INACTIVE, PRIORITY = 6

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 through 3 and ACTIVE in steps 4 through 6, and by using Minimum_Off_Time in step 4.

7.3.1.7 COV Tests

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

7.3.1.8 Binary Object Change of State Tests

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

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

Purpose: To verify that the properties of binary objects that collectively track state changes (changes in Present_Value) function as required. If the Change_Of_State_Count, Change_Of_State_Time, and Time_Of_State_Count_Reset properties are not supported this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.

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

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

Test Steps:

1. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value
2. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value,
 'Property Value' = ACTIVE | INACTIVE
3. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count
4. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count,
 'Property Value' = (any valid value, N)
5. IF (Present_Value is writable) THEN
 IF (the value returned in step 2 was ACTIVE) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
 ELSE
 WRITE Present_Value = ACTIVE
 VERIFY Present_Value = ACTIVE
 ELSE
 MAKE (Present_Value change to the opposite state)
6. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Date
7. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local Date,
 'Property Value' = (the current local date, D)
8. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time
9. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time,
 'Property Value' = (any valid time stamp, T_{Loc})
10. WAIT **Internal Processing Fail Time**
11. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Time
12. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Time,
 'Property Value' = (a date and time such that the date = D and the time is approximately T_{Loc})
13. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count
14. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Change_Of_State_Count,

- 'Property Value' = N + 1
15. IF (Change_Of_State_Count is writable) THEN
 WRITE Change_Of_State_Count = 0
 VERIFY Change_Of_State_Count = 0
 ELSE
 MAKE (Change_Of_State_Count = 0)
 16. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time
 17. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time,
 'Property Value' = (any valid time stamp, T_{LOC})
 18. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Time_Of_State_Count_Reset
 19. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Time_Of_State_Count_Reset,
 'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})

7.3.1.9 Binary Object Elapsed Active Time Tests

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

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

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

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

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

Test Steps:

1. IF (Present_Value is writable) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
 ELSE
 MAKE (Present_Value = INACTIVE)
2. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Elapsed_Active_Time
3. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Elapsed_Active_Time,
 'Property Value' = (the elapsed active time, T_{ELAPSED} in seconds)
4. WAIT (1 minute)
5. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),

- 'Property Identifier' = Elapsed_Active_Time
6. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Elapsed_Active_Time,
 'Property Value' = (the same T_{ELAPSED} as step 3)
 7. IF (Present_Value is writable) THEN
 WRITE Present_Value = ACTIVE
 VERIFY Present_Value = ACTIVE
 ELSE
 MAKE (Present_Value = ACTIVE)
 8. WAIT (**Internal Processing Fail Time** + 30 seconds)
 9. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Elapsed_Active_Time
 10. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Elapsed_Active_Time,
 'Property Value' = (T: (T_{ELAPSED} + 30) ≤ T ≤ (T_{ELAPSED} + 30 + **Internal Processing Fail Time**))
 11. IF (Present_Value is writable) THEN
 WRITE Present_Value = INACTIVE
 VERIFY Present_Value = INACTIVE
 ELSE
 MAKE (Present_Value = INACTIVE)
 12. IF (Elapsed_Active_Time is writable) THEN
 WRITE Elapsed_Active_Time = 0
 VERIFY Elapsed_Active_Time = 0
 ELSE
 MAKE (Elapsed_Active_Time = 0)
 13. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Date
 14. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local Date,
 'Property Value' = (the current local date, D)
 15. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time
 16. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the IUT's Device object),
 'Property Identifier' = Local_Time,
 'Property Value' = (any valid time stamp, T_{LOC})
 17. TRANSMIT ReadProperty-Request,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Time_Of_Active_Time_Reset
 18. RECEIVE ReadProperty-ACK,
 'Object Identifier' = (the object being tested),
 'Property Identifier' = Present_Value,
 'Property Value' = (a date and time such that the date = D and the time is approximately T_{LOC})

7.3.1.10 Event_Enable Tests

7.3.1.10.1 Event_Enable Test for TO_OFFNORMAL, 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.

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).

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 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' =            (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 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.

7.3.1.10.2 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. TRANSMIT 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

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).

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)

3. IF (Protocol_Revision is present AND Protocol_Revision $\geq$ 13) THEN
 VERIFY pStatusFlags = (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 $\geq$ 13) THEN
 VERIFY pStatusFlags = (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' = OFFNORMAL,
 '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 $\geq$ 13) THEN
 VERIFY pStatusFlags = (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 TO_FAULT transition)

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),
  '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),
  'Notification Class' = (the class corresponding to the object being tested),
  'Priority' = (Pfault),
  'Event Type' = (any valid event type),
  'Message Text' = (optional, any valid message text),

```

```

        '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),
        'Notification Class' =            (the class corresponding to the object being tested),
        'Priority' =                      (Pnormal),
        'Event Type' =                    (any valid event type),
        'Message Text' =                  (optional, any valid message text),
        '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),
            'Notification Class' =        (the class corresponding to the object being 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),
        '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),
 '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)

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.

7.3.1.12 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.

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

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.

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.

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' = (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' = 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' = (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' = 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

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

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.

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' = (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' = 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' = (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' = 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 = (?, FALSE)
13. VERIFY pCurrentState = NORMAL
14. MAKE (pMonitoredValue a value that exceeds pHighLimit)

15. WAIT ($pTimeDelay$ + **Notification Fail Time**)
16. CHECK (verify that no notification message was transmitted)
17. VERIFY $pCurrentState$ = NORMAL

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

7.3.1.14 Process_Identifier Tests

7.3.1.14.1 Process_Identifier Property Test

Dependencies: None.

BACnet-2001 Reference Clause: 12.11.4.

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 $0 \dots 2^{32}-1$))
ELSE
WRITE Process_Identifier = (any value in the range $0 \dots 2^{32}-1$)
6. VERIFY Process_Identifier = (the value used in step 5)

7.3.1.14.2 Recipient_List Test

Dependencies: None

BACnet Reference Clause: 12.21.8.

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 $0 \dots 2^{32}-1$ 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

Dependencies: ReadProperty Service Execution Tests, 7.1; WriteProperty Service Execution Tests 7.2.2

BACnet Reference Clauses: 12.18.11, 12.19.4, 12.19.11, 12.20.4, 12.20.10

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.

Dependencies: None

BACnet Reference Clause: 12.

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. If the algorithm of the object does not support NORMAL to NORMAL transitions, then the TO-OFFNORMAL bit of the Event_Enable shall be TRUE. If the IUT does not contain any objects which can transition to any offnormal state, then this test shall be skipped.

Test Steps:

1. READ EMT = Event_Message_Texts
2. IF (Event_Enable is (TRUE, ?, ?)) THEN {
3. IF (pMonitoredValue is writable) THEN

```

        WRITE pMonitoredValue = (a value that is offnormal)
    ELSE
        MAKE (pMonitoredValue a value that is offnormal)
4.  WAIT (pTimeDelay)
5.  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_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)
6.  VERIFY Event_Message_Texts = EMT
    }
7.  IF (Event_Enable is (?, ?, TRUE)) THEN {
8.  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)
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 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)
11. VERIFY Event_Message_Texts = EMT
    }
12. IF (Event_Enable is (?, TRUE, ?)) THEN {
13. MAKE (a condition exist that will cause O1 to generate a TO-FAULT transition)
14. 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)
15. 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. 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 <> (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 <> (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.X2.1 against an object, O2, which supports both Event_Algorithm_Inhibit_Ref and Event_Algorithm_Inhibit, and instead of writing Event_Algorithm_Inhibit, write the property referenced by Event_Algorithm_Inhibit_Ref to change the value in the Event_Algorithm_Inhibit property.

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

7.3.1.20.2 Event_Algorithm_Inhibit Writable Test

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

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

Test Steps:

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

7.3.1.21 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 that a notification is generated. Set Reliability_Evaluation_Inhibit to TRUE. Verify that the Reliability changes to NO_FAULT_DETECTED and that a TO_NORMAL notification is generated. Remove the fault condition and ensure that no notification is generated. Make a fault condition exist and verify that Reliability remains NO_FAULT_DETECTED and that no notification is generated.

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

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. VERIFY Reliability = NO_FAULT_DETECTED
3. MAKE (a condition exist that would cause O1 to generate a TO_FAULT transition)
4. BEFORE **Notification Fail Time**
 - RECEIVE UnconfirmedEventNotification-Request
 - 'Process Identifier' = (the value configured for the transition),
 - 'Initiating Device Identifier' = IUT,
 - 'Event Object Identifier' = O1,
 - 'Time Stamp' = (any valid timestamp),
 - 'Priority' = (any valid priority),
 - 'Event Type' = CHANGE_OF_RELIABILITY,
 - '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)
5. WRITE Reliability_Evaluation_Inhibit = TRUE
 6. BEFORE **Internal Processing Fail Time + Notification Fail Time**
 RECEIVE UnconfirmedEventNotification-Request
 'Process Identifier' = (the value configured for the transition),
 'Initiating Device Identifier' = IUT,
 'Event Object Identifier' = O1,
 'Time Stamp' = (any valid timestamp),
 'Priority' = (any valid priority),
 'Event Type' = CHANGE_OF_RELIABILITY,
 '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)
 7. VERIFY Reliability = NO_FAULT_DETECTED
 8. VERIFY pCurrentState = NORMAL
 9. MAKE (remove the fault condition)
 10. WAIT (pTimeDelayNormal)
 11. WAIT **Notification Fail Time**
 12. CHECK (that the IUT did not send any event notifications for O1)
 13. MAKE (a condition exist that would cause O1 to generate a TO_NORMAL transition)
 14. WAIT **Notification Fail Time**
 15. VERIFY Reliability = NO_FAULT_DETECTED
 16. VERIFY pCurrentState = NORMAL
 17. CHECK (that the IUT did not send any event notifications for O1)

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

7.3.1.21.2 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 $\geq$ 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)
- ELSE
4. MAKE (a condition exist which would cause O1 to transition to FAULT, if Event_Detection_Enable were TRUE)
5. WAIT **Notification Fail Time**
6. CHECK (that the IUT did not send any event notifications due to this condition)
7. VERIFY pCurrentState = NORMAL
8. VERIFY Acked_Transitions = (T,T,T)
9. IF (Protocol_Revision is present AND Protocol_Revision $\geq$ 1) THEN
VERIFY Event_Time_Stamps = [NO_TS, NO_TS, NO_TS]
10. IF (Event_Message_Texts property exists) THEN
VERIFY Event_Message_Texts = ["", "", ""]

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.2.4.

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

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.3.4.

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.

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

BACnet Reference Clause: 12.5.

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, Object_Property_Reference, Window_Interval, and Window_Samples in turn. After each reinitialization the TD pauses and verifies that new sampling has begun.

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

Test Steps:

1. VERIFY Minimum_Value = (a value x : $-\text{INF} < x < \text{INF}$),
2. VERIFY Average_Value = (a value $\neq \text{NaN}$),
3. VERIFY Maximum_Value = (a value x : $\text{Minimum\_Value} \leq x < \text{INF}$),
4. VERIFY Attempted_Samples = (a value $x > 0$),
5. VERIFY Valid_Samples = (a value $x > 0$),
6. WRITE Attempted_Samples = 0,
7. VERIFY Attempted_Samples = 0,
8. VERIFY Minimum_Value = INF,

9. VERIFY Maximum_Value = -INF,
10. VERIFY Average_Value = NaN,
11. VERIFY Valid_Samples = 0,
12. WAIT (at least two sample times),
13. VERIFY Minimum_Value = (a value x: $-\text{INF} < x < \text{INF}$),
14. VERIFY Average_Value = (a value $\neq \text{NaN}$),
15. VERIFY Maximum_Value = (a value x: $\text{Minimum\_Value} \leq x < \text{INF}$),
16. VERIFY Attempted_Samples = (a value $x \geq 2$),
17. VERIFY Valid_Samples = (a value $x \geq 2$),
18. WRITE Window_Interval = (any new value that will result in an appropriate sample time),
19. VERIFY Attempted_Samples = 0,
20. VERIFY Minimum_Value = INF,
21. VERIFY Maximum_Value = -INF,
22. VERIFY Average_Value = NaN,
23. VERIFY Valid_Samples = 0,
24. WAIT (at least two sample times),
25. VERIFY Minimum_Value = (a value x: $-\text{INF} < x < \text{INF}$),
26. VERIFY Average_Value = (a value $\neq \text{NaN}$),
27. VERIFY Maximum_Value = (a value x: $\text{Minimum\_Value} \leq x < \text{INF}$),
28. VERIFY Attempted_Samples = (a value $x \geq 2$),
29. VERIFY Valid_Samples = (a value $x \geq 2$),
30. WRITE Window_Samples = (any new value that will result in an appropriate sample time),
31. VERIFY Attempted_Samples = 0,
32. VERIFY Minimum_Value = INF,
33. VERIFY Maximum_Value = -INF,
34. VERIFY Average_Value = NaN,
35. VERIFY Valid_Samples = 0,
36. IF (Object_Property_Reference is writable) THEN {
 - WAIT (at least two sample times),
 - VERIFY Minimum_Value = (a value x: $-\text{INF} < x < \text{INF}$),
 - 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),
 - VERIFY Attempted_Samples = 0,
 - VERIFY Minimum_Value = INF,
 - VERIFY Maximum_Value = -INF,
 - VERIFY Average_Value = NaN,
 - VERIFY Valid_Samples = 0

7.3.2.4.2 Managing the Sample Window

BACnet Reference Clause: 12.5.15

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

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

Configuration Requirements: The IUT shall be configured with an Averaging object used to monitor a property that can be controlled by the testing agent or by the TD. The sampling interval shall be configured to allow time to change the monitored property value and to determine if each of the properties Minimum_Value, Average_Value, Maximum_Value, Attempted_Samples, and Valid_Samples correctly changes after each sample interval.

Test Steps:

1. WRITE Attempted_Samples = 0,
2. VERIFY Attempted_Samples = 0,
3. VERIFY Minimum_Value = INF,
4. VERIFY Maximum_Value = -INF,
5. VERIFY Average_Value = NaN,
6. VERIFY Valid_Samples = 0,
7. REPEAT X = (1 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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.6.4.

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

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

BACnet Reference Clause: 12.6.11.

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

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

BACnet Reference Clause: 12.7.4.

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

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

BACnet Reference Clause: 12.7.11.

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 Test

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

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

BACnet Reference Clause: 12.9.

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.

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

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

BACnet Reference Clause: 12.9.

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.

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

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

BACnet Reference Clause: 12.9.

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.

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 Test

7.3.2.9.1 All Writes Successful with Post Delay Test

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

BACnet Reference Clause: 12.10.8.

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 Simple-ACK-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

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

BACnet Reference Clause: 12.10.8.

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.

Test Steps:

1. WRITE Present_Value = X
2. RECEIVE Simple-ACK-PDU
3. VERIFY In_Process = TRUE
4. CHECK (for the externally visible actions and verify that they stop when the failure occurs)
5. VERIFY In_Process = FALSE,
6. VERIFY All_Writes_Successful = FALSE
7. WRITE Present_Value = Y
8. RECEIVE Simple-ACK-PDU
9. VERIFY In_Process = TRUE
10. CHECK (for the externally visible actions and verify that they continue to the end after the failure occurs)
11. VERIFY In_Process = FALSE,
12. VERIFY All_Writes_Successful = FALSE

7.3.2.9.3 External Writes Test

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

BACnet Reference Clause: 12.10.8.

Purpose: To verify that a Command object can write to external objects. If the IUT does not support writing to external objects from a Command object this test shall be omitted.

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 Present_Value of (Analog Value, 0) in the TD.

Test Steps:

1. WRITE Present_Value = X
2. RECEIVE Simple-ACK-PDU
3. BEFORE **External Command Fail Time**
RECEIVE WriteProperty-Request,
 'Object Identifier' = (Analog Value, 0),
 'Property Identifier' = Present_Value,
 'Property Value' = (any Real value)
4. VERIFY In_Process = TRUE
5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY In_Process = FALSE

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.

7.3.2.9.4 Empty Action List Test

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.10.8.

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 Simple-ACK-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

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.10.8.

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 Simple-ACK-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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 12.10.8 and 12.10.9.

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

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.10.8 and 12.10.9.

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. TRANSMIT WriteProperty-Request,


```

        'Object Identifier' = O,
        'Property Identifier' = Present_Value,
        'Property 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

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: 12.10.8 and 12.10.9

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

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

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

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

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

7.3.2.9.9 Action_Text Size Changes Action Size Test

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: 12.10.8 and 12.10.9

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

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

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

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

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

7.3.2.10 Device Object Tests

Most necessary 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.

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 L₂ and L₃ 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' = P₁,
'Monitored Object Identifier' = O₁,
'Issue Confirmed Notifications' = TRUE,
'Lifetime' = L₁
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
RECEIVE ConfirmedCOVNotification-Request,
'Subscriber Process Identifier' = P₁,
'Initiating Device Identifier' = IUT,
'Monitored Object Identifier' = O₁,
'Time Remaining' = (a value approximately equal to L₁),
'List of Values' = (values appropriate to the object type of the monitored object)
4. TRANSMIT BACnet-SimpleACK-PDU
5. VERIFY Active_COV_Subscriptions = {{ {TD, P₁}, {O₁, Present_Value }, TRUE, (a value less than L₁), (a valid Increment if the property is REAL) }}
6. TRANSMIT SubscribeCOV-Request,

- 'Subscriber Process Identifier' = P₂,
 'Monitored Object Identifier' = O₂,
 'Issue Confirmed Notifications' = FALSE,
 'Lifetime' = L₂
7. RECEIVE BACnet-SimpleACK-PDU
8. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = P₂,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O₂,
 'Time Remaining' = (a value approximately equal to L₂),
 'List of Values' = (values appropriate to the object type of the monitored object)
9. VERIFY Active_COV_Subscriptions =
 {{ {TD, P₁}, {O₁, Present_Value}, TRUE, (a value less than L₁),
 (a valid Increment if the property is REAL)},
 {{ {TD, P₂}, {O₂, Present_Value}, FALSE, (a value less than L₂),
 (a valid Increment if the property is REAL)}}}
10. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = P₃,
 'Monitored Object Identifier' = O₃,
 'Issue Confirmed Notifications' = FALSE,
 'Lifetime' = L₃
11. RECEIVE BACnet-SimpleACK-PDU
12. BEFORE **Notification Fail Time**
 RECEIVE UnconfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = P₃,
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = O₃,
 'Time Remaining' = (a value approximately equal to L₃),
 'List of Values' = (values appropriate to the object type of the monitored object)
13. VERIFY Active_COV_Subscriptions =
 {{ {TD, P₁}, {O₁, Present_Value}, TRUE, (a value less than L₁),
 (a valid Increment if the property is REAL)},
 {{ {TD, P₂}, {O₂, Present_Value}, FALSE, (a value less than L₂),
 (a valid Increment if the property is REAL)},
 {{ {TD, P₃}, {O₃, Present_Value}, FALSE, (a value less than L₃),
 (a valid Increment if the property is REAL)}}}
14. WAIT L₁ + the IUT's timer granularity

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, O₁, O₂, and O₃, for which the device supports SubscribeCOVProperty. 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 L₂ and L₃ 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

Dependencies: ReadProperty Service Execution Tests, 9.18

BACnet Reference Clause: 12.11.34

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

Dependencies: ReadProperty Service Execution Tests, 9.18

BACnet Reference Clause: 12.11.34

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

Dependencies: ReadProperty Service Execution Tests, 9.18

BACnet Reference Clause: 12.11.34

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

Dependencies: ReadProperty Service Execution Tests, 9.18

BACnet Reference Clause: 12.11.34

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

Test Concept: The Database_Revision property of the Device object is read. An object's name 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 taking wrapping into account)

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

Dependencies: WriteProperty Service Execution Tests, 9.22

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 Simple-ACK-PDU
3. VERIFY Group_Members = 2, ARRAY INDEX = 0
4. VERIFY Group_Member_Names = 2, ARRAY INDEX = 0
5. VERIFY Present_Value = 2, ARRAY INDEX = 0
6. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Global Group object being tested),
 'Property Identifier' = Group_Members,
 'Property Array Index' = 0,
 'Property Value' = (some value greater than 2)
7. RECEIVE Simple-ACK-PDU
8. VERIFY Group_Members = (the value written in step 6), ARRAY INDEX = 0
9. VERIFY Group_Member_Names = (the value written in step 6), ARRAY INDEX = 0
10. VERIFY Present_Value = (the value written in step 6), ARRAY INDEX = 0
11. VERIFY Group_Members = (a BACnetDeviceObjectPropertyReference containing
 (Device, Instance number 4194303)), ARRAY INDEX = (some value from 3 through the value written in step
 6)
12. VERIFY Group_Member_Names = (an empty string),
 ARRAY INDEX = (some value from 3 through the value written in step 6)
13. VERIFY Present_Value = 'Access_Result' = PropertyAccessError (PROPERTY,
 VALUE_NOT_INITIALIZED),
 ARRAY INDEX = (some value from 3 through the value written in step 6)
14. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Global Group object being tested),
 'Property Identifier' = Group_Members,
 'Property Value' = (a one-element array containing any valid BACnetDeviceObjectPropertyReference)
15. RECEIVE Simple-ACK-PDU
16. VERIFY Group_Members = 1, ARRAY INDEX = 0
17. VERIFY Group_Member_Names = 1, ARRAY INDEX = 0
18. VERIFY Present_Value = 1, ARRAY INDEX = 0
19. VERIFY Group_Members = (the array written in step 14)
20. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Global Group object being tested),
 'Property Identifier' = Group_Members,
 'Property Value' = (an array of two or more valid BACnetDeviceObjectPropertyReference values)
21. RECEIVE Simple-ACK-PDU
22. VERIFY Group_Members = (the size of the array written in step 20), ARRAY INDEX = 0
23. VERIFY Group_Member_Names = (the size of the array written in step 20), ARRAY INDEX = 0
24. VERIFY Present_Value = (the size of the array written in step 20), ARRAY INDEX = 0
25. VERIFY Group_Members = (the array written in step 20)
26. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Global Group object being tested),
 'Property Identifier' = Group_Members,
 'Property Array Index' = 0,
 'Property Value' = (some value between 0 and the size of the array written in step 20)
27. RECEIVE Simple-ACK-PDU
28. VERIFY Group_Members = (the size of the array written in step 26), ARRAY INDEX = 0
29. VERIFY Group_Member_Names = (the size of the array written in step 26), ARRAY INDEX = 0
30. VERIFY Present_Value = (the size of the array written in step 26), ARRAY INDEX = 0

7.3.2.13.2 Resizing Group_Members by Writing Group_Member_Names Property Test

Dependencies: WriteProperty Service Execution Tests, 9.22

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

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

7.3.2.14 Group Object Test

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

BACnet Reference Clause: 12.14.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.15.5.

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.

7.3.2.16 Life Safety Zone Object Tests

7.3.2.16.1 Tracking Value Test

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.16.5.

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**

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.17.12.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.17.13.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.17.16.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 12.18.11 and 12.18.12.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.18.4.

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

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: 12.18.11 and 12.18.12

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

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

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

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 Simple-ACK-PDU
3. VERIFY Number_Of_States = 2
4. VERIFY State_Text = 2, ARRAY INDEX = 0
5. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = (some value greater than 2)
6. RECEIVE Simple-ACK-PDU
7. VERIFY Number_Of_States = (the value written in step 5)
8. VERIFY State_Text = (the value written in step 5), ARRAY INDEX = 0
9. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = State_Text,
 'Property Value' = (State_Text array of length 2)
10. RECEIVE Simple-ACK-PDU
11. VERIFY Number_Of_States = 2
12. VERIFY State_Text = 2, ARRAY INDEX = 0
13. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = State_Text,
 'Property Value' = (State_Text array of length greater than 2)
14. RECEIVE Simple-ACK-PDU
15. VERIFY Number_Of_States = (the length of the array written in step 13)
16. VERIFY State_Text = (the length of the array written in step 13), ARRAY INDEX = 0
17. TRANSMIT WriteProperty-Request,
 'Object Identifier' = (the Multi-state Input object being tested),
 'Property Identifier' = Number_Of_States,
 'Property Value' = 2
18. RECEIVE Simple-ACK-PDU
19. VERIFY State_Text = (an array consisting of elements 1 & 2 from the array written in step 13)
20. VERIFY Number_Of_States = 2

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 12.19.11 and 12.19.12.

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

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

BACnet Reference Clause: 12.19.4.

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

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: 12.19.11 and 12.19.12

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 12.20.10 and 12.20.11.

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

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: 12.20.10 and 12.20.11

Test Steps:

Tests to verify the Number_Of_States value and State_Text array size of Multi-state Value objects are defined in 7.3.2.18.5. 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, N₁, shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, N₂, 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, 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 BACnetSimpleACK-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, 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 makes parameters of a custom program network visible. Since BACnet does not define the functionality of the program there are no standard tests to verify this functionality.

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 D_1 , D_2 , 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

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

BACnet Reference Clause: 12.24.6.

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 V_1 .

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 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' = D_2)
| (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_2)
| MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**

7. VERIFY Present_Value = V₁

7.3.2.23.2 Weekly_Schedule Property Test

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

BACnet Reference Clause: 12.24.7.

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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date matching a WeekNDay's 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date matching a WeekNDay's 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date matching a WeekNDay's 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

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

BACnet Reference Clause: 12.24.8.

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

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-8. 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a 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

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

BACnet Reference Clauses: 12.24.7, 12.24.8.

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

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

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

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

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

BACnet Reference Clauses: 12.24.4, 12.24.7, 12.24.8, 12.24.9.

Purpose: To verify the restoration behavior in an Exception_Schedule.

Test Concept: The IUT is configured with a Schedule object containing an Exception_Schedule with BACnetTimeValue entries that do not include the time 00:00. The local date and time are changed to a value between 00:00 and the first entry in the Exception_Schedule. Present_Value is read to verify that 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

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

BACnet Reference Clause: 12.24.4, 12.24.7, 12.24.9.

Purpose: To verify the restoration behavior in a Weekly_Schedule.

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

Present_Value is checked again to verify that it contains the Schedule_Default value, or V_{last} for implementations with a Protocol_Revision less than 4.

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

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D_1) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_1) |
MAKE (the local date and time = D_1)
2. **WAIT Schedule Evaluation Fail Time**
3. IF (Protocol_Revision is present AND Protocol_Revision $\geq$ 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}

7.3.2.23.7 List_Of_Object_Property_Reference Internal Test

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

BACnet Reference Clause: 12.24.10.

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. D_1 represents the date and time of the first of these two BACnetTimeValues, with corresponding value V_1 , while D_2 and V_2 (a value distinguishable from V_1) represent the second BACnetTimeValue. A time D_t is defined to occur between D_1 and D_2 .

Test Steps:

1. (TRANSMIT TimeSynchronization-Request, 'Time' = D_t) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D₁) |
MAKE (the local date and time = D_t)
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

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

BACnet Reference Clause: 12.24.10.

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 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. (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

Dependencies: WriteProperty Service Execution Tests, 9.22

BACnet Reference Clauses: Exception_Schedule, 12.24.8

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

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

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

Test Steps:

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

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

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 V₁. 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 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 = V_r
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 = 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' = D_2)
 - | (TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_2)

| 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 = V_r
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 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 7-16. 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 the last week of the month. 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 6, 2D. Date is in the last week of the month, 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 6, and 2D. Date is not in the last week of the month.	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

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

BACnet Reference Clause: 12.24.8.

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

Test Concept: The IUT's local date and time are changed to values that are selected by the TD based on the criteria in Table 7-16.1. 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

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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a 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.

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 D_N) 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a 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 $D_i =$ (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)
 }

7.3.2.23.10.3.10 Revision 4 Calendar Entry WeekNDay Odd-Numbered Month Test

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date matching a WeekNDay's 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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that 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

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

BACnet Reference Clause: 12.24.9.

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 = D_1)
3. **WAIT Schedule Evaluation Fail Time**
 4. VERIFY Present_Value = 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 Present_Value = V_{default}
 8. IF (the Schedule object is configured with an Exception_Schedule) THEN
 9. (TRANSMIT TimeSynchronization-Request, 'Time' = D_3) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_3) |
MAKE (the local date and time = D_3)
 10. **WAIT Schedule Evaluation Fail Time**
 11. VERIFY Present_Value = V_3
 12. (TRANSMIT TimeSynchronization-Request, 'Time' = D_4) |
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D_4) |
MAKE (the local date and time = D_4)
 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

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

BACnet Reference Clauses: 12.24.7 and 12.24.8.

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

BACnet Reference Clauses: 12.24, 12.24.10

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

Test Concept: Two Date/Time, values, 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 the IUT cannot be configured to these requirements, then this test shall be omitted. Other properties 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 .

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. REPEAT P = (writable property in List_Of_Property_References)
 VERIFY P = V₁
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D₂)
 | (TRANSMIT UTCTimeSynchronization-Request, 'Time'=D₁)
 | MAKE (the local date and time = D₂)
6. WAIT(**Schedule Evaluation Fail Time**)
7. VERIFY S, Present_Value = V₂
8. REPEAT P = (writable property in List_Of_Property_References)
 VERIFY P = V₂

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

7.3.2.23.11.2 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.

Dependencies: TimeSynchronization Service Execution Tests, 9.30, UTCTimeSynchronization Service Execution Tests, 9.31

BACnet Reference Clauses: 12.24, 12.24.9

Test Concept: Two Date/Time values, D₁ and D₂, are chosen by the TD based on the criteria in Table 7-18 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, S, 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, 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 D₁ or D₂, 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

Dependencies: TimeSynchronization Service Execution Tests, 9.26, UTCTimeSynchronization Service Execution Tests, 9.31

BACnet Reference Clauses: 12.24, 12.24.10

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

Dependencies: TimeSynchronization Services Execution Tests, 9.26; UTCTimeSynchronization Service Execution Tests, 9.31

BACnet Reference Clauses: 12.24, 12.24.9

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
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.

#### 7.3.2.23.12 Revision 4 Midnight Evaluation Test

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

BACnet Reference Clause: 12.24.4.

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<sub>default</sub>, and containing at least one of, and if possible, both (non-overlapping):

- a Weekly\_Schedule containing a time-value pair at time D<sub>1</sub> (not 00:00) with a non-NULL value V<sub>1</sub> different from V<sub>default</sub>, and no scheduled write operations during the day after D<sub>1</sub>, and none on the day following.
- an Exception\_Schedule with an event occurring on a day different from D<sub>1</sub>, containing a time-value pair at time D<sub>3</sub> with a non-NULL value V<sub>3</sub> different from V<sub>default</sub>, and no scheduled write operations during the day after D<sub>3</sub>, and none on the day following.

Two additional times, used in the execution of the test, are defined as follows:

- D<sub>2</sub> occurring on the same day as D<sub>1</sub>, after D<sub>1</sub>, and before midnight.
- D<sub>4</sub> occurring on the same day as D<sub>3</sub>, after D<sub>3</sub>, and before midnight.

It is recommended that to minimize testing time, D<sub>1</sub> through D<sub>4</sub> 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 <sub>1</sub> | D <sub>2</sub> | 00:00 | D <sub>3</sub> | D <sub>4</sub> | 00:00 |
|---------------------|----------------|----------------|-------|----------------|----------------|-------|
| Exception_Schedule: | -              | -              | -     | V <sub>3</sub> | -              | -     |
| Weekly_Schedule:    | V <sub>1</sub> | -              | -     | -              | -              | -     |

|                |                |                      |                |                      |
|----------------|----------------|----------------------|----------------|----------------------|
| Present Value: | V <sub>1</sub> | V <sub>default</sub> | V <sub>3</sub> | V <sub>default</sub> |
|----------------|----------------|----------------------|----------------|----------------------|

Test Steps:

1. IF (the Schedule object is configured with a Weekly\_Schedule) THEN
2. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>2</sub>) |  
MAKE (the local date and time = D<sub>2</sub>)
3. WAIT **Schedule Evaluation Fail Time**
4. VERIFY Present\_Value = V<sub>1</sub>
5. WAIT (until 00:00)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present\_Value = V<sub>default</sub>
8. IF (the Schedule object is configured with an Exception\_Schedule) THEN
9. (TRANSMIT TimeSynchronization-Request, 'Time' = D<sub>4</sub>) |  
(TRANSMIT UTCTimeSynchronization-Request, 'Time' = D<sub>4</sub>) |  
MAKE (the local date and time = D<sub>4</sub>)
10. WAIT **Schedule Evaluation Fail Time**
11. VERIFY Present\_Value = V<sub>3</sub>
12. WAIT (until 00:00)
13. WAIT **Schedule Evaluation Fail Time**
14. VERIFY Present\_Value = V<sub>default</sub>

### 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

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

BACnet Reference Clause: 12.25.5.

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**

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

Purpose: To verify that logging is enabled at the time specified by Start\_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 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

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

BACnet Reference Clause: 12.25.7.

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

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

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

Test Steps:

1. WRITE Enable = FALSE
2. WAIT **Internal Processing Fail Time**
3. WRITE Record\_Count = 0
4. WRITE Enable = TRUE
5. READ X = Total\_Record\_Count
6. WAIT **Internal Processing Fail Time**
7. MAKE (IUT collect another record)
8. WAIT (**Notification Fail Time** + **Internal Processing Fail Time**)

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

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

```

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

```

#### **7.3.2.24.4 Log\_Interval Test**

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

Purpose: To verify that the logging period is controlled by Log\_Interval.

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

Configuration Requirements: Start\_Time, if present, shall be configured with a date and time preceding the beginning of the test. Stop\_Time, if present shall be configured with the latest possible date and time, in order that it occur after the end of the test. Stop\_When\_Full, if configurable, shall be set to FALSE. Enable shall be set to TRUE. Non-zero values shall be chosen for Log\_Interval in accordance with the range and resolution specified by the manufacturer for this property.

Test Steps:

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

#### **7.3.2.24.5 COV\_Resubscription\_Interval Test**

Dependencies: Confirmed Notifications Subscription, 8.10.1.

BACnet Reference Clause: 12.25.10.

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  $\geq$  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  $\geq$  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  $\leq$  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  $\geq$  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  $\geq$  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

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

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.

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 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

##### 7.3.2.24.6.2 Stop\_When\_Full FALSE Test

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

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

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

BACnet Reference Clause: 12.25.13.

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

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

Purpose: To verify that the Record\_Count property indicates the number of records that are stored in the Log\_Buffer.

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

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

BACnet Reference Clause: 12.25.16.

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 100<sup>th</sup> 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. Record\_Count is set to zero and Total\_Record\_Count is read. 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. WRITE Record\_Count = 0
2. WAIT **Internal Processing Fail Time**
3. READ X = Total\_Record\_Count
4. READ Y = Record\_Count
5. WRITE Enable = TRUE
6. WHILE ( Record\_Count = Y + 1 ) DO { }
7. WRITE Enable = FALSE
8. WAIT **Internal Processing Fail Time**
9. 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

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

BACnet-2001 Reference Clauses: 12.23.19, 12.23.20, and 12.23.26.

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

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

BACnet Reference Clauses: 12.25.5, 12.25.9, and 12.25.10.

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

Dependencies: ReadRange Service Execution Tests, 9.21; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.25.14.

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

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

Test Configuration: The logging object, 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.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Record\_Count = 0
3. VERIFY Record\_Count = 1
4. TRANSMIT ReadRange

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

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

#### 7.3.2.24.14 Time\_Change Test

Dependencies: ReadRange Service Execution Tests, 9.21; (TimeSynchronization Service Execution Tests, 9.30 or UTCTimeSynchronization Service Execution Tests, 9.31)

BACnet Reference Clause: 12.25.14.

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

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

Test Configuration: The log is configured to acquire data by whatever means available. The Log\_Buffer should be cleared such that the Record\_Count is 0. Configure the logging such that the entire test may be run without the trend buffer overflowing.

Test Steps:

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

#### 7.3.2.24.15 COV-Sampling Verification Test

Dependencies: ReadRange Service Execution, 9.21

BACnet Reference Clause: 12.25.10-11; 13.1

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 a COV\_Increment property that is set to a value other than 0.0, the Client\_COV\_Increment is set to a value other than 0.0 or NULL, or the monitored property is not of datatype REAL.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Record\_Count = 0
3. WRITE Interval = 0
4. WRITE Enable = TRUE
5. WAIT ( 10 seconds )
6. MAKE (monitored property change its value)
7. WAIT ( 60 seconds )
8. MAKE (monitored property change its value)
9. WAIT ( 90 seconds )
10. MAKE (monitored property change its value)
11. WAIT ( 40 seconds )
12. MAKE (monitored property change its value)
13. WAIT **Notification Fail Time**
14. WRITE Enable = FALSE
15. READ N = RecordCount
16. REPEAT X = (1 through 4) {

```

TRANSMIT ReadRange
 'Object Identifier' = O1,
 'Property Identifier' = Log_Buffer,
 'Reference Index' = N-5+X,
 'Count' = 1
RECEIVE ReadRangeAck
 'Object Identifier' = O1,
 'Property Identifier' = Log_Buffer,
 'Result Flags' = (False, False, False),
 'Item Count' = 1
 'Item Data' = ((one data record storing the timestamp in TS[X]))
}

```

17. CHECK( TS[2] - TS[1]  $\approx$  60 seconds )
18. CHECK( TS[3] - TS[2]  $\approx$  90 seconds )
19. CHECK( TS[4] - TS[3]  $\approx$  40 seconds )

#### 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

Dependencies: ReadRange Service Execution, 9.21;

BACnet Reference Clause: 12.25.27, 12.30.12

Purpose: To verify that logged samples are based on the triggered Logging\_Type.

Test Concept: The log 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 IUT shall be configured such that the monitored object's Logging\_Type is set to TRIGGERED.

Test Steps:

1. WRITE Enable = FALSE
2. WRITE Record\_Count = 0 -- results in a buffer purged record
3. WRITE Enable = TRUE -- results in a logging enable record
4. WAIT (10 seconds)
5. WRITE Trigger = TRUE
6. WAIT (20 seconds)
7. WRITE Trigger = TRUE
8. WAIT (40 seconds)
9. WRITE Trigger = TRUE
10. WAIT (30 seconds)
11. WRITE Enable = FALSE -- results in a logging disabled record
12. VERIFY RecordCount = 6
13. REPEAT X = (1 through 6)
  - TRANSMIT ReadRange
    - 'Object Identifier' = O1,
    - 'Property Identifier' = Log\_Buffer,
    - 'Reference Index' = X,
    - 'Count' = 1
  - RECEIVE ReadRangeAck
    - 'Object Identifier' = O1,
    - 'Property Identifier' = Log\_Buffer,
    - 'Result Flags' = (?, ?, False),
    - 'Item Count' = 1
    - 'Item Data' = ( (one data record storing the timestamp in TS[X]) )
14. CHECK ( TS[3] - TS[2]  $\approx$  10 seconds )
15. CHECK ( TS[4] - TS[3]  $\approx$  20 seconds )
16. CHECK ( TS[5] - TS[4]  $\approx$  40 seconds )
17. CHECK ( TS[6] - TS[5]  $\approx$  30 seconds )

#### 7.3.2.25 Event Log Tests

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

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

##### 7.3.2.25.1 Internal Logging of Notifications



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

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

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

Test Steps:

1. WRITE Enable = TRUE
2. MAKE (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, ?, TRUE}, |
| 'Item Count' =          | 2,                |

'Item Data' = (logged data that matches the information received in steps 3 and 6, except that Process\_Identifier may be any value and is not required to match)

13. CHECK ( $T2 > T1$ , and that the notifications were logged in order)

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

### 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.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,
  - 'Process Identifier' = (any valid process identifier),
  - 'Initiating Device Identifier' = TD,
  - 'Event Object Identifier' = (any valid object identifier),
  - 'Time Stamp' = (T1, any valid timestamp),
  - 'Notification Class' = (any valid notification class),
  - 'Priority' = (any valid priority),
  - 'Event Type' = (any standard event type),
  - 'Message Text' = (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, ?, TRUE},  
     'Item Count' = 2,  
     'Item Data' = (logged data that matches the information received in steps 2 and 4,  
                     except that Process\_Identifier can be any value and is not required  
 to  
                     match)
9. CHECK (that the events were logged in the order in which they were received)

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

### 7.3.2.25.3 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.

Test Steps:

1. WRITE Enable = TRUE
2. READ RC = LO1, Record\_Count
3. MAKE (a condition exist that will cause the object to generate an event transition)
4. WAIT D1
5. 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)
6. TRANSMIT BACnet-SimpleACK-PDU
7. 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)
8. RECEIVE BACnet-SimpleACK-PDU
9. 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                                                                            |
| 10. TRANSMIT BACnet-SimpleACK-PDU |                                                                               |
| 11. TRANSMIT ReadRange-Request,   |                                                                               |
| 'Object Identifier' =             | LO1,                                                                          |
| 'Property Identifier' =           | Log_Buffer,                                                                   |
| 'Reference Index' =               | RC,                                                                           |
| 'Count' =                         | -1                                                                            |
| 12. RECEIVE ReadRange-ACK,        |                                                                               |
| 'Object Identifier' =             | LO1,                                                                          |
| 'Property Identifier' =           | Log_Buffer,                                                                   |
| 'Result Flags' =                  | {FALSE, ?, TRUE},                                                             |
| 'Item Count' =                    | 1,                                                                            |
| 'Item Data' =                     | (logged data that matches the information received in step                    |
| 4,                                |                                                                               |
|                                   | except that Process_Identifier can be any value and is not required to match) |

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

#### 7.3.2.25.4 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.

Test Steps:

1. WRITE Enable = TRUE
2. TRANSMIT ConfirmedEventNotification-Request,
 

|                                  |                                                 |
|----------------------------------|-------------------------------------------------|
| 'Process Identifier' =           | (any valid process identifier),                 |
| 'Initiating Device Identifier' = | IUT,                                            |
| 'Event Object Identifier' =      | (any valid object identifier),                  |
| 'Time Stamp' =                   | (T1, any valid timestamp),                      |
| 'Notification Class' =           | (any valid notification class),                 |
| 'Priority' =                     | (any valid priority),                           |
| 'Event Type' =                   | (any standard event type),                      |
| 'Message Text' =                 | (optional, any valid message text),             |
| 'Notify Type' =                  | ACK_NOTIFICATION,                               |
| 'From State' =                   | (state S1, any valid state for this event type) |
3. RECEIVE BACnet-SimpleACK-PDU
4. READ RC = LO1, Record\_Count
5. TRANSMIT ReadRange-Request,

|                           |                                                                                                                                                |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 'Object Identifier' =     | LO1,                                                                                                                                           |
| 'Property Identifier' =   | Log_Buffer,                                                                                                                                    |
| 'Reference Index' =       | RC,                                                                                                                                            |
| 'Count' =                 | -1                                                                                                                                             |
| 6. RECEIVE ReadRange-ACK, |                                                                                                                                                |
| 'Object Identifier' =     | LO1,                                                                                                                                           |
| 'Property Identifier' =   | Log_Buffer,                                                                                                                                    |
| 'Result Flags' =          | {FALSE, ?, TRUE},                                                                                                                              |
| 'Item Count' =            | 1,                                                                                                                                             |
| 'Item Data' =             | (logged data that matches the information received in step 2,<br>except that Process_Identifier can be any value and is not required<br>match) |
| to                        |                                                                                                                                                |

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

**7.3.2.26 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**

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.29.7 and 12.29.8.

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

Dependencies: WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.29.7 and 12.29.8.

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



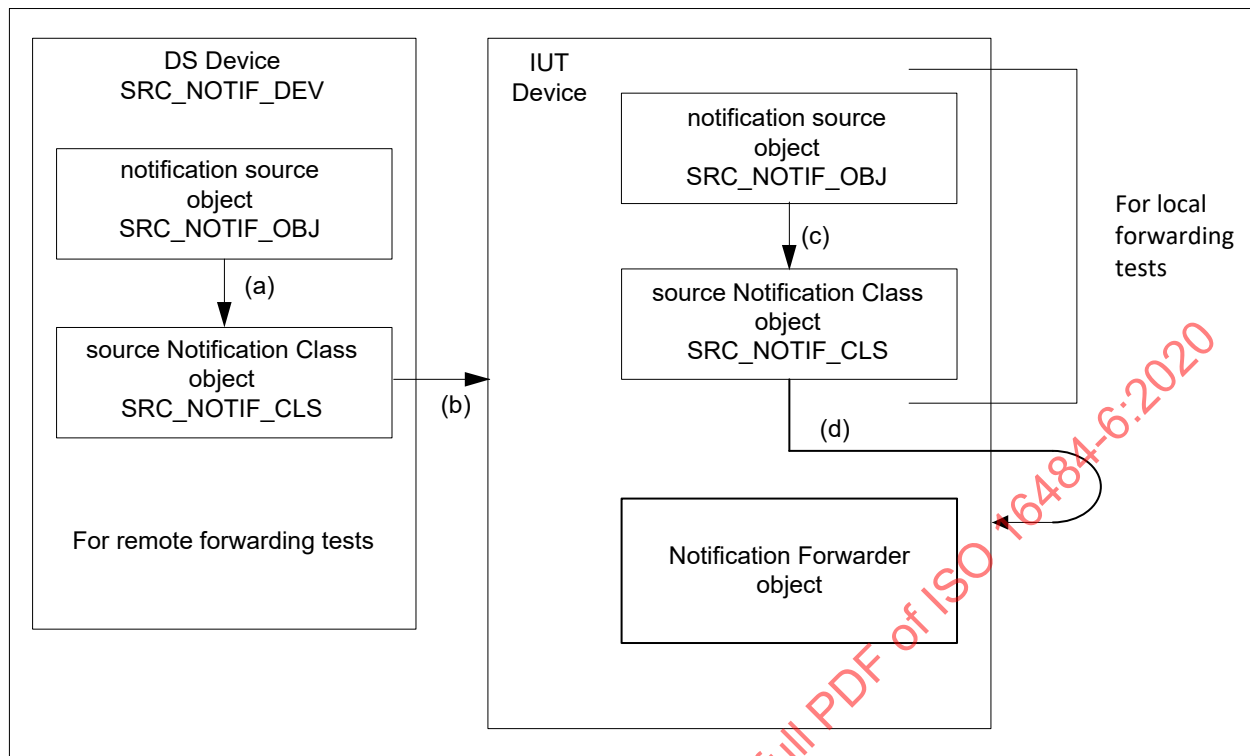


Figure 7-31-1. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 1.

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

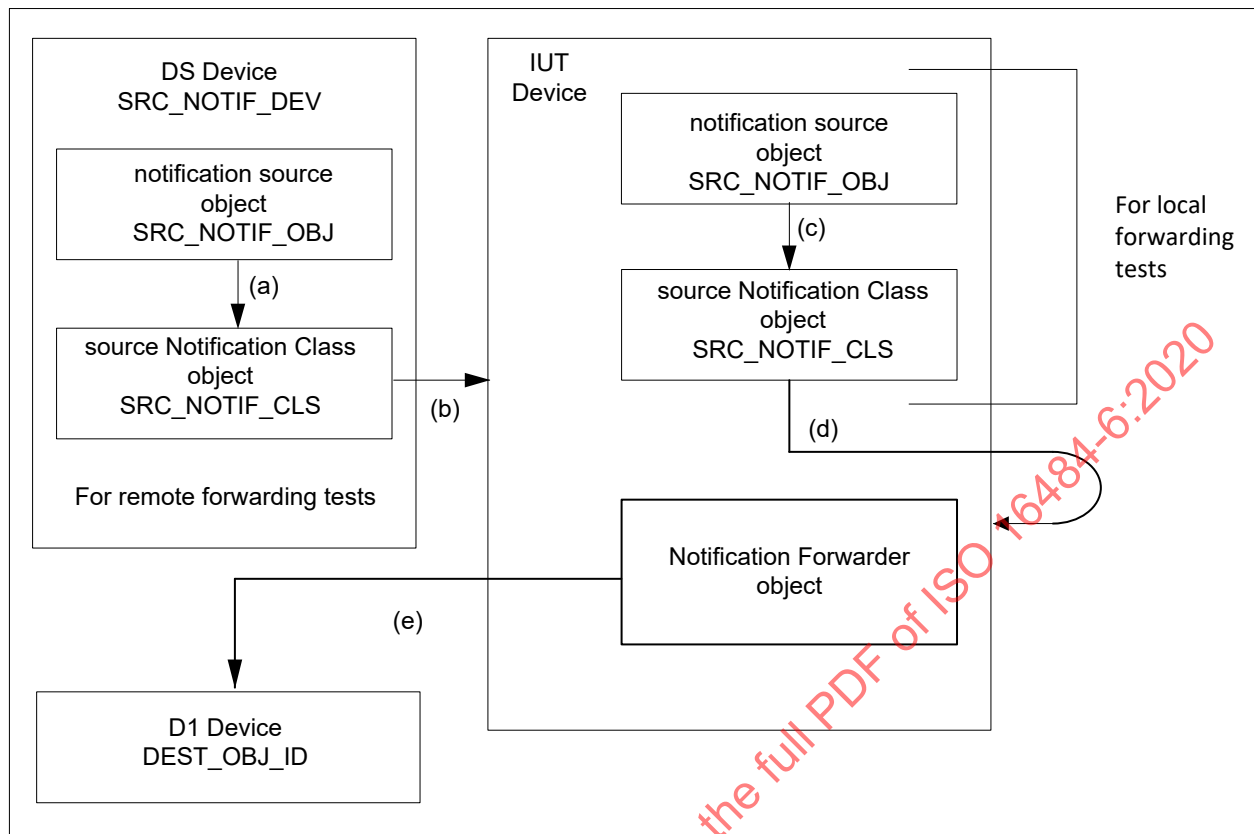


Figure 7-31-2. Logical relationship of objects required for Notification Forwarder functionality tests using base setup 2.

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.

If the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted.



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

Test Steps:

1. MAKE (Recipient\_List =  
 {(all), -- Valid Days  
 (all), -- From Time, To Time  
 DEST\_OBJ\_ID2, -- Recipient D2  
 DEST\_PROCESS\_ID, -- Process Identifier  
 FALSE, -- Issue Confirmed Notifications  
 {T, T, T} -- Transitions  
 }) -- One list element
2. MAKE (Out\_Of\_Service = TRUE)
3. VERIFY Out\_Of\_Service = TRUE
4. VERIFY Status\_Flags = (?, FALSE, ?, 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)
11. TRANSMIT SOURCE = DS, UnconfirmedEventNotification-Request,  
 'Process Identifier' = SRC\_PROCESS\_ID,  
 'Initiating Device Identifier' = SRC\_NOTIF\_DEV,  
 'Event Object Identifier' = SRC\_NOTIF\_OBJ,  
 'Time Stamp' = (any valid time stamp),  
 'Notification Class' = SRC\_NOTIF\_CLS,  
 'Priority' = (any valid priority),  
 'Event Type' = (any valid event type),  
 'Message Text' = (optional, any valid message text),  
 'Notify Type' = SRC\_NOTIF\_TYP,  
 'AckRequired' = (any valid value), -- absent if Notify Type is ACK\_NOTIFICATION  
 'From State' = (any valid From\_State), -- absent if Notify Type is ACK\_NOTIFICATION  
 'To State' = (any valid To\_State),  
 'Event Values' = (any valid event values) -- absent if Notify Type is ACK\_NOTIFICATION
12. BEFORE **Notification Fail Time** --The following can be in any order  
 RECEIVE DESTINATION = D1, UnconfirmedEventNotification-Request  
 RECEIVE DESTINATION = D2, UnconfirmedEventNotification-Request

### 7.3.2.30.7 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),<br>(any range sufficient for the duration of the test),<br>DEST_OBJ_ID,<br>DEST_PROCESS_ID,<br>FALSE, | -- Valid Days<br>-- From Time, To Time<br>-- Recipient D1<br>-- Process Identifier<br>-- Issue Confirmed |
| Notifications             | {T, T, T}<br>})                                                                                                                                        | -- Transitions<br>-- 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 7.3.2.31.2 Recipient\_List Forwarding Test and 7.3.2.31.3 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  $\neq$  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:

1. 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 <= TR1 <= 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, DEST\_PROCESS\_ID, FALSE, TR1 })
 

|  |                                             |
|--|---------------------------------------------|
|  | -- Recipient D1                             |
|  | -- Process Identifier                       |
|  | -- Issue Confirmed Notifications            |
|  | -- Time Remaining where (TR-1) <= TR1 <= 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 > test duration and TR2 <> TR |
|                         | }                          | -- One list element                                       |
2. RECEIVE BACnet-SimpleACK-PDU
3. READ SR = Subscribed\_Recipients
4. CHECK (SR = {DEST\_OBJ\_ID, DEST\_PROCESS\_ID, DEST\_CONF\_NOTIF1, TR3 })
 

|  |                                               |
|--|-----------------------------------------------|
|  | -- Recipient D1                               |
|  | -- Process Identifier                         |
|  | -- Issue Confirmed Notifications              |
|  | -- 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 <> P\_ID) DO  
(Next index X)
8. WHILE (Z = 1 to PF1\_SIZE) DO  
IF (Z <> 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 <> 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<br>(all), -- From Time, To Time<br>IUT -- Recipient<br>} -- 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<br>(all), -- From Time, To Time<br>IUT -- Recipient<br>} -- 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 object 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 number 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:

1. 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 |
2. WAIT Notification Fail Time
3. 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:

1. 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                        |
2. 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:

1. 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
2. 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.

Test Configuration: 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.

Test Configuration: 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 ~~Event\_State~~ 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

## 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

**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,                       |
| 'Time Stamp' =                       | T1,                       |
| 'Acknowledgement Source' =           | (any valid value),        |
| 'Time of Acknowledgement' =          | (any valid value)         |
5. 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.

BACnet Reference Clause: 13.6.

Dependencies: SubscribeCOV Service Execution Tests, 9.10; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

### 8.2.1 Change of Value Notification from an Analog Input, Analog Output, and Analog Value Object Present\_Value Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of Analog Input, Analog Output, and Analog Value 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.

Test Steps:

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

1. TRANSMIT SubscribeCOV-Request,
 

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| 'Subscriber Process Identifier' = | (any value > 0 chosen by the TD), |
| 'Monitored Object Identifier' =   | X,                                |
| 'Issue Confirmed Notifications' = | TRUE,                             |
| 'Lifetime' =                      | L                                 |
2. RECEIVE BACnet-SimpleACK-PDU
3. 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
 RECEIVE BACnet-SimpleACK-PDU
 BEFORE Notification Fail Time
 RECEIVE ConfirmedCOVNotification-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Initiating Device Identifier' = IUT,
 'Monitored Object Identifier' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the initial Present_Value and new Status_Flags)
 TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present_Value is now writable) THEN
 WRITE X, Present_Value = (any value that differs from "initial Present_Value" by less than "increment")
 RECEIVE BACnet-SimpleACK-PDU
 ELSE
 MAKE (Present_Value = any value that differs from "initial Present_Value" by less than "increment")
9. WAIT Notification Fail Time
10. CHECK (verify that no COV notification was transmitted)
11. IF (Present_Value is now writable) THEN
 WRITE X, Present_Value = (any value that differs from "initial Present_Value" by an amount greater than
 "increment")
 RECEIVE BACnet-SimpleACK-PDU
 ELSE
 MAKE (Present_Value = any value that differs from "initial Present_Value" 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' = X,
 'Time Remaining' = (any value appropriate for the Lifetime selected),
 'List of Values' = (the new Present_Value and new Status_Flags)
13. TRANSMIT BACnet-SimpleACK-PDU
14. TRANSMIT SubscribeCOV-Request,
 'Subscriber Process Identifier' = (the same value used in step 1),
 'Monitored Object Identifier' = X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
 WRITE X, Out_Of_Service = FALSE
 RECEIVE BACnet-SimpleACK-PDU

```

### 8.2.2 Change of Value Notification from an Analog Input, Analog Output, and Analog Value Object Status\_Flags Property

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out\_Of\_Service property will accomplish this task. For

implementations where it is not possible to write to Status\_Flags or Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

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

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out\_Of\_Service = TRUE | WRITE X, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
     MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. IF (WriteProperty is used in step 5) THEN  
     RECEIVE BACnet-SimpleACK-PDU
7. 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 new Status\_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE X, Out\_Of\_Service = FALSE  
     RECEIVE BACnet-SimpleACK-PDU

### 8.2.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present\_Value Property

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

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

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

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

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

#### 8.2.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status\_Flags Property

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out\_Of\_Service property will accomplish this task. For implementations where it is not possible to write to Status\_Flags or Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

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

1. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (any value > 0 chosen by the TD),  
     'Monitored Object Identifier' = X,  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = L
2. RECEIVE BACnet-SimpleACK-PDU
3. RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out\_Of\_Service = TRUE | WRITE X, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
     MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. IF (WriteProperty is used in step 5) THEN  
     RECEIVE BACnet-SimpleACK-PDU
7. 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 new Status\_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE X, Out\_Of\_Service = FALSE  
     RECEIVE BACnet-SimpleACK-PDU

#### 8.2.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, or Life Safety Zone Object Present\_Value Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present\_Value property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present\_Value of the monitored object is changed and a notification shall be received. The Present\_Value may be changed using the WriteProperty service or by another means such as changing the input signal represented by the object. For some implementations it may be necessary to write to the

Out\_Of\_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

REPEAT X = (one supported object of each type from the set Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone) 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. RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. IF (Out\_Of\_Service is writable) THEN  
     WRITE X, Out\_Of\_Service = TRUE  
     RECEIVE BACnet-SimpleACK-PDU  
     BEFORE **Notification Fail Time**  
         RECEIVE ConfirmedCOVNotification-Request,  
             'Subscriber Process Identifier' = (the same value used in step 1),  
             'Initiating Device Identifier' = IUT,  
             'Monitored Object Identifier' = X,  
             'Time Remaining' = (any value appropriate for the Lifetime selected),  
             'List of Values' = (the initial Present\_Value and the new Status\_Flags)  
         TRANSMIT BACnet-SimpleACK-PDU
6. IF (Present\_Value is now writable) THEN  
     WRITE X, Present\_Value = (any value that differs from "initial value")  
     RECEIVE BACnet-SimpleACK-PDU  
     ELSE  
         MAKE (Present\_Value = any value that differs from "initial value")
7. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same value used in step 1),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = X,  
         'Time Remaining' = (any value appropriate for the Lifetime selected),  
         'List of Values' = (the new Present\_Value and new Status\_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out\_Of\_Service is writable) THEN  
     WRITE X, Out\_Of\_Service = FALSE  
     RECEIVE BACnet-SimpleACK-PDU



### 8.2.6 Change of Value Notification from a Multi-state Input, Multi-state Output Multi-state Value, Life Safety Point, and Life Safety Zone Object Status\_Flags Property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out\_Of\_Service property will accomplish this task. For implementations where it is not possible to write to Status\_Flags or Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped.

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

Test Steps:

REPEAT X = (one supported object of each type from the set Multi-state input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone) 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. RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = X,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value and initial Status\_Flags)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WRITE X, Out\_Of\_Service = TRUE. | WRITE X, Status\_Flags = (a value that differs from "initial Status\_Flags") |  
     MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")
6. IF (WriteProperty is used in step 5) THEN  
     RECEIVE BACnet-SimpleACK-PDU
7. 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 new Status\_Flags)
8. TRANSMIT BACnet-SimpleACK-PDU
9. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = X
10. RECEIVE BACnet-SimpleACK-PDU
11. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE X, Out\_Of\_Service = FALSE  
     RECEIVE BACnet-SimpleACK-PDU

### 8.2.7 Change of Value Notification from Loop Object Present\_Value Property

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.

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. RECEIVE ConfirmedCOVNotification-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Initiating Device Identifier' = IUT,  
     'Monitored Object Identifier' = O1,  
     'Time Remaining' = (any value appropriate for the Lifetime selected),  
     'List of Values' = (the initial Present\_Value, initial Status\_Flags, initial Setpoint, and initial Controlled\_Variable\_Value)
4. TRANSMIT BACnet-SimpleACK-PDU
5. TRANSMIT ReadProperty-Request,  
     'Object Identifier' = O1,  
     'Property Identifier' = COV\_Increment
6. RECEIVE BACnet-ComplexACK-PDU,  
     'Object Identifier' = O1,  
     'Property Identifier' = COV\_Increment,  
     'Property Value' = (a value "increment" that will be used below)
7. IF (Out\_Of\_Service is writable) THEN  
     WRITE O1, Out\_Of\_Service = TRUE  
     BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same value used in step 1),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = O1,  
         'Time Remaining' = (any value appropriate for the Lifetime selected),  
         'List of Values' = (the initial Present\_Value, new Status\_Flags, initial Setpoint, and initial Controlled\_Variable\_Value)
8. TRANSMIT BACnet-SimpleACK-PDU
8. IF (Present\_Value is now writable) THEN  
     WRITE O1, Present\_Value = (any value that differs from "initial Present\_Value" by less than "increment")  
     ELSE  
         MAKE (Present\_Value = any value that differs from "initial Present\_Value" 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 "initial Present\_Value" by an amount greater than "increment")

ELSE

MAKE (Present\_Value = any value that differs from "initial Present\_Value" 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, initial Setpoint, and initial 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 L, Out\_Of\_Service = FALSE

**8.2.8 Change of Value Notification from a Loop Object Status\_Flags Property**

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

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Status\_Flags property of the monitored object is then changed and a notification shall be received. The value of the Status-Flags property can be changed by using the WriteProperty service or by another means. For some implementations writing to the Out\_Of\_Service property will accomplish this task. For implementations where it is not possible to write to Status\_Flags or Out\_Of\_Service or change the Status\_Flags by any other means, this test shall be skipped.

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

Configuration Requirements: At the beginning of the test, the Out\_Of\_Service property shall have a value of FALSE.

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. RECEIVE ConfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same value used in step 1),  
'Initiating Device Identifier' = IUT,  
'Monitored Object Identifier' = O1,  
'Time Remaining' = (any value appropriate for the Lifetime selected),  
'List of Values' = (the initial Present\_Value, initial Status\_Flags, initial Setpoint, and initial Controlled\_Variable\_Value)

4. TRANSMIT BACnet-SimpleACK-PDU

5. WRITE O1, Out\_Of\_Service = TRUE | WRITE O1, Status\_Flags = (a value that differs from "initial Status\_Flags") | MAKE (Status\_Flags = any value that differs from "initial Status\_Flags")

6. IF (WriteProperty is used in step 5) THEN

RECEIVE BACnet-SimpleACK-PDU

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' = O1,  
 'Time Remaining' = (any value appropriate for the Lifetime selected),  
 'List of Values' = (the initial Present\_Value, new Status\_Flags, initial Setpoint, and initial Controlled\_Variable\_Value)
8. TRANSMIT BACnet-SimpleACK-PDU  
 9. TRANSMIT SubscribeCOV-Request,  
     'Subscriber Process Identifier' = (the same value used in step 1),  
     'Monitored Object Identifier' = O1  
 10. RECEIVE BACnet-SimpleACK-PDU  
 11. IF (Out\_Of\_Service was changed in step 5) THEN  
     WRITE L, Out\_Of\_Service = FALSE  
     RECEIVE BACnet-SimpleACK-PDU

### 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:

1. 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

### 8.3 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.

Dependencies: SubscribeCOV Service Execution Tests, 9.10; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 13.7.

#### 8.3.1 Change of Value Notification from an Analog Input, Analog Output, and Analog Value Object Present\_Value Property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property of Analog Input, Analog Output, and Analog Value objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.1 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of

the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

### **8.3.2 Change of Value Notification from an Analog Input, Analog Output, and Analog Value Object Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Analog Input, Analog Output, and Analog Value objects.

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

### **8.3.3 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Present\_Value Property**

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

Test Steps: The steps for this test case are identical to the test steps in 8.2.3 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

### **8.3.4 Change of Value Notification from a Binary Input, Binary Output, and Binary Value Object Status\_Flags Property**

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

Test Steps: The steps for this test case are identical to the test steps in 8.2.4 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

### **8.3.5 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone Object Present\_Value Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present\_Value property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects.

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

### **8.3.6 Change of Value Notification from a Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone Object Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.6 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all of the

ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services. The MAC address used for the notification message shall be such that the TD is one of the recipients.

### **8.3.7 Change of Value Notification from Loop Object Present\_Value Property**

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.

### **8.3.8 Change of Value Notification from a Loop Object Status\_Flags Property**

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Status\_Flags property of a Loop object.

Test Steps: The steps for this test case are identical to the test steps in 8.2.8 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.

### **8.3.9 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.

BACnet Reference Clauses: 13.7.

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. 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.

Test Steps:

1. IF (Restart\_Notification\_Recipients is writable) THEN  
    WRITE(Restart\_Notification\_Recipients = any non-empty list of Recipients)  
ELSE  
    MAKE (Restart\_Notification\_Recipients contain any non-empty list of Recipients)
2. READ T1 = Local\_Time
3. MAKE(the IUT reset)
4. 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))  
    }  
5. VERIFY Time\_Of\_Device\_Restart = T2  
6. CHECK (T1 ~ T2)  
7. VERIFY Last\_Restart\_Reason = R

Note to tester: Not all IUTs can accurately differentiate between the types of restart reasons and thus no requirements are placed on the value returned in step 4. The test shall pass regardless of the order in which the IUT generates the UnconfirmedCOVNotification-Requests in step 4. The value of T2 shall be the same for each notification sent out in step 4.

#### 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 pStatusFlags = (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  $\geq$  13) THEN  
    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
  14. VERIFY pCurrentState = NORMAL
  15. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
    VERIFY Event\_Time\_Stamps = (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  $\geq$  13)  
    OR ((Protocol\_Revision is present AND Protocol\_Revision  $\leq$  12)  
    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  $\geq$  13) THEN

    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)

    VERIFY pCurrentState = OFFNORMAL

IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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  $\geq$  13) THEN

    VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)

    VERIFY pCurrentState = NORMAL

IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN

    VERIFY Event\_Time\_Stamps = (T1, Ta, T2)

}

3. IF ((Protocol\_Revision is present AND Protocol\_Revision  $\leq$  12)

    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 numerical 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 pStatusFlags = (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 pStatusFlags = (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.

Test Steps:

1. VERIFY pCurrentState = NORMAL
2. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN  
VERIFY pStatusFlags = (FALSE, FALSE, FALSE, FALSE)
3. IF (pMonitoredValue is writable) THEN  
WRITE pMonitoredValue = (a different value)  
ELSE  
MAKE (pMonitoredValue 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  $\geq$  13) THEN  
VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
8. VERIFY pCurrentState = OFFNORMAL
9. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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 (pTimeDelayNormal)
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  $\geq$  13) THEN



- VERIFY pStatusFlags = (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)

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.

#### 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  $\geq$  13) THEN  
     VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
11. VERIFY pCurrentState = pHighDiffLimit
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: (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  $\geq$  13) THEN  
     VERIFY pStatusFlags = (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: (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  $\geq$  13) THEN

VERIFY pStatusFlags = (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: (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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  $\geq$  13) THEN  
 VERIFY pStatusFlags = (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.

#### 8.4.7 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),<br>(pPreviousCount: any valid value),<br>(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),<br>(pPreviousCount: CN1),<br>(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 OFF-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. 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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = LIFE_SAFETY_ALARM
 IF (Protocol_Revision is present AND Protocol_Revision ≥ 1) THEN
 VERIFY Event_Time_Stamped = (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 pStatusFlags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = LIFE_SAFETY_ALARM
 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, *, *)
 }
}

```

#### 8.4.8.7 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. MAKE (pMode = different value that maintains pCurrentState as NORMAL)
5. WAIT (pTimeDelayNormal)
6. 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-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
7. TRANSMIT BACnet-SimpleACK-PDU
8. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN  
 VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
9. VERIFY pCurrentState = NORMAL
10. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
 VERIFY Event\_Time\_Stamps = (\*, \*, T1)
11. IF (Event\_Message\_Texts property exists) THEN  
 VERIFY Event\_Message\_Texts = (\*, \*, S1)
12. MAKE (pMonitoredValue have a value that corresponds to an OFFNORMAL state)
13. MAKE (pMode = different value that maintains pCurrentState as OFFNORMAL)
14. WAIT (pTimeDelay)
15. 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' = OFFNORMAL,  
 'Event Values' = pMonitoredValue, pMode, pStatusFlags, pOperationExpected
16. TRANSMIT BACnet-SimpleACK-PDU
17. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  13) THEN



- VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
18. VERIFY pCurrentState = OFFNORMAL
  19. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
VERIFY Event\_Time\_Stamps = (T2, \*, \*)
  20. IF (Event\_Message\_Texts property exists) THEN  
VERIFY Event\_Message\_Texts = (S2, \*, \*)
  21. MAKE (pMonitoredValue have a value that corresponds to a LIFE\_SAFETY\_ALARM state)
  22. MAKE (pMode = different value that maintains pCurrentState = LIFE\_SAFETY\_ALARM)
  23. WAIT (pTimeDelay)
  24. 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' =                   | (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                                   |
  25. TRANSMIT BACnet-SimpleACK-PDU
  26. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 13) THEN  
VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
  27. VERIFY pCurrentState = OFFNORMAL
  28. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
VERIFY Event\_Time\_Stamps = (T3, \*, \*)
  29. 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  $\geq$  13) THEN  
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
  7. VERIFY pCurrentState = OFFNORMAL
  8. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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 pStatusFlags = (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 pStatusFlags = (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)
 }
}

```

#### 8.4.8.10 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  $\geq$  13) THEN  
    VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = LIFE\_SAFETY\_ALARM
8. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
    VERIFY Event\_Time\_Stamp = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (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 pStatusFlags = (TRUE, FALSE, ?, ?)
 VERIFY pCurrentState = OFFNORMAL
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, *, *)
}

```

#### 8.4.9 EXTENDED Tests (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.

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).

Test Steps:

- 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_Stamped = (TS1, *, *)

```

```

}
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 pStatusFlags = (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  $\geq$  13)  
OR ((Protocol\_Revision is present AND Protocol\_Revision  $\leq$  12)  
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  $\geq$  13) THEN  
VERIFY pStatusFlags = (TRUE, FALSE,?,?)
8. VERIFY pCurrentState = OFFNORMAL
9. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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 pStatusFlags = (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 pStatusFlags = (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 ≤ 12)
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  $\geq$  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  $\geq$  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 pStatusFlags = (TRUE, FALSE, ?, ?)
7. VERIFY pCurrentState = HIGH\_LIMIT
8. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 1) THEN  
    VERIFY Event\_Time\_Stamps = (Toffnormal, \*, \*)
9. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = (a value x: (pLowLimit < x < pHighLimit))  
ELSE  
    MAKE (pMonitoredValue have a value x: (pLowLimit < x < pHighLimit))
10. WAIT (pTimeDelayNormal)
11. BEFORE **Notification Fail Time**  
    RECEIVE ConfirmedEventNotification-Request,  
        'Process Identifier' = (any valid process ID),  
        'Initiating Device Identifier' = IUT,  
        'Event Object Identifier' = (the object being tested),



- 'Time Stamp' = (Tnormal: 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  $\geq$  13)) THEN  
     VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
14. VERIFY pCurrentState = NORMAL
15. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
     VERIFY Event\_Time\_Stamps = (Tnormal, \*, Tnormal)
16. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (x < pLowLimit))  
 ELSE  
     MAKE (pMonitoredValue have a value x: (x < pLowLimit))
17. WAIT (pTimeDelay)
18. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (Tlowlimit: 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  $\geq$  13)) THEN  
     VERIFY pStatusFlags = (TRUE, FALSE, ?, ?)
21. VERIFY pCurrentState = LOW\_LIMIT
22. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
     VERIFY Event\_Time\_Stamps = (Tlowlimit, \*, Tnormal)
23. IF (pMonitoredValue is writable) THEN  
     WRITE pMonitoredValue = (a value x: (Low\_Limit < x < High\_Limit))  
 ELSE  
     MAKE (pMonitoredValue have a value x: (Low\_Limit < x < High\_Limit))
24. WAIT (pTimeDelayNormal)
25. BEFORE **Notification Fail Time**  
     RECEIVE ConfirmedEventNotification-Request,  
     'Process Identifier' = (any valid process ID),  
     'Initiating Device Identifier' = IUT,  
     'Event Object Identifier' = (the object being tested),  
     'Time Stamp' = (Tlowtonormal: 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  $\geq$  13)) THEN  
     VERIFY pStatusFlags = (FALSE, FALSE, ?, ?)
28. VERIFY pCurrentState = NORMAL
29. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THEN  
     VERIFY Event\_Time\_Stamp = (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  $\neq$  {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  $\geq$  13) THEN  
     VERIFY pStatusFlags = {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 pStatusFlags = {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 the NONE 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 CHANGE\_OF\_RELIABILITY for Event Enrollment Fault Condition Precedence (ConfirmedEventNotifications)**

**Purpose:** To verify that the Event Enrollment object's fault detection gives precedence to faults in the Event Enrollment object, then faults in the monitored object, and finally faults detected by the configured Fault algorithm.

**Test Concept:** The test concept corresponds to 8.5.17.7

**Configuration Requirements:** The configuration requirements are identical to those in 8.5.17.7, 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, 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, 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.

## 8.5 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 Integer or 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

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 the NONE fault Algorithm (UnconfirmedEventNotifications)

Purpose: To verify the correct operation of the NONE 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.

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, ?, ?),  
(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, ?, ?),  
 (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

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

#### 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.

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. IF (pMonitoredValue is writable) THEN

WRITE pMonitoredValue = NFV1

ELSE

MAKE (pMonitoredValue = NFV1)

7. 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))

8. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

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

### 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, 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. MAKE (remove the fault condition)
7. 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, ?, ?),  
 (a vendor specified set of values))
8. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

#### 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. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
7. 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))
8. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

#### 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. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
7. 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))

8. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

#### 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. IF (pMonitoredValue is writable) THEN  
    WRITE pMonitoredValue = NV1  
ELSE  
    MAKE (pMonitoredValue = NV1)
7. 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))                 |

8. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

#### 8.5.17.7 CHANGE\_OF\_RELIABILITY for Event Enrollment Fault Condition Precedence (UnconfirmedEventNotifications)

Purpose: To verify that the Event Enrollment object's fault detection gives precedence to faults in the Event Enrollment object, then faults in the monitored object, and finally faults detected by the configured Fault algorithm.

Test Concept: Select an Event Enrollment object, EE1, which monitors an object, O2, which can transition into FAULT. EE1 should be able to be put into a state where it has an internal fault (internal to the Event Enrollment object and unrelated to the Reliability of the monitored object). Starting with both objects in a NORMAL state, causes a condition, which results in a fault in O2. Verify that EE1 reports the fault. Make a condition exist that results in EE1 entering an internal fault. Verify that EE1 reports the new fault condition. Verify that a fault detectable by the fault algorithm does not generate an event. Clear EE1's the internal fault condition and verify that EE1 reports O2's fault. Clear the condition causing O2's fault and verify that EE1 reports fault algorithm event. Clear the condition causing the fault algorithm and verify the return to NORMAL event occurs.

Configuration Requirements: EE1 is configured to detect and report faults and contains a fault algorithm. EE1 and O2 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 pCurrentState = NORMAL
2. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED
3. MAKE (a condition exist which will cause O2 to detect a fault)
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 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' = FALSE,
  - 'From State' = NORMAL,
  - 'To State' = FAULT,
  - 'Event Values' = (MONITORED\_OBJECT\_FAULT,
  - (T, T, ?, ?),
  - O2,
  - (optional, the value of the monitored property),
  - (optional, Reliability of O2),
  - (optional, Status\_Flags of O2),
  - (0 or more other properties of O2))

5. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
6. MAKE (a condition exist which will cause EE1 to transition into internal fault)
7. 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 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' = FALSE,
    - 'From State' = FAULT,
    - 'To State' = FAULT,
    - 'Event Values' = ((R1: any value other than MONITORED\_OBJECT\_FAULT and NO\_FAULT\_DETECTED), (T, T, ?, ?), O2, (optional, the value of the monitored property), (optional, Reliability of O2), (optional, Status\_Flags of O2), (0 or more other properties of O2))
8. VERIFY pCurrentReliability = R1
9. MAKE (a condition that results in a fault detectable by the configured fault algorithm with a reliability value, R2, different from R1)
10. CHECK (that the IUT does not send any notifications)
11. VERIFY pCurrentReliability = R1
12. MAKE (clear the condition that caused EE1 to enter into an internal fault)
11. 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 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' = FALSE,
    - 'From State' = FAULT,
    - 'To State' = FAULT,
    - 'Event Values' = (MONITORED\_OBJECT\_FAULT, (T, T, ?, ?), O2, (optional, the value of the monitored property), (optional, Reliability of O2), (optional, Status\_Flags of O2), (0 or more other properties of O2))
12. VERIFY pCurrentReliability = MONITORED\_OBJECT\_FAULT
13. MAKE (clear the condition that caused O2 to enter into fault)
14. 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 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' = FALSE,  
 'From State' = FAULT,  
 'To State' = FAULT,  
 'Event Values' = (R2,  
 (T, T, ?, ?),  
 O2,  
 (optional, the value of the monitored property),  
 NO\_FAULT\_DETECTED,  
 (optional, Status\_Flags of O2),  
 (0 or more other properties of O2))

15. VERIFY pCurrentReliability = R2

16. MAKE (clear the condition for the fault algorithm)

17. 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 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' = FALSE,  
 'From State' = FAULT,  
 'To State' = NORMAL,  
 'Event Values' = (NO\_FAULT\_DETECTED,  
 (F, F, ?, ?),  
 O2,  
 (the value of the monitored property),  
 NO\_FAULT\_DETECTED,  
 (optional, Status\_Flags of O2),  
 (0 or more other properties of O2))

19. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

20. VERIFY pCurrentState = NORMAL

Notes to Tester: If O2 is located in the IUT, then the IUT shall know and report the property values of O2 in the CHANGE\_OF\_RELIABILITY notifications. If O2 is not located in the IUT, then more time must be allowed between making or clearing a fault condition in O2 and the IUT detecting the change in O2's Reliability (the Notification Fail Time allowance does not start until after the IUT has acquired the information from O2).

#### 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,<br>(T, T, ?, ?),<br>M1,<br>(optional, property value of M1),<br>(optional, M1 Status_Flags, (?, T, ?, ?)),<br>(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,<br>(F, F, ?, ?),<br>M1,<br>(optional, property value of M1),<br>(optional, M1 Status_Flags, (?, F, ?, ?)),<br>(0 or more other properties of M1)) |
9. VERIFY pCurrentReliability = NO\_FAULT\_DETECTED

### 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<br>MONITORED_OBJECT_FAULT<br>and NO_FAULT_DETECTED),<br>(T, T, ?, ?),<br>(M1, any valid monitored object),<br>(optional, property value of M1),<br>(optional, M1 Status_Flags, (?, F, ?, ?)),<br>(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,  
(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

#### 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. 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)
5. VERIFY pCurrentState = OFFNORMAL
5. MAKE (O1 enter a fault state)
6. 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))

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. 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)

#### 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

## 8.6 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.

Dependencies: None.

BACnet Reference Clause: 13.11.

**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.

Dependencies: None.



BACnet Reference Clause: 13.12.

### **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.

Dependencies: None.

BACnet Reference Clause: 13.13.

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.

Dependencies: None.

BACnet Reference Clause: 13.13.

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.

Dependencies: None.

BACnet Reference Clause: 13.14.

#### **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.

Dependencies: None.

BACnet Reference Clause: 13.15.

##### 8.11.1 Confirmed Notifications Subscription

Purpose: To verify that the IUT can initiate a SubscribeCOVProperty service request for confirmed notifications.

Test Steps:

1. RECEIVE SubscribeCOVProperty-Request,  
    'Subscriber Process Identifier' = (any valid process identifier),  
    'Monitored Object Identifier' = (any valid object identifier),  
    'Issue Confirmed Notifications' = TRUE,  
    'Lifetime' = (any non-zero value),  
    'Monitored Property Identifier' = (any valid property identifier),  
    'COV Increment' = (any valid value – optional)
2. 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.

Test Steps:

1. RECEIVE SubscribeCOVProperty-Request,  
    'Subscriber Process Identifier' = (any valid process identifier),  
    'Monitored Object Identifier' = (any valid object identifier),  
    'Issue Confirmed Notifications' = FALSE,  
    'Lifetime' = (any non-zero value),  
    'Monitored Property Identifier' = (any valid property identifier),  
    'COV Increment' = (any valid value – optional)
2. 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.

Test Steps:

1. RECEIVE SubscribeCOVProperty-Request,  
    'Subscriber Process Identifier' = (any valid process identifier),  
    'Monitored Object Identifier' = (any valid object identifier),  
    'Monitored Property Identifier' = (any valid property identifier),  
    'COV Increment' = (any valid value – optional)
2. 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.

Dependencies: None.

BACnet Reference Clause: 14.1.

**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.

Dependencies: None.

BACnet Reference Clause: 14.2.

#### 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.

Dependencies: None.

BACnet Reference Clause: 15.1.

#### 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.

Dependencies: None.

BACnet Reference Clause: 15.2.

#### **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.

Dependencies: None.

BACnet Reference Clause: 15.3.

#### **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.

Dependencies: None.

BACnet Reference Clause: 15.4.

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.

Dependencies: None.

BACnet Reference Clause: 15.5.

### 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),  
     'Priority 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)

### 8.19 ReadPropertyConditional Service Initiation Tests

This clause defines the tests necessary to demonstrate support for initiating ReadPropertyConditional service requests.

Dependencies: None.

BACnet Reference Clause: 15.6.

#### 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),  
     (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),  
     (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.

Dependencies: None.

BACnet Reference Clause: 15.7.

### 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 capable of accessing one or more properties of a single object in the TD via the ReadProperty and ReadPropertyMultiple services. If the IUT cannot be configured in this way, then this test shall be omitted.

Test Steps:

1. MAKE (a condition in the IUT that would normally cause it to send a ReadPropertyMultiple request to the TD to access one or more properties values of a single object)
2. WAIT (a time interval specified by the vendor as sufficient for the IUT to determine that the TD does not support the ReadPropertyMultiple service)
3. REPEAT X = (the properties that the IUT is to read) DO {

```

RECEIVE ReadProperty-Request,
 'Object Identifier' = (object identifier referenced by X),
 'Property Identifier' = (property identifier referenced by X)
TRANSMIT ReadProperty-Ack,
 'Object Identifier' = (object identifier referenced by X),
 'Property Identifier' = (property identifier referenced by X),
 'Property Value' = (any valid value)
}

```

#### 8.20.5.2 Fallback to ReadProperty on Reject - UNRECOGNIZED\_SERVICE Response

BACnet Reference Clauses: 15.5 and 15.7

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.

Test Steps:

1. RECEIVE ReadPropertyMultiple-Request,
  - 'Object Identifier' = (object identifier of the specified object),
  - 'List of Property References' = (one or more properties of the specified object)
2. TRANSMIT BACnet-Reject-PDU,
  - 'Reject Reason' = UNRECOGNIZED\_SERVICE
3. REPEAT X = (the properties from Step 1) 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.

Dependencies: None.

BACnet Reference Clause: 15.8.

##### 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' = (any Trend Log object),  
'Property Identifier' = Log\_Buffer

#### 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' = (any Trend Log object),  
'Property Identifier' = Log\_Buffer,  
'Reference Index' = (any Unsigned value),  
'Count' = (any INTEGER value)

#### 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 Simple-Ack-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 Simple-Ack-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 D1)
      - 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 Containing a Specific Datatype**

Purpose: To verify that the IUT can initiate one or more ReadRange requests that access and present a tester-specified portion of log records having a specific datatype, using any valid range. It is a generic test used to test data presentation requirements.

Test Concept: Run test in Clause 8.21.8 and verify that the data presentation meets the criteria specified by the BIBB being tested.

Note to Tester: The values presented by the IUT may differ from the values transmitted on the wire due to rounding, truncation, formatting, language, conversion, etc.

Note to Tester: The IUT is not required to display records containing log-status values.

### **8.22 WriteProperty Service Initiation Tests**

This clause defines the tests necessary to demonstrate support for initiating WriteProperty service requests.

Dependencies: None.

BACnet Reference Clause: 15.9.

#### **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.

#### 8.23 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.

Dependencies: None.

BACnet Reference Clause: 15.10.

##### 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.



Dependencies: None.

BACnet Reference Clause: 16.1.

#### 8.24.1 Indefinite Duration, Disable, No Password

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for an indefinite time duration and do not convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Enable/Disable' = DISABLE,

#### 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 at least 5 characters)

#### 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 at least 5 characters)

#### 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 at least 5 characters)

#### 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,

**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

**8.24.7 Time Duration, Disable-Initiation, Password**

Purpose: To verify that the IUT can initiate DeviceCommunicationControl service requests that indicate communication should cease for a specific time duration and that convey a password.

Test Steps:

1. RECEIVE DeviceCommunicationControl-Request,  
'Time Duration' = (any unsigned value in the range from 1 to 65535),  
'Enable/Disable' = DISABLE  
'Password' = (a password of up to 20 characters)

**8.25 ConfirmedPrivateTransfer Service Initiation Test**

Dependencies: None.

BACnet Reference Clause: 16.2.

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**

Dependencies: None.

BACnet Reference Clause: 16.3.

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.

Dependencies: None.

BACnet Reference Clause: 16.4.

#### 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 at least 5 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 at least 5 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.

Dependencies: None.

BACnet Reference Clause: 16.5.

**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.

Dependencies: None.

BACnet Reference Clause: 16.6.

#### 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

Dependencies: None.

BACnet Reference Clause: 16.7.

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**

Dependencies: None.

BACnet Reference Clause: 16.8.

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.

Dependencies: None.

BACnet Reference Clause: 16.9.

**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.

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Object Identifier' = (any object identifier)

**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.

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Object Name' = (any CharacterString)

**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.

Test Steps:

1. RECEIVE  
DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Device Instance Range Low Limit' = (any integer X:  $1 \leq X \leq$  'Device Instance Range High Limit'),

'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' ≤ Y ≤ 4,194,303),  
'Object Identifier' = (any object identifier)

#### 8.32.4 Object Name Selection with a Device Instance Range

Purpose: To verify that the IUT can initiate Who-Has service requests using the object name form with a device instance range. If the IUT cannot be caused to issue a Who-Has request of this form, then this test shall be omitted.

Test Steps:

##### 1. RECEIVE

DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
SOURCE = IUT,  
Who-Has-Request,  
'Device Instance Range Low Limit' = (any integer X: 1 ≤ X ≤ 'Device Instance Range High Limit'),  
'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' ≤ Y ≤ 4,194,303),  
'Object Name' = (any CharacterString)

#### 8.33 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.

Dependencies: None.

BACnet Reference Clause: 16.10.

##### 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: 1 ≤ X ≤ 'Device Instance Range High Limit'),  
'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' ≤ Y ≤ 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.

Dependencies: ReadProperty Service Execution tests, 9.18.

BACnet Reference Clauses: 17.2 and 12.11.

#### 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<sub>IUT</sub>)
2. TRANSMIT VT-Open-ACK,  
     'Remote VT Session Identifier' = (any valid session identifier, S<sub>TD</sub>)
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.

Dependencies: ReadProperty Service Execution tests, 9.18.

BACnet Reference Clauses: 17.3 and 12.11.

#### 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,  $S_{TD}$ , 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,  $S_{TD,i}$  and  $S_{TD,j}$ , 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.

Dependencies: VT-Open Service Initiation Tests, 8.36; VT-Open Service Execution Tests, 9.34; VT-Close Service Execution, 9.35.

BACnet Reference Clause 17.4.

#### 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

Dependencies: None.

BACnet Reference Clause: 24.4.

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

Dependencies: Initial Test, 8.39.1.

BACnet Reference Clause: 24.1.4.

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

Dependencies: None.

BACnet Reference Clauses: 24.2.2 and 24.5.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 24.2.3 and 24.5.

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.

Dependencies: None.

BACnet Reference Clause: 24.2.1(a, b), 24.2.4.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 24.2.4 and 24.5.

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

Dependencies: None.

BACnet Reference Clause: 24.2.5.

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

Dependencies: ReadProperty Service Initiation Tests, 8.18.

BACnet Reference Clauses: 24.3.1, 24.3.2, and 24.5.

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)

## 9. APPLICATION SERVICE EXECUTION TESTS

The test cases defined in this clause shall be used to verify that a BACnet device correctly implements the service procedure for the specified application service. BACnet devices shall be tested for the proper execution of each application service for which the PICS indicates execution is supported.

For each application service included in this clause several test cases are defined that collectively test the various options and features defined for the service in the BACnet standard. A test case is a sequence of one or more messages that are exchanged between the implementation under test (IUT) and the testing device (TD) in order to determine if a particular option or feature is correctly implemented. Multiple test cases that have a similar or related purpose are collected into test groups.

Under some circumstances an IUT may be unable to demonstrate conformance to a particular test case because the test applies to a feature that requires a particular BACnet object or optional property that is not supported in the IUT. For example, a device may support the File Access services but restrict files to stream access only. Such a device would have no way to demonstrate that it could implement the record access features of the File Access services. When this type of situation occurs the IUT shall be considered to be in conformance with BACnet provided the PICS documentation clearly indicates the restriction. Failure to document the restriction shall constitute nonconformance to the BACnet standard. All features and optional parameters for BACnet application services shall be supported unless a conflict arises because of unsupported objects or unsupported optional properties.

For each application service the tests are divided into two types, positive tests and negative tests. The positive tests verify that the IUT can correctly handle cases where the service is expected to be successfully completed. The negative tests verify correct handling for various error cases that may occur. Negative tests include inappropriate service parameters but they do not include cases with encoding errors or otherwise malformed PDUs. Tests to ensure that the IUT can handle malformed PDUs are defined in 13.4.

Many test cases allow flexibility in the value to be used in a service parameter. The tester is free to choose any value within the constraints defined in the test case. The IUT shall be able to respond correctly to any valid selection the tester might make. The EPICS is considered to be a definitive reference indicating the BACnet functionality supported and the configuration of the object database. Any discrepancies between the BACnet functionality or the value of properties in the object database as defined in the EPICS, and the values returned in messages defined for a test case constitutes a failure of the test. For example, if a test step involved reading a property of an object in the database the returned value must match the value provided in the EPICS.

### 9.1 AcknowledgeAlarm Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing AcknowledgeAlarm service requests.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 13.5.

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 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 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:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (pTimeDelay)
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' =                   | NORMAL,                                             |
| 'To State' =                     | (any appropriate non-normal event state),           |
| 'Event Values' =                 | (the values appropriate to the event type)          |
4. TRANSMIT BACnet-SimpleACK-PDU
5. RECEIVE
  - DESTINATION = (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: 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' =                   | NORMAL,                                             |
| 'To State' =                     | (any appropriate non-normal event state),           |
| 'Event Values' =                 | (the values appropriate to the event type)          |
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE, TRUE, TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
 

|                                      |                                                                              |
|--------------------------------------|------------------------------------------------------------------------------|
| 'Acknowledging Process Identifier' = | (the value of the 'Process Identifier' parameter in the event notification), |
| 'Event Object Identifier' =          | (the 'Event Object Identifier' from the event notification),                 |
| 'Event State Acknowledged' =         | (the state specified in the 'To State' parameter of the notification),       |
| 'Time Stamp' =                       | (the time stamp conveyed in the notification),                               |
| 'Acknowledgement Source' =           | (any valid value),                                                           |
| '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  $\geq$  1) THEN

## BEFORE Notification Fail Time

RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (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  $\geq$  1) THEN

## BEFORE Notification Fail Time

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' = (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 = (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' = (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)

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.

#### 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.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (pTimeDelay)
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' = NORMAL,  
'To State' = (any appropriate non-normal event state),  
'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN  
RECEIVE  
DESTINATION = (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' = NORMAL,  
'To State' = (any appropriate non-normal event state),  
'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)



6. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (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  $\geq$  1) THEN  
 BEFORE **Notification Fail Time**  
 RECEIVE  
 DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (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' = (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
9. IF (the notification in step 8 was not broadcast) THEN  
 IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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)

Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 9 shall be the same address used in step 4. Inclusion of the 'To State' parameter in acknowledgement notifications was added in protocol version 1, protocol revision 1. Implementations that precede this version will not include this parameter. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 9.

#### 9.1.1.5 Successful Alarm Acknowledgment of Unconfirmed Event Notifications Using the Sequence Number Form of the 'Time of Acknowledgment' Parameter

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  $\geq$  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  $\geq$  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.

Test Steps:

1. MAKE (a change that triggers the detection of an event in the IUT)
2. WAIT (pTimeDelay)
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' = NORMAL,  
 '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' = NORMAL,  
 'To State' = (any appropriate offnormal event state),  
 'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE, TRUE, TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,  
 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (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' = (T1: 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' = (T1),  
 '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' = (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)
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' = (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)
20. TRANSMIT BACnet-SimpleACK-PDU
21. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (TRUE, TRUE, TRUE)

Notes to Tester: The destination address used for the acknowledgment notification in steps 12 and 19 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 12, 13, 19, and 20.

#### 9.1.1.11 Successful Alarm Re-Acknowledgment of Unconfirmed Event Notifications

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  $\geq$  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.

Test Steps:

1. MAKE (a change that triggers the detection of an offnormal event in the IUT)
2. WAIT (pTimeDelay)
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' = NORMAL,
    - 'To State' = (any appropriate offnormal event state),
    - 'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE, TRUE, TRUE)
6. TRANSMIT AcknowledgeAlarm-Request,
  - 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
  - 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
  - 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
  - 'Time Stamp' = (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)

Notes to Tester: The destination address used for the acknowledgment notification in steps 9 and 14 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4, 9, and 14.

#### 9.1.1.12 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.

Test Steps:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT (pTimeDelay)
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' = NORMAL,
    - 'To State' = (S1: any appropriate non-normal 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' = NORMAL,
    - 'To State' = (S2: any appropriate non-normal event state),
    - 'Event Values' = (the values appropriate to the event type)
6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)
8. TRANSMIT AcknowledgeAlarm-Request,
  - 'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),
  - 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),
  - 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),
  - 'Time Stamp' = (any valid time stamp),
  - '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 = (FALSE,TRUE,TRUE)
11. TRANSMIT AcknowledgeAlarm-Request,

'Acknowledging Process Identifier' = (the process identifier configured for this event),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 'Time of Acknowledgment' = (the current time using a Time format)

## 12. RECEIVE BACnet-SimpleACK-PDU

13. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  1) THENBEFORE **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  $\geq$  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)

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 5, 6, 15, and 16.

#### 9.1.2.2 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the 'Acknowledging Process Identifier' is Invalid

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 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:**

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. WAIT pTimeDelay
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' = NORMAL,
    - 'To State' = (S1: any appropriate non-normal event state),
    - 'Event Values' = (the values appropriate to the event type)
4. IF (the notification in step 3 was not a broadcast) THEN
  - RECEIVE
    - DESTINATION = (at least one device other than the TD),
    - SOURCE = IUT,
    - UnconfirmedEventNotification-Request,

- 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (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' = NORMAL,  
 'To State' = (S2: any appropriate non-normal event state),  
 'Event Values' = (the values appropriate to the event type)
5. VERIFY (the 'Event Object Identifier' from the event notification), Acked\_Transitions = (FALSE,TRUE,TRUE)
  6. TRANSMIT AcknowledgeAlarm-Request,
 

'Acknowledging Process Identifier' = (the value of the 'Process Identifier' parameter in the event notification),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (a time stamp older than the one conveyed in the notification),  
 '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 = (FALSE,TRUE,TRUE)
  9. TRANSMIT AcknowledgeAlarm-Request,
 

'Acknowledging Process Identifier' = (the process identifier configured for this event),  
 'Event Object Identifier' = (the 'Event Object Identifier' from the event notification),  
 'Event State Acknowledged' = (the state specified in the 'To State' parameter of the notification),  
 'Time Stamp' = (the time stamp conveyed in the notification),  
 '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  $\geq$  1) THEN
 

**BEFORE Notification Fail Time**

RECEIVE

DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST | TD,  
 SOURCE = IUT,  
 UnconfirmedEventNotification-Request,  
 'Process Identifier' = (the process identifier configured for this event),  
 'Initiating Device Identifier' = IUT,  
 'Event Object Identifier' = (the object detecting the alarm),  
 'Time Stamp' = (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)

```

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3. The destination address used for the acknowledgment notification in step 12 shall be the same address used in step 4. When multiple event notifications are expected for a specific event, the order that the IUT transmits them in is irrelevant. If the IUT can only be configured with one recipient in the Recipient\_List property of the issuing Notification\_class object, omit steps 4 and 12.

#### 9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the '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 Acks\_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 Acks\_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.

**Dependencies:** SubscribeCOV Service Initiation Tests 8.10.

**BACnet Reference Clause:** 13.6.

### 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 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 and Status\_Flags 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)

#### 9.2.2 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.

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. WAIT (a value two times Lifetime)
- 4. TRANSMIT ConfirmedCOVNotification-Request,
  - 'Subscriber Process Identifier' = (the process identifier used in step 2),
  - '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)

5. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  10) THEN  
 RECEIVE  
     BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (UNKNOWN\_SUBSCRIPTION) |  
     (BACnet-SimpleACK-PDU)  
 ELSE  
     RECEIVE BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (any valid error code for class SERVICES) |  
     (BACnet-SimpleACK-PDU)

### 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.

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' = (a process identifier different from the one used in step 2),  
     '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  $\geq$  10) THEN  
 RECEIVE  
     BACnet-Error-PDU,  
         Error Class = SERVICES,  
         Error Code = (UNKNOWN\_SUBSCRIPTION) |  
     (BACnet-SimpleACK-PDU)  
 ELSE  
     RECEIVE  
         BACnet-Error-PDU,  
             Error Class = SERVICES,  
             Error Code = (any valid error code for class SERVICES) |  
         (BACnet-SimpleACK-PDU)

### 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  $\geq$  10) THEN
 

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

#### 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.

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 used in step 2),             |
| 'Initiating Device Identifier' =  | TD,                                                  |
| 'Monitored Object Identifier' =   | (any object Y supporting COV notification except X), |
| '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  $\geq$  10) THEN
 

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

#### 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.

Test Configuration: 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> ,<br>Time_Of_Device_Restart = (any valid time),<br>Last_Restart_Reason = LRR <sub>1</sub> ) |
2. CHECK(that the IUT processes the notification as described by the vendor)

#### 9.3.2 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 Steps:

1. RECEIVE SubscribeCOV,
 

|                                   |                                 |
|-----------------------------------|---------------------------------|
| 'Subscriber Process Identifier' = | (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' = | (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 and Status_Flags 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)

#### 9.3.3 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)

### 9.3.4 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)

### 9.3.5 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)



#### 9.4 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 acknowledgement 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 ConfirmedEventNotification Without a Notification Class Parameter

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 type 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)

## 9.5 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.X1 except that the UnconfirmedEventNotification requests are used instead of ConfirmedEventNotification requests and the IUT does not acknowledge receiving the notifications.

## 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.

Dependencies: None.

BACnet Reference Clause: 13.11.

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.

Dependencies: None.

BACnet Reference Clause: 13.12.

### 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

### 9.9 LifeSafetyOperation Service Execution Test

This clause defines the tests necessary to demonstrate support for executing LifeSafetyOperation service requests.

#### 9.9.1 Reset Single Object Execution Tests

Dependencies: None.

BACnet Reference Clause: 13.13.

Purpose: To verify that the IUT can correctly execute a LifeSafetyOperation service request to a single Life Safety Object.

Test Concept: A Life Safety object is toggled between a NORMAL and OFFNORMAL state. The STATUS should latch in the OFFNORMAL state until the LifeSafetyOperation service request is transmitted. This test may be omitted when the device does not support latching.

Configuration Requirements: The IUT must support a Life Safety Object that supports alarming. This object is configured into the NORMAL state, driven to the OFFNORMAL state and returned to the NORMAL state by adjusting its Present\_Value. The STATUS should remain latched in the OFFNORMAL state until the reset is issued. This test must be repeated for each reset function supported.

Test Steps:

1. REPEAT X = (All supported enumerations that reset the object) DO {
2.     MAKE (the selected object enter a latched state where enumeration X will reset the object)
3.     MAKE (Event-State = NORMAL)
4.     IF (enumeration value creates an Event\_State = FAULT) THEN  
       CHECK (Event\_State = FAULT)  
   ELSE  
       CHECK (Event\_State = OFFNORMAL)
5.     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)
6.     RECEIVE BACnet-SimpleACK-PDU
7.     VERIFY (Object), STATUS = NORMAL)
- }

#### 9.9.2 Reset Multiple Object Execution Tests

Dependencies: None.

BACnet Reference Clause: 13.13.

Purpose: To verify that the IUT can correctly execute a LifeSafetyOperation service request to multiple Life Safety objects.

Test Concept: Two Life Safety objects are toggled between a NORMAL and an OFFNORMAL state. The STATUS should latch in the OFFNORMAL state until the LifeSafetyOperation service request is transmitted. This test may be omitted when the device does not support latching.

Configuration Requirements: The IUT must support a Life Safety Object that supports alarming and have at least two objects that can be latched. This object is configured into the NORMAL state, driven to the OFFNORMAL state and returned to the NORMAL state by adjusting its Present\_Value. The STATUS should remain latched in the OFFNORMAL state until the reset is issued. This test must be repeated for each reset function supported.

Test Steps:

1. REPEAT X = (All supported enumerations that reset the object) DO {

2. MAKE (the selected objects enter a latched state where enumeration X will reset the objects)
3. MAKE (Event\_State = NORMAL)
4. IF (enumeration value creates an Event\_State = FAULT) THEN  
     CHECK (Event\_State = FAULT)  
     ELSE  
         CHECK (Event\_State = OFFNORMAL)
5. TRANSMIT LifeSafetyOperation-Request,  
     'Requesting Process Identifier' = (any valid identifier),  
     'Requesting Source' = (any valid character string),  
     'Request' = (any valid LifeSafetyOperation request)
6. RECEIVE BACnet-SimpleACK-PDU
7. VERIFY (Object), STATUS = NORMAL)

### 9.9.3 Silencing Execution Test

Dependencies: None.

BACnet Reference Clause: 13.13.

Purpose: To verify that the IUT can correctly execute a LifeSafetyOperation service request to silence an alarming device.

Test Concept: An audible device and/or visual device is attached to the IUT and is sounding/flashing. A LifeSafetyOperation service request is transmitted to silence the sounder/strobe.

Configuration Requirements: The IUT must be fitted with needed audible and visual equipment.

Test Steps:

- 1 REPEAT X = (All supported enumerations that silence the object) DO {
2. MAKE (the selected object enter a state where enumeration X will commence alerts)
3. MAKE (Event\_State = NORMAL)
4. 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)
5. RECEIVE BACnet-SimpleACK-PDU
6. CHECK (Sounder/Strobe inactive)

### 9.10 SubscribeCOV Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing SubscribeCOV service requests.

Dependencies: None.

BACnet Reference Clause: 13.14.

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) |
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

Dependencies: Indefinite lifetime COV subscriptions, 9.10.1.1.

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 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' = (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' = (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 (the lifetime of the subscription)  
 7. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)  
 8. 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' = (any valid process identifier),  
     'Monitored Object Identifier' = (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' = (the same identifier used in the subscription),  
             'Initiating Device Identifier' = IUT,  
             'Monitored Object Identifier' = (the same object used in the subscription),  
             'Time Remaining' = 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' = (the same identifier used in the subscription),  
                 'Initiating Device Identifier' = IUT,  
                 'Monitored Object Identifier' = (the same object used in the subscription),  
                 'Time Remaining' = 60,

- 'List of Values' = (values appropriate to the object type of the monitored object)
4. 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 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' = (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
      - 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' = (the requested subscription lifetime),
          - '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

Dependencies: TimeSynchronization Service Execution Tests, 9.30; UTC-TimeSynchronization Service Execution Tests, 9.31.

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  $D_1$ , 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.

Test Configuration: 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)

#### 9.10.2 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.

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)

### 9.11 SubscribeCOVProperty Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing SubscribeCOVProperty service requests.

Dependencies: None.

BACnet Reference Clause: 13.15.

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. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.11.1.2.



Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     IF (the IUT supports confirmed notifications) THEN  
         RECEIVE BACnetConfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same identifier used in the subscription),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),  
         'Time Remaining' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
         'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     ELSE  
         RECEIVE BACnet-Error PDU,  
         Error Class = SERVICES,  
         Error Code = SERVICE\_REQUEST\_DENIED | OTHER

#### 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. An implementation that supports COV reporting cannot respond with an error for both this test and the test in 9.11.1.1.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,  
     'Subscriber Process Identifier' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE,  
     'Lifetime' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     IF (the IUT supports confirmed notifications) THEN  
         RECEIVE BACnetUnconfirmedCOVNotification-Request,  
         'Subscriber Process Identifier' = (the same identifier used in the subscription),  
         'Initiating Device Identifier' = IUT,  
         'Monitored Object Identifier' = (the same object used in the subscription),  
         'Time Remaining' = (any value > 0 if automatic cancellation is supported, otherwise 0),  
         'List of Values' = (values appropriate to the property subscribed to, and any other properties the IUT provides with it, such as Status-Flags)  
     ELSE  
         RECEIVE BACnet-Error PDU,  
         Error Class = SERVICES,  
         Error Code = SERVICE\_REQUEST\_DENIED | OTHER

#### 9.11.1.3 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

Dependencies: Indefinite lifetime COV subscriptions, 9.11.1.1.

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.

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)

Notes to Tester: The IUT shall not transmit a COV notification message.

#### 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)

Notes to Tester: The IUT shall not transmit a COV notification message. An error message is not an acceptable response.

#### 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' = (any valid process identifier),  
     'Monitored Object Identifier' = (any object supporting COV notifications),  
     'Issue Confirmed Notifications' = TRUE | FALSE,  
     'Lifetime' = (a value between 60 seconds and 300 seconds),  
     'Monitored Property Identifier' = (any valid property supporting COV notifications)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**  
     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 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

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 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)

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' = (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

RECEIVE BACnetUnconfirmedCOVNotification-Request,

'Subscriber Process Identifier' = (the same identifier used in the subscription),  
 'Initiating Device Identifier' = IUT,  
 'Monitored Object Identifier' = (the same object used in the subscription),  
 'Time Remaining' = (a value greater than 0 and less than the requested subscription lifetime),  
 'List of Values' = (values appropriate to the object type of the monitored object including the changed value that triggered the notification)

6. WAIT (the lifetime of the subscription)

7. MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)

Notes to Tester: The IUT shall not transmit a COV notification message addressed to the TD after step 6.

### 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' = (any valid process identifier),  
 'Monitored Object Identifier' = (any object supporting COV notifications),  
 'Issue Confirmed Notifications' = TRUE | FALSE,

```

 'Lifetime' = 60,
 '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' = 60,
 'List of Values' = (values appropriate to the object type of the monitored object including
 the changed value of 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' = 60,
 'List of Values' = (values appropriate to the object type of the monitored object
 including the changed value of that triggered the notification)

4. TRANSMIT SubscribeCOVProperty-Request,
 'Subscriber Process Identifier' = (any valid process identifier),
 'Monitored Object Identifier' = (any object supporting COV notifications),
 'Issue Confirmed Notifications' = TRUE | FALSE,
 'Lifetime' = (a value between 180 and 300 seconds),
 'Monitored Property Identifier' = (any valid property supporting COV notifications)
5. RECEIVE BACnet-SimpleACK-PDU
6. 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 subscription lifetime),
 'List of Values' = (values appropriate to the object type of the monitored object
 including the changed value of that triggered the notification)

 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 subscription lifetime),
 'List of Values' = (values appropriate to the object type of the monitored object
 including the changed value of that triggered the notification)

```

#### 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 datatype REAL. The subscription request specifies a COV increment. The monitored property is changed by an amount less than the increment, and the TD waits to ensure that the IUT does not generate a notification. The monitored property is changed by an amount slightly more than is required to cause a COV notification, and the TD waits for the notification.

Test Configuration: If the property being subscribed to has a related COV\_Increment property in the object, then the value of the COV\_Increment property should be significantly different than the COV increment provided in the subscription service.

Test Steps:

1. TRANSMIT SubscribeCOVProperty-Request,
  - 'Subscriber Process Identifier' = (any valid process identifier),
  - 'Monitored Object Identifier' = (any object supporting COV notifications),
  - 'Issue Confirmed Notifications' = TRUE | FALSE,
  - 'Lifetime' = (any value that will ensure no re-subscription is required to complete the test),
  - 'Monitored Property Identifier' = (any valid property supporting COV notifications),
  - 'COV Increment' = (any valid increment value)
2. RECEIVE BACnet-SimpleACK-PDU
3. BEFORE **Notification Fail Time**
  - IF (the subscription was for confirmed notifications) THEN
    - RECEIVE BACnetConfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = (the same object used in the subscription),
      - 'Time Remaining' = (the requested lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
    - TRANSMIT BACnet-SimpleACK-PDU
  - ELSE
    - RECEIVE BACnetUnconfirmedCOVNotification-Request,
      - 'Subscriber Process Identifier' = (the same identifier used in the subscription),
      - 'Initiating Device Identifier' = IUT,
      - 'Monitored Object Identifier' = (the same object used in the subscription),
      - 'Time Remaining' = (the requested lifetime),
      - 'List of Values' = (values appropriate to the object type of the monitored object including the value of monitored property)
4. MAKE (the monitored property change by less than the COV increment)
5. CHECK (verify that the IUT did not transmit a notification message for the monitored property)
6. MAKE (the monitored property change by slightly more than COV Increment less the amount changed in step 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)



### 9.11.2 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 )  
     ELSE  
         RECEIVE BACnet-Error PDU,  
             Error Class = OBJECT  
             Error Code = OPTIONAL\_FUNCTIONALITY\_NOT\_SUPPORTED

#### 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 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

### 9.12 AtomicReadFile Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing AtomicReadFile service requests.



Dependencies: None.

BACnet Reference Clause: 14.1.

**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 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' = S,  
     'File Start Position' = 0,  
     'Requested Octet Count' = (the number of octets in the test file)
2. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Position' = 0,  
     'File Data' = (the known contents of the test file)

##### 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:  $n \geq$  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.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

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  $n: n \geq$  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  $\geq 0$ ),  
'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.

Dependencies: AtomicReadFile Service Execution Tests 9.12; ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 14.2.



**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  $\geq$  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 < 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.

Dependencies: WriteProperty Service Execution Tests 9.22.

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:  $0 < n < \text{InitialNumRecords}$ )
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.

Dependencies: WriteProperty Service Execution Tests 9.22.

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 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 octets as the initial data for the file.

Test Steps:

1. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = 0,
  - 'Requested Octet Count' = (any number  $\geq$  the number of octets in the test data)
2. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Position' = 0,
  - 'File Record Data' = (the initial data)
3. TRANSMIT AtomicWriteFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = 0,
  - 'File Data' = (the test data)
4. RECEIVE AtomicWriteFile-ACK,
  - 'File Start Position' = 0
5. TRANSMIT AtomicReadFile-Request,
  - 'File Identifier' = S,
  - 'File Start Position' = 0,
  - 'Requested Octet Count' = (any number  $>$  the number of octets in the test data)
6. RECEIVE AtomicReadFile-ACK,
  - 'End of File' = TRUE,
  - 'File Start Position' = 0,
  - 'File Data' = (the test data)
7. VERIFY (R), Modification\_Date = (the current date and time)
8. VERIFY (R), ARCHIVE = FALSE
9. VERIFY (R), Number\_Of\_Records = (the number of records in the test data)

#### 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. 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 initial 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 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 AtomicWriteFile-Request,  
     'File Identifier' = S,  
     'File Start Position' = -1,  
     'File Data' = (the test data)
6. RECEIVE AtomicWriteFile-ACK,  
     'File Start Position' = InitialNumOctets,
7. TRANSMIT AtomicReadFile-Request,  
     'File Identifier' = S,  
     'File Start Position' = InitialNumOctets,  
     'Requested Octet Count' = (the number of octets in the test data)
8. RECEIVE AtomicReadFile-ACK,  
     'End of File' = TRUE,  
     'File Start Position' = InitialNumOctets,  
     'File Data' = (the test data)
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 + 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.

Dependencies: WriteProperty Service Execution Tests 9.22.

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:  $0 < n < \text{InitialNumOctets}$ )
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.

Dependencies: WriteProperty Service Execution Tests 9.22.

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:

1. TRANSMIT AtomicWriteFile-Request,  
     'File Identifier' = R,  
     'File Start Position' = 0,  
     'File Data' = (any stream file data)
2. 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:

1. 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)
2. 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:  $n < -1$  |  $n >$  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.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause 15.1.

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.

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

#### 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.

Test Steps:

1. READ InitialList = (L), ListProp2. 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  
     ELSE  
         RECEIVE AddListElement-Error,  
             Error Class = SERVICES,  
             Error Code = INVALID\_PARAMETER\_DATATYPE  
             'First Failed Element' = 2
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.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause 15.2.

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  $\geq$  7) THEN
 

|                          |                   |
|--------------------------|-------------------|
| RemoveListElement-Error, |                   |
| Error Class =            | PROPERTY,         |
| Error Code =             | INVALID_DATA_TYPE |
| 'First Failed Element' = | 2                 |

 ELSE
 

|                                  |                      |
|----------------------------------|----------------------|
| RECEIVE RemoveListElement-Error, |                      |
| Error Class =                    | SERVICES   PROPERTY, |
| Error Code =                     | OTHER                |
| 'First Failed Element' =         | 2                    |
4. VERIFY (L), ListProp = InitialList

### 9.16 CreateObject Service Execution Tests

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 15.3.

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 )
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 )  
'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 Identifier' = (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 Type' = (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 Identifier' = (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 Steps:

1. READ X1 = Object\_List
2. TRANSMIT CreateObject-Request,  
     'Object Type' = (any creatable object type),  
     'List Of Initial Values' = (a list of one or more properties and their initial values, that the IUT will accept initial values for, with one of the values being out of range)
3. IF (Protocol\_Revision is present AND Protocol\_Revision ≥ 4) THEN  
     RECEIVE CreateObject-Error PDU,  
         Error Class = PROPERTY,  
         Error Code = VALUE\_OUT\_OF\_RANGE  
         'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
     ELSE  
         RECEIVE CreateObject-Error,  
             Error Class = PROPERTY,  
             Error Code = VALUE\_OUT\_OF\_RANGE | OTHER  
             'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
4. CHECK (Verify that the new object was not created)
5. TRANSMIT CreateObject-Request,  
     'Object Type' = (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)  
     ELSE  
         RECEIVE CreateObject-Error,  
             Error Class = PROPERTY,  
             Error Code = VALUE\_OUT\_OF\_RANGE | INVALID\_DATATYPE | OTHER  
             'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
             | (BACnet-Reject-PDU  
                 Reject Reason = INVALID\_PARAMETER\_DATATYPE | INVALID\_TAG)
7. READ X2 = Object\_List
8. CHECK (X1 = X2)

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

Purpose: To verify the correct execution of the CreateObject service request when an object identifier is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Identifier' = (any unique object identifier of a type that is creatable and an instance

- 'List Of Initial Values' = number that is creatable),  
(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 PDU,  
Error Class = PROPERTY,  
Error Code = VALUE\_OUT\_OF\_RANGE  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)  
ELSE  
RECEIVE CreateObject-Error,  
Error Class = PROPERTY,  
Error Code = VALUE\_OUT\_OF\_RANGE | OTHER  
'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
  3. CHECK (Verify that the new object was not created)
  4. TRANSMIT CreateObject-Request,  
'Object Identifier' = (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)  
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  $\geq$  4) THEN  
RECEIVE BACnet-Error PDU,  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT  
ELSE  
RECEIVE BACnet-Error PDU  
Error Class = OBJECT,  
Error Code = UNKNOWN\_OBJECT | NO\_OBJECTS\_OF\_SPECIFIED\_TYPE | OTHER

#### 9.16.2.6 Attempting to Create an Object with an instance of 4194303

Purpose: This test case verifies the correct execution of the CreateObject service request when the 'Object Specifier' parameter conveys an object identifier with an instance of 4194303. This test shall be performed only if the Protocol\_Revision property is present in the Device object and has a value greater than or equal to 4.

Test Steps:

1. TRANSMIT CreateObject-Request,  
'Object Identifier' = (any object identifier representing a creatable object-type with an instance of 4194303)
2. RECEIVE BACnet-Reject-PDU,

'Reject Reason' = PARAMETER\_OUT\_OF\_RANGE

### 9.17 DeleteObject Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing DeleteObject service requests.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 15.4.

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. TRANSMIT DeleteObject-Request,  
'Object Identifier' = X
2. RECEIVE BACnet-Error-PDU,  
Error Class = OBJECT,  
Error Code = OBJECT\_DELETION\_NOT\_PERMITTED
3. VERIFY (X), Object\_Name = (the Object\_Name specified in the EPICS)
4. 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.

Dependencies: None.

BACnet Reference Clause: 15.5.

#### 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. VERIFY (Device, X),  
    Object\_List = (the first element of the Object\_List array as specified in the EPICS),  
    ARRAY INDEX = 1

##### 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.

BACnet Reference Clause: 5.4.5.3

Test Steps:

1. VERIFY (Device, X), Object\_List = (the entire Object\_List array)

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

#### 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.

Dependencies: None.

BACnet Reference Clause: 15.6.

### 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.

Dependencies: None.

BACnet Reference Clause: 15.7.

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' = (the value of P1 specified in the EPICS)

### 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' = (the value of P1 specified in the EPICS),  
'Property Identifier' = P2,  
'Property Value' = (the value of P2 specified in the EPICS)  
-- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

### 9.20.1.3 Reading a Single Property from Multiple Objects

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' = (the value of P1 specified in the EPICS),  
'Object Identifier' = Object2,  
'Property Identifier' = P2,  
'Property Value' = (the value of P2 specified in the EPICS)  
-- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

### 9.20.1.4 Reading Multiple Properties from Multiple Objects

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' = (the value of P1 specified in the EPICS),  
 'Property Identifier' = P2,  
 'Property Value' = (the value of P2 specified in the EPICS),  
 'Property Identifier' = P3,  
 'Property Value' = (the value of P3 specified in the EPICS),  
 'Object Identifier' = Object2,  
 'Property Identifier' = P4,  
 'Property Value' = (the value of P4 specified in the EPICS),  
 'Property Identifier' = P5,  
 'Property Value' = (the value of P5 specified in the EPICS),  
 'Property Identifier' = P6,  
 'Property Value' = (the value of P6 specified in the EPICS)

-- ... (An appropriate value must be returned for each property included in the ReadPropertyMultiple-Request.)

**9.20.1.5 Reading Multiple Properties with a Single Embedded Access Error**

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' = (the value of P1 specified in the EPICS),  
 'Property Identifier' = P2,  
 'Property Value' = (the value of P2 specified in the EPICS),  
 'Property Identifier' = P3,  
 'Error Class' = PROPERTY,  
 'Error Code' = UNKNOWN\_PROPERTY,  
 'Property Identifier' = P4,  
 'Property Value' = (the value of P4 specified in the EPICS)

**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' = (the value of P1 specified in the EPICS),
- 'Property Identifier' = P2,
- 'Property Value' = (the value of P2 specified in the EPICS),
- 'Property Identifier' = P3,
- 'Error Class' = PROPERTY,
- 'Error Code' = UNKNOWN\_PROPERTY,
- 'Property Identifier' = P4,
- 'Error Class' = PROPERTY,
- 'Error Code' = UNKNOWN\_PROPERTY,
- 'Object Identifier' = Object2,
- 'Property Identifier' = P5,
- 'Error Class' = OBJECT,
- 'Error Code' = UNKNOWN\_OBJECT,
- 'Property Identifier' = P6,
- 'Error Class' = OBJECT,
- 'Error Code' = UNKNOWN\_OBJECT

#### 9.20.1.7 Reading ALL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier ALL. One instance of each object-type supported is tested.

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
- TRANSMIT ReadPropertyMultiple-Request,
- 'Object Identifier' = ObjectX,
- 'Property Identifier' = ALL
- RECEIVE ReadPropertyMultiple-ACK,
- 'Object Identifier' = ObjectX,
- REPEAT P = (each property supported by ObjectX) DO {
- 'Property Identifier' = P,
- 'Property Value' = (any valid value for P)
- }
- }

Notes to Tester: Any proprietary properties that are supported for the object-type shall also be returned (see BACnet 15.7.3.1.2). If a property which is defined by the standard to be not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry or an error code is returned for that property. If Protocol\_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.

#### 9.20.1.8 Reading OPTIONAL Properties

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request that uses the special property identifier OPTIONAL. One instance of each object-type supported is tested.

Test Steps:

1. REPEAT ObjectX = (one instance of each supported object type) DO {
- TRANSMIT ReadPropertyMultiple-Request,
- 'Object Identifier' = ObjectX,
- 'Property Identifier' = OPTIONAL
- RECEIVE ReadPropertyMultiple-ACK,

```

 'Object Identifier' = ObjectX,
 REPEAT P = (each optional property supported by Object1) DO {
 'Property Identifier' = P,
 'Property Value' = (the value of P specified in the EPICS)
 }
 }

```

Notes to Tester: If no optional properties are supported then an empty 'List of Results' shall be returned for the specified property.

#### 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. One instance of each object-type supported is tested. The property identifier REQUIRED means that only those standard properties having a conformance code of "R" or "W" shall be returned.

Test Steps:

```

1. REPEAT ObjectX = (one instance of each supported object type) DO {
 TRANSMIT ReadPropertyMultiple-Request,
 'Object Identifier' = ObjectX,
 'Property Identifier' = REQUIRED
 RECEIVE ReadPropertyMultiple-ACK,
 'Object Identifier' = ObjectX,
 REPEAT P = (each required property defined for ObjectX) DO {
 'Property Identifier' = P,
 'Property Value' = (any valid value for P)
 }
 }

```

Notes to Tester: If a property which is defined by the standard to be not readable using the ReadPropertyMultiple service is in the specified object, and Protocol\_Revision < 7, then either no entry or an error code is returned for that property. If Protocol\_Revision >= 7, then the entry shall contain 'Error Class': PROPERTY and 'Error-Code': READ\_ACCESS\_DENIED for that property. Property\_List (371) shall not appear in the List of Results.

#### 9.20.1.10 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 =  
 $(\text{MaxAPDU} - \text{NonSegRqstHdrSize} - \text{ObjIdSize} - \text{TagsSize}) / \text{PropIdSize} =$   
 $(\text{MaxAPDU} - 11) / 2$

If the IUT does support receiving segmented requests:

MaxPropsPerRqst =  
 $((\text{MaxAPDU} - \text{SegRqstHdrSize}) * \text{MaxRxSegs} - \text{ObjIdSize} - \text{TagsSize}) / \text{PropIdSize} =$   
 $((\text{MaxAPDU} - 6) * \text{MaxRxSegs} - 7) / 2$

If the IUT does not support sending segmented responses:

MaxPropsPerResp =  
 $(\text{MaxAPDU} - \text{NonSegRespHdrSize} - \text{ObjIdSize} - \text{TagsSize}) / (\text{PropIdSize} + \text{TagsSize} + \text{ObjIdSize}) =$   
 $(\text{MaxAPDU} - 10) / 9$

If the IUT does support sending segmented responses:

MaxPropsPerResp =  
 $((\text{MaxAPDU} - \text{SegRespHdrSize}) * \text{MaxTxSegs} - \text{ObjIdSize} - \text{TagsSize}) / (\text{PropIdSize} + \text{TagsSize} + \text{ObjIdSize}) =$   
 $((\text{MaxAPDU} - 5) * \text{MaxTxSegs} - 7) / 9$

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)

### 9.2.0.2 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.

#### 9.2.0.2.1 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)

#### 9.2.0.2.2 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  $\geq$  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.

Dependencies: None.

BACnet Reference Clause: 15.8.

Configuration Requirements: The IUT shall be configured with a Trend Log object that contains a set of known trend data. The TD must have exact knowledge of the trend data 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.

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 Steps:

1. TRANSMIT ReadRange-Request,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer
2. RECEIVE Read-Range-ACK,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer,  
    'Result Flags' = {TRUE, TRUE, FALSE},  
    'Item Count' = (the number of trend records in the test object),  
    'Item Data' = (all of the trend records in the test object)

Notes to Tester: The trend data 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.

#### 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 Log\_Buffer. 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.

Test Steps:

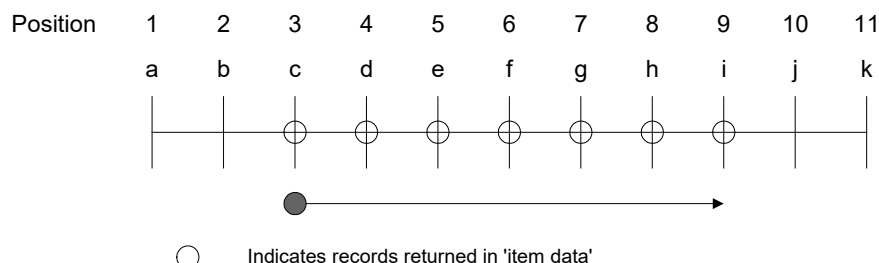
1. TRANSMIT ReadRange-Request,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer,  
    'Reference Index' = (any value  $x$ :  $1 \leq x \leq \text{Record\_Count}$ ),  
    'Count' = (any value  $x$ :  $0 < x \leq \text{Record\_Count} - x + 1$ )
2. RECEIVE Read-Range-ACK,  
    'Object Identifier' = (the log object configured for this test),  
    'Property Identifier' = Log\_Buffer,  
    'Result Flags' = {?, ?, FALSE},  
    'Item Count' = (the same value used in the 'Count' parameter in step 1),  
    'Item Data' = (all of the specified trend records 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.

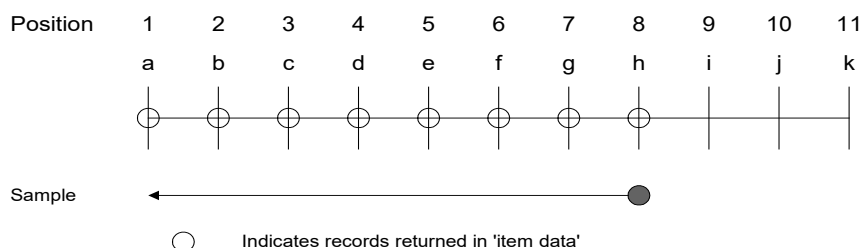
**Test Concept:** A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Position' option and a negative value for 'Count'. The 'Reference Index' and 'Count' are selected so that the results can be conveyed in a single acknowledgement.

**Test Steps:**

1. TRANSMIT ReadRange-Request,  
 'Object Identifier' = (the log object configured for this test),  
 'Property Identifier' = Log\_Buffer,  
 'Reference Index' = (any value  $x$ :  $1 \leq x \leq \text{Record\_Count}$ ),  
 'Count' = (any value  $y$ :  $y < 0$  AND  $|y| \leq x$ )
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 position. The items specified include the item at the index specified by  $x$ , 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.

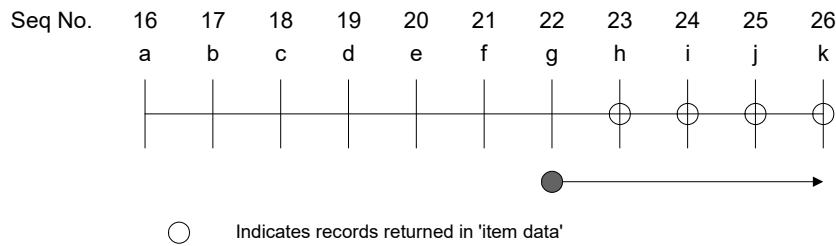
**Test Steps:**

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.

**Test Configuration:** Configure the TD such that no time change requests occur. Configure the TD such that it contains at least 3 items in the Log\_Buffer.

**Test Steps:**

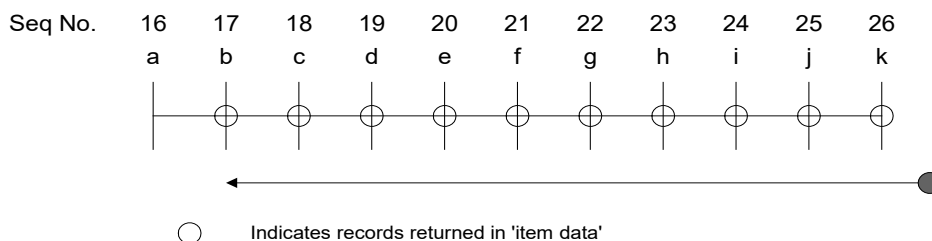
1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Time' = ( $x$ , selected as described above),
  - 'Count' = (any value  $y$ :  $0 < |y| \leq \text{number of records in the buffer}$ )
2. RECEIVE 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' =  $(\text{Total\_Record\_Count} - |y| + 1)$

**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 Reading Items by Time Range

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request to return items specified by indicating a range of times that are to be included.

**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 "Time Range" option. The 'Beginning Time' and 'Ending Time' are selected so that the results can be conveyed in a single acknowledgement.

**Test Steps:**

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the Trend Log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Beginning Time' = (any value before the last time in the buffer),
  - 'Ending Time' = (any value > 'Beginning Time')
2. RECEIVE Read-Range-ACK,
  - 'Object Identifier' = (the Trend Log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Result flags' = {TRUE, TRUE, FALSE},
  - 'Item Count' = (the number of trend records meeting the specified criteria),
  - 'Item Data' = (all of the specified trend records)

**Notes to Tester:** The first item returned shall be the first one in the buffer that has a timestamp newer (later time) than the time specified by the 'Beginning Time' parameter. The last item returned shall be the one with a timestamp older (earlier time) than or equal to the one specified by the 'Ending Time' parameter

#### 9.21.1.6 Reading a Range of Items that do not Exist

**Purpose:** To verify that the IUT correctly responds to a ReadRange service request when there are no items within the specified range.

**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 Trend Log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Beginning Time' = (any value that will result in a time interval for which there are no items present),
  - 'Ending Time' = (any value that will result in a time interval for which there are no items present)
2. RECEIVE Read-Range-ACK,
  - 'Object Identifier' = (the Trend Log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - '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:

1. 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)
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 Time' = 12:00:00.00
  - 'Count' = -10
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)

#### 9.21.1.9 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:

1. TRANSMIT ReadRange-Request,
  - 'Object Identifier' = (the log object configured for this test),
  - 'Property Identifier' = Log\_Buffer,
  - 'Reference Sequence Number' = (any value  $x$ :  $(\text{Total\_Record\_Count} - \text{Record\_Count} + 1) \leq x \leq (\text{Total\_Record\_Count} - y + 1)$ ),
  - '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 trend 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.

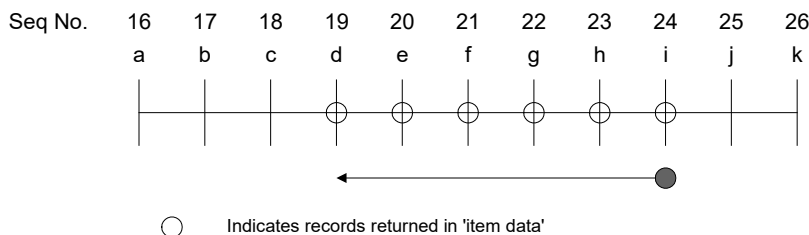
**Test Concept:** A ReadRange request is transmitted by the TD requesting a range of items known to be in the Log\_Buffer. This range is specified using the 'By Sequence' option and a negative value for 'Count'. The 'Reference Sequence Number' and 'Count' are selected so that the results can be conveyed 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 Data Type Verification Test**

Purpose: To verify that the ReadRange service performs properly for each data type supported.

Test Concept: Set Log\_DeviceObjectProperty to an external property and verify that ReadRange performs properly after data has been acquired.

Test Configuration: Set Log\_DeviceObjectProperty to an external property of type (REAL, UNSIGNED, INTEGER, BOOLEAN, BIT STRING, ENUMERATED). Make the IUT collect samples. Then use one of the qualifying Clause 9.21.1 tests to verify the operation of the ReadRange-Request and ReadRange-Ack. Qualifying tests are: 9.21.1.1, 9.21.1.2, 9.21.1.3, 9.21.1.4, 9.21.1.4.1, 9.21.1.7 and 9.21.1.8.

**9.21.1.12 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. If the error is conveyed by an error response from a remote device, verify that the Error Class and Error Code in the response is logged.

Test Concept: Make the monitored object fail and respond with an error by setting the Log\_DeviceObjectProperty to an invalid device or object. 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.

Test Steps:

1. WRITE (Invalid object into the Log\_DeviceObjectProperty of the log object)
2. WAIT (until IUT attempts to read a sample for the Log\_Buffer)
3. VERIFY (Log\_Buffer contains a failure entry of unknown object)

**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' =  $(x - z + 1)$

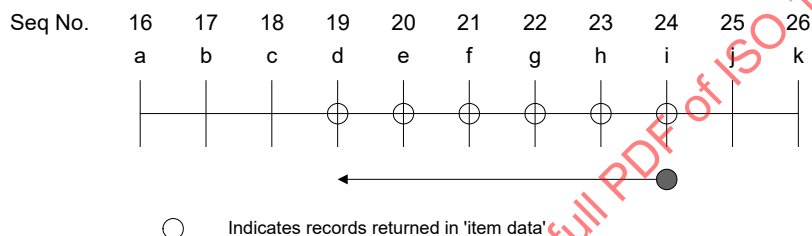
Notes to Tester: Although expressed as a bySequence exchange, these could alternately be expressed byTime.

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                                          |



## 9.22 WriteProperty Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing WriteProperty service requests.

Dependencies: None.

BACnet Reference Clause: 15.9.

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. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WriteProperty-Request,
 

|                         |          |
|-------------------------|----------|
| 'Object Identifier' =   | Object1, |
| 'Property Identifier' = | P1,      |



'Property Array Index' = (any value N:  $1 \leq N \leq$  the size of the array)  
'Property Value' = (any value of the correct datatype for this array except the value verified for this element in step 1)

3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2), ARRAY INDEX = N

#### 9.22.1.2 Writing a Commandable Property Without a Priority

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. VERIFY (Object1), Priority\_Array = (the value defined for this property in the EPICS), ARRAY INDEX = 16
2. TRANSMIT WriteProperty-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = Present\_Value,  
'Property Value' = (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1)
3. RECEIVE Simple-ACK-PDU
4. VERIFY (Object1), Priority\_Array = (the value used in step 2), ARRAY INDEX = 16

#### 9.22.1.3 Writing a Non-Commandable Property with a Priority

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. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WriteProperty-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = P1,  
'Priority' = (any valid priority)  
'Property Value' = (any valid value defined for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1)
3. RECEIVE BACnet-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. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1),  
     'Property Array Index' = (any positive integer)
3. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  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 = (the value defined for this property in the EPICS)

#### 9.22.2.2 Writing Array Properties with an Array Index that is Out of Range

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. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1),  
     'Property Array Index' = (any positive integer that is larger than the supported size of the array)
3. RECEIVE BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = INVALID\_ARRAY\_INDEX
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

#### 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 invalid for the property. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ V = (Object1), P1
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value with an invalid datatype)
3. RECEIVE (BACnet-Error PDU,  
     Error Class = PROPERTY,  
     Error Code = INVALID\_DATATYPE)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_PARAMETER\_DATATYPE)  
     | (BACnet-Reject-PDU  
         Reject Reason = INVALID\_TAG)
4. VERIFY (Object1), P1 = V

#### 9.22.2.4 Writing with a Property Value that is Out of Range

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

Test Concept: The TD attempts to write to a property using a value that is outside of the supported range. If the IUT does not contain any writable properties that have restricted ranges, then this test shall be skipped.

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS);
2. TRANSMIT WriteProperty-Request,  
     'Object Identifier' = (Object1, any object with writable properties),  
     'Property Identifier' = (P1, any writable property with a restricted range of values),  
     'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. IF (Protocol\_Revision is present AND Protocol\_Revision  $\geq$  4) THEN  
     RECEIVE (BACnet-Error PDU,  
         Error Class = PROPERTY,  
         Error Code = VALUE\_OUT\_OF\_RANGE)  
     ELSE  
         RECEIVE (BACnet-Error-PDU,  
             Error Class = PROPERTY,  
             Error Code = VALUE\_OUT\_OF\_RANGE) |  
         (BACnet-Reject-PDU,  
             Reject Reason = PARAMETER\_OUT\_OF\_RANGE)
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

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 and for constructed values, the constructed value shall match the structure defined by the ASN.1, and all field values shall be within the ranges defined by the standard for those field values.

#### 9.22.2.5 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

### 9.23 WritePropertyMultiple Service Execution Tests

This clause defines the tests necessary to demonstrate support for executing WritePropertyMultiple service requests.

Dependencies: None.

BACnet Reference Clause: 15.10.

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\_Identifiers 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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,  
    'Object Identifier' = Object1,  
    'Property Identifier' = P1,  
    'Property Value' = (any value of the appropriate datatype except for the one verified in step 1)
3. RECEIVE BACnet-SimpleACK-PDU

4. VERIFY (Object1), P1 = (the value specified in step 2)

### 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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)
3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any value of the appropriate datatype except for the one verified in step 1),
  - 'Property Identifier' = P2,
  - 'Property Value' = (any value of the appropriate datatype except for the one verified in step 2)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 2)
6. VERIFY (Object1), P2 = (the value specified for P2 in step 2)

### 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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. VERIFY (Object2), P2 = (the value specified for this property in the EPICS)
3. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any value of the appropriate datatype except for the one verified in step 1),
  - 'Object Identifier' = Object2,
  - 'Property Identifier' = P2,
  - 'Property Value' = (any value of the appropriate datatype except for the one verified in step 2)
4. RECEIVE BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object2), P2 = (the value specified for P2 in step 3)

### 9.23.1.4 Writing Multiple Properties to Multiple Objects

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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)
3. VERIFY (Object2), P3 = (the value specified for this property in the EPICS)
4. VERIFY (Object2), P4 = (the value specified for this property in the EPICS)
5. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' = Object1,  
     'Property Identifier' = P1,  
     'Property Value' = (any value of the appropriate datatype except for the one verified in step 1),  
     'Property Identifier' = P2,  
     'Property Value' = (any value of the appropriate datatype except for the one verified in step 2),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P3,  
     'Property Value' = (any value of the appropriate datatype except for the one verified in step 3),  
     'Object Identifier' = Object2,  
     'Property Identifier' = P4,  
     'Property Value' = (any value of the appropriate datatype except for the one verified in step 4)
6. RECEIVE BACnet-BACnet-SimpleACK-PDU
7. VERIFY (Object1), P1 = (the value specified for P1 in step 5)
8. VERIFY (Object1), P2 = (the value specified for P2 in step 5)
9. VERIFY (Object2), P3 = (the value specified for P3 in step 5)
10. VERIFY (Object2), P4 = (the value specified for P4 in step 5)

#### 9.23.1.5 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.

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' = (Device, X),
  - '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 Simple-ACK
3. VERIFY (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 Simple-ACK-PDU
4. VERIFY (Object1), P1 = (the value used in step 2)

#### 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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,
 

|                         |                                                                                |
|-------------------------|--------------------------------------------------------------------------------|
| 'Object Identifier' =   | Object1,                                                                       |
| 'Property Identifier' = | P1,                                                                            |
| 'Property Value' =      | (any value of the appropriate datatype except for the one verified in step 1), |
| 'Property Identifier' = | P2,                                                                            |
| 'Property Value' =      | (any value of the appropriate datatype)                                        |
3. RECEIVE WritePropertyMultiple-Error,
 

|                      |                   |
|----------------------|-------------------|
| Error Class =        | PROPERTY,         |
| Error Code =         | UNKNOWN_PROPERTY, |
| objectIdentifier =   | Object1,          |
| propertyIdentifier = | P2                |
4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

### 9.23.2.2 Writing Multiple Properties with an Object Access Error

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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,
 

|                         |                                                                                |
|-------------------------|--------------------------------------------------------------------------------|
| 'Object Identifier' =   | Object1,                                                                       |
| 'Property Identifier' = | P1,                                                                            |
| 'Property Value' =      | (any value of the appropriate datatype except for the one verified in step 1), |
| 'Object Identifier' =   | BadObject,                                                                     |
| 'Property Identifier' = | P2,                                                                            |
| 'Property Value' =      | (any value of the appropriate datatype)                                        |
3. RECEIVE WritePropertyMultiple-Error,
 

|               |                 |
|---------------|-----------------|
| Error Class = | OBJECT,         |
| Error Code =  | UNKNOWN_OBJECT, |

objectIdentifier = BadObject,  
propertyIdentifier = P2

4. VERIFY (Object1), P1 = (the value specified for P1 in step 2)

#### 9.23.2.3 Writing Multiple Properties with a Write Access Error

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. VERIFY (Object1), P1 = (the value specified for this property in the EPICS)
2. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)
3. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = P1,  
'Property Value' = (any value of the appropriate datatype except for the one verified in step 1),  
'Property Identifier' = P2,  
'Property Value' = (any value of the appropriate datatype except for the one verified in step 1)
4. RECEIVE WritePropertyMultiple-Error,  
Error Class = PROPERTY,  
Error Code = WRITE\_ACCESS\_DENIED,  
objectIdentifier = Object1,  
propertyIdentifier = P2
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object1), P2 = (the value specified for this property in the EPICS)

#### 9.23.2.4 Writing Non-Array Properties with an Array Index

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

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

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

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = Object1,  
'Property Identifier' = P1,  
'Property Value' = (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1),  
'Property Array Index' = (any positive integer)

3. RECEIVE WritePropertyMultiple-Error,  
     Error Class =       PROPERTY,  
     Error Code =       PROPERTY\_IS\_NOT\_AN\_ARRAY,  
     objectIdentifier =   Object1,  
     propertyIdentifier =   P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

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

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

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

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

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' =   Object1,  
     'Property Identifier' =   P1,  
     'Property Value' =       (any value of the correct datatype for this property subject to the restrictions specified in the EPICS as defined in 4.4.2, except the value verified in step 1),  
     'Property Array Index' = (any positive integer that is larger than the supported size of the array)
3. RECEIVE WritePropertyMultiple-Error,  
     Error Class =       PROPERTY,  
     Error Code =       INVALID\_ARRAY\_INDEX,  
     objectIdentifier =   Object1,  
     propertyIdentifier =   P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

#### 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 invalid for the property. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ V = (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 = (the value defined for this property in the EPICS)

#### 9.23.2.7 Writing with a Property Value that is Out of Range

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when an attempt is made to write a value that is outside of the supported range. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. The TD attempts to write to this property using a value that is outside of the supported range.

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS),
2. TRANSMIT WritePropertyMultiple-Request,  
'Object Identifier' = (Object1, any object with writable properties),  
'Property Identifier' = (P1, any property with a restricted range of values),  
'Property Value' = (any value, of the correct datatype, that is outside the supported range)
3. RECEIVE WritePropertyMultiple-Error,  
Error Class = PROPERTY,  
Error Code = VALUE\_OUT\_OF\_RANGE,  
objectIdentifier = Object1,  
propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

#### 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

#### 9.24 DeviceCommunicationControl Service Execution Test

This clause defines the tests necessary to demonstrate support for executing DeviceCommunicationControl service requests.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 16.1.

##### 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.

Dependencies: ReinitializeDevice Service Execution Tests, 9.27.

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 > **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. 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 > **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 ReinitializeDevice-Request,
  - 'Reinitialize State of Device' = WARMSTART,
  - 'Password' = (any appropriate password as described in the Test Concept)
- 8. RECEIVE BACnet-SimpleACK-PDU
- 9. CHECK (Did the IUT perform a WARMSTART reboot?)
- 10. VERIFY (Device, X), (any required non-array property) = (the value for this property as described in the EPICS)

#### 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

Dependencies: ReinitializeDevice Service Execution Tests, 9.27.

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

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
  - 'Enable/Disable' = DISABLE-INITIATION,
  - 'Password' = (any appropriate password as described in the Test Concept)
2. RECEIVE BACnet-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**

BACnet Reference Clause: 16.1.1.1.2.

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**

BACnet Reference Clause: 16.1.1.1.2.

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)

#### 9.24.2 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,  
    '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,
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)

### 9.24.2.3 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
  - RECEIVE BACnet-Error-PDU,
    - Error Class = SERVICES,
    - Error Code = COMMUNICATION\_DISABLED
  - ELSE
    - CHECK(that the IUT responded with BACnet-Error-PDU with an Error Class of SERVICES and an Error Code of COMMUNICATION\_DISABLED, or 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.

Dependencies: None.

BACnet Reference Clause: 16.2.

Purpose: To verify the ability to correctly execute a ConfirmedPrivateTransfer service request.

Test Concept: The service procedure implied by a particular private transfer service is defined by the vendor. This test simply verifies that an appropriate acknowledgment is returned and that any externally visible actions defined by the vendor are observed.

Configuration Requirements: The IUT shall be configured to execute at least one ConfirmedPrivateTransfer service. The service parameters that are to be provided in the request and a list of any externally visible actions that should be apparent to the tester shall also be provided.

Test Steps:

1. TRANSMIT ConfirmedPrivateTransfer-Request,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the EPICS),  
     'Service Number' = (any service number provided by the vendor),  
     'Service Parameters' = (the service parameters provided for this service)
2. RECEIVE ConfirmedPrivateTransfer-ACK,  
     'Vendor ID' = (the Vendor\_Identifier specified in the Device object of the EPICS),  
     'Service Number' = (the service number used in step 1),  
     'Result Block' = (the expected results provided by the vendor)
3. CHECK (Did the externally visible actions take place?)

#### **9.26 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.

Dependencies: None.

BACnet Reference Clause: 16.4.

##### **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 COLDSTART with an Invalid Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a COLDSTART is attempted and an invalid password is provided. If the IUT does not provide password protection this test case shall be omitted.

Test Steps:

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

- Error Code = PASSWORD\_FAILURE  
3. CHECK (Did the IUT reboot?)

Notes to Tester: The IUT shall not reinitialize.

#### 9.27.2.2 WARMSTART with an Invalid Password

Purpose: To verify the correct execution of the ReinitializeDevice request service procedure when a WARMSTART is attempted and an invalid password is provided. If the IUT does not provide password protection this test case shall be omitted.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = WARMSTART,  
    'Password' = (any invalid password)
2. RECEIVE BACnet-Error-PDU,  
    Error Class = SECURITY,  
    Error Code = PASSWORD\_FAILURE
3. CHECK (Did the IUT reboot?)

Notes to Tester: The IUT shall not reinitialize.

#### 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 a password is required but no password is provided.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = COLDSTART,
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) |  
        (BACnet-Error-PDU,  
            Error Class = SERVICES,  
            Error Code = MISSING\_REQUIRED\_PARAMETER)
3. CHECK (The IUT did NOT 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.

#### 9.27.2.4 WARMSTART with Missing Password

Purpose: To verify that the correct BACnet Error PDU is returned when a WARMSTART is attempted and a password is required but no password is provided.

Test Steps:

1. TRANSMIT ReinitializeDevice-Request,  
    'Reinitialized State of Device' = WARMSTART,
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 |
 (BACnet-Error-PDU,
 Error Class = SERVICES,
 Error Code = MISSING_REQUIRED_PARAMETER)
3. CHECK (The IUT did NOT perform a WARMSTART reboot)

```

Notes to Tester: External indications that the IUT has reinitialized, such as LEDs or startup message traffic, shall be used to confirm reinitialization whenever possible.

### 9.28 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.

BACnet Reference Clause: 16.5.

#### 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

BACnet does not define a service procedure for executing the UnconfirmedTextMessage service and thus no tests are needed.

### 9.30 TimeSynchronization Service Execution Tests

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 16.7.

#### 9.30.1 Positive TimeSynchronization Service Execution Tests

The purpose of this test group is to verify correct execution of TimeSynchronization service requests under circumstances where the service is expected to be successfully completed.

##### 9.30.1.1 Local Broadcast

Purpose: To verify that the IUT resets its local time and date in response to a local broadcast TimeSynchronization service request.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Date
2. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Date,  
 'Property Value' = (any valid date referred to as "InitialDate" below)
3. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Time
4. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Time,  
 'Property Value' = (any valid time referred to as "InitialTime" below)
5. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = UTC\_Offset
6. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = UTC\_Offset,  
 'Property Value' = (any valid offset referred to as "InitialUTC\_Offset" below)
7. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Daylight\_Savings\_Status
8. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Daylight\_Savings\_Status,  
 'Property Value' = (any valid status referred to as "InitialDaylight\_Savings\_Status" below)
9. TRANSMIT  
 DA = LOCAL BROADCAST,  
 SA = TD,  
 BACnet-Unconfirmed-Request-PDU,  
 'Service Choice' = TimeSynchronization-Request,  
 date = (any date other than InitialDate),  
 time = (any time that does not correspond to InitialTime)
10. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Date
11. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Date,  
 'Property Value' = (the date specified in step 9)
12. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Time
13. RECEIVE ReadProperty-ACK,  
 'Object Identifier' = (the IUT's Device object),  
 'Property Identifier' = Local\_Time,  
 'Property Value' = (the time specified in step 9)

Notes to Tester: The time value returned by the IUT in step 13 shall agree with the time specified in step 9 within the resolution for time specified in the EPICS. If the time returned by the IUT indicates that a small amount of time has passed (< 1 second) since the TimeSynchronization request was received the result shall be considered to be a pass. If the time indicates that the day of week is unspecified but all other fields are correct the result shall be considered to be a pass.

#### 9.30.1.2 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.

Test Steps: This test is identical to 9.30.1.1 except that the TimeSynchronization-Request in step 9 shall be transmitted using the IUT's MAC address as the destination.

Notes to Tester: The passing results are identical to 9.30.1.1.

#### 9.31 UTCTimeSynchronization Service Execution Tests

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 16.8.

##### 9.31.1 Positive UTCTimeSynchronization Service Execution Tests

The purpose of this test group is to verify correct execution of UTCTimeSynchronization service 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.

Test Steps: The test steps are identical to the steps in 9.30.1.1 except that in step 9 the UTCTimeSynchronization request is used and the date and time conveyed represent UTC.

Passing Results: The passing results are identical to 9.30.1.1 except that the date in step 9 shall be corrected for InitialUTC\_Offset, and the time in step 13 shall be corrected for both InitialUTC\_Offset and Daylight\_Savings\_Status (as defined in BACnet 16.7.2).

###### 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 9.30.1.1 except that in step 9 the UTCTimeSynchronization request is used and the date and time conveyed represent UTC and the UTCTimeSynchronization-Request shall be transmitted using the IUT's MAC address as the destination.

Notes to Tester: The passing results are identical to 9.30.1.1.

#### 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.

Dependencies: None.

BACnet Reference Clause: 16.9.

##### 9.32.1 Execution of Who-Has Service Requests Originating from the Local Network

The purpose of this test group is to verify the correct execution of the Who-Has request service procedure for messages originating from the local network.

###### 9.32.1.1 Object ID Version with No Device Range

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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Object Identifier' = (any object identifier specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in step 1),
  - 'Object Name' = (the object name specified in the EPICS for this object)

#### 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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Object Name' = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in the EPICS for this object),
  - 'Object Name' = (the object name specified in step 1)

#### 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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),
  - 'Object Identifier' = (any object identifier specified in the EPICS),
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in step 1),
  - 'Object Name' = (the object name specified in the EPICS for this object)



**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.

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' = (any object identifier specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

**9.32.1.5 Object Name Version with IUT Inside of the Device Range**

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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),
  - 'Object Name' = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in the EPICS for this object),
  - 'Object Name' = (the object name specified in step 1)

**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),  
(any object name specified in the EPICS)

'Object Name' =

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.

Test Steps:

1. TRANSMIT

DA = LOCAL BROADCAST,

SA = TD,

Who-Has-Request,

'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),

'Device Instance High Limit' = (The Device object instance number of the IUT)

'Object Identifier' = (any object identifier specified in the EPICS)

2. WAIT **Internal Processing Fail Time**

3. RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST,

SA = IUT,

I-Have-Request,

'Device Identifier' = (the IUT's Device object),

'Object Identifier' = (the object identifier specified in step 1),

'Object Name' = (the object name specified in the EPICS for this object)

#### 9.32.1.8 Object ID Version with IUT Device Instance Equal to the Low Limit of the Device Range

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.

Test Steps:

1. TRANSMIT

DA = LOCAL BROADCAST,

SA = TD,

Who-Has-Request,

'Device Instance Low Limit' = (The Device object instance number of the IUT),

'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),

'Object Identifier' = (any object identifier specified in the EPICS)

2. WAIT **Internal Processing Fail Time**

3. RECEIVE

DA = LOCAL BROADCAST | GLOBAL BROADCAST,

SA = IUT,

I-Have-Request,

'Device Identifier' = (the IUT's Device object),

'Object Identifier' = (the object identifier specified in step 1),

'Object Name' = (the object name specified in the EPICS for this object)

#### 9.32.1.9 Object Name Version with IUT Device Instance Equal to the High Limit of the Device Range

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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (any value L:  $0 \leq L <$  the Device object instance number of the IUT),
  - 'Device Instance High Limit' = (The Device object instance number of the IUT),
  - 'Object Name' = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in the EPICS for this object),
  - 'Object Name' = (the object name specified in step 1)

#### 9.32.1.10 Object Name Version with IUT Device Instance Equal to the Low Limit of the Device Range

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.

Test Steps:

1. TRANSMIT
  - DA = LOCAL BROADCAST,
  - SA = TD,
  - Who-Has-Request,
  - 'Device Instance Low Limit' = (The Device object instance number of the IUT),
  - 'Device Instance High Limit' = (any value H:  $H >$  the Device object instance number of the IUT),
  - 'Object Name' = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in step 1),
  - 'Object Name' = (the object name specified in the EPICS for this object)

#### 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.

Test Steps:

1. TRANSMIT Who-Has-Request,
  - 'Object Name' = (any object name specified in the EPICS),
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DA = LOCAL BROADCAST | GLOBAL BROADCAST,
  - SA = IUT,
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in the EPICS for this object),
  - 'Object Name' = (the object name specified in step 1)

### 9.32.1.12 Who-Has After Object\_Name Changed

Dependencies: Who-Has Service Execution Tests, 9.32.1.2

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Name property of an object in the device is changed.

Test Concept: The Object\_Name property of the referenced object is read to determine its initial value. The Object\_Name property is then changed to a different value, V2, which is not already used by an object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Name' parameter, using the values V1 and V2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Name property and has the value V1. If IUT does not support objects with modifiable Object\_Name properties, then this test shall be skipped.

Test Steps:

1. READ V1 = O1, Object\_Name
2. IF (Object\_Name is writable) THEN  
    WRITE O1, Object\_Name = V2  
ELSE  
    MAKE (O1, Object\_Name = V2)
3. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Name' = V1
4. WAIT **Internal Processing Fail Time**
5. CHECK (Verify that the IUT does not respond with an I-Have request)
6. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Name' = V2
7. RECEIVE  
    DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
    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

Dependencies: Who-Has Service Execution Tests, 9.32.1.1

BACnet Reference Clause: 16.9

Purpose: To verify that a device correctly responds to Who-Has service requests after the Object\_Identifier property of an object in the device is changed.

Test Concept: The Object\_Identifier property of the referenced object, O1, is verified to contain the value O1. The Object\_Identifier property is then changed to a different value, O2, which is not already in use by a different object in the IUT. The test then verifies correct responses to Who-Has requests that include an 'Object Identifier' parameter, using the values O1 and O2.

Configuration: An object, O1, exists within the IUT that has a modifiable Object\_Identifier property. If the IUT does not support objects with modifiable Object\_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 **Internal Processing Fail Time**
5. CHECK (Verify that the IUT does not respond with an I-Have request)
6. TRANSMIT  
    DESTINATION = GLOBAL BROADCAST,  
    Who-Has-Request,  
    'Object Identifier' = O2
7. RECEIVE  
    DESTINATION = LOCAL BROADCAST | GLOBAL BROADCAST,  
    I-Have-Request,  
    'Device Identifier' = (the IUT's Device object),  
    'Object Identifier' = O2

### 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.

Test Steps:

1. TRANSMIT  
    DESTINATION = LOCAL BROADCAST,  
    SA = ID,  
    DNET = GLOBAL BROADCAST,  
    SNET = (any remote network number),  
    SADR = (any MAC address valid for the specified network),  
    Who-Has-Request,  
    'Object Identifier' = (any object identifier specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE  
    DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified in step 1),  
    I-Have-Request,  
    'Device Identifier' = (the IUT's Device object),  
    'Object Identifier' = (the object identifier specified in step 1),  
    'Object Name' = (the object name specified in the EPICS for this object)

#### 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.

Test Steps:

1. TRANSMIT
  - DESTINATION = LOCAL BROADCAST,
  - SA = TD,
  - SNET = (any remote network number),
  - SADR = (any MAC address valid for the specified network),
  - Who-Has-Request,
  - 'Object Identifier' = (any object identifier specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
  - DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified in step 1),
  - I-Have-Request,
  - 'Device Identifier' = (the IUT's Device object),
  - 'Object Identifier' = (the object identifier specified in step 1),
  - 'Object Name' = (the object name specified in the EPICS for this object)

### 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.

Dependencies: None.

BACnet Reference Clause: 16.10.

##### 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 > 0 such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),

'Device Instance Range High Limit' = (any value > Device Instance Low Limit such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit)

## 2. WAIT Internal Processing Fail Time

## 3. CHECK (verify that the IUT does not respond)

**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:  $0 \leq L <$  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 =

SNET =

SADR =

Who-Is-Request

GLOBAL BROADCAST,

(any remote network number),

(any MAC address valid for the specified network),

2. BEFORE **Unconfirmed Response Fail Time**

RECEIVE

DESTINATION =

network

GLOBAL BROADCAST | REMOTE BROADCAST (to the

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 that the IUT responds with an I-Am service that is of the form global broadcast, remote broadcast or unicast.

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 | LOCAL BROADCAST  
 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.

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 17.2.

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.

Dependencies: ReadProperty Service Execution Tests, 9.18; VT-Open Service Execution Tests, 9.34.

BACnet Reference Clause: 17.3.

#### 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

Dependencies: AddListElement Service Execution, 9.14; AddListElement Service Initiation, 8.14.

BACnet Reference Clauses: 24.2.1 and 24.4.

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' = ( $SK_{AB}$ , BACnetAddress for device A)

Note: The List\_Of\_Session\_Keys property value shall be enciphered using  $PK_B$ .

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  $PK_B$ .

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  $PK_B$ . 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' = ( $SK_{AB}$ , BACnetAddress for device B)

Note: The List\_Of\_Session\_Keys property value shall be enciphered using  $PK_A$ .

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  $PK_A$ .

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  $PK_A$ .

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**

Dependencies: AddListElement Service Execution 9.14.

BACnet Reference Clauses: 24.2.1 and 24.5.

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

Dependencies: None.

BACnet Reference Clauses: 24.2.2 and 24.5.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 24.2.3 and 24.5.

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<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),

'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

Dependencies: ReadProperty Service Initiation Tests, 8.18.

BACnet Reference Clauses: 24.2.4 and 24.5.

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

Dependencies: None.

BACnet Reference Clauses: 24.2.5 and 24.5.

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

Dependencies: None.

BACnet Reference Clauses: 24.2.5 and 24.5.

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 24.3.1, 24.3.2, and 24.5.

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<sub>TD,IUT</sub>.

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),  
    '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 ins 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

Dependencies: None

BACnet Reference Clause: UNRECOGNIZED\_SERVICE, 18.9.9

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

Test Steps:

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

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

### 9.39.2 Unsupported Unconfirmed Services Test

Dependencies: None

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

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

Test Steps:

1. VERIFY System\_Status == OPERATIONAL | OPERATIONAL\_READ\_ONLY
2. REPEAT X = (all unconfirmed services that the IUT does not execute) DO {  
     TRANSMIT X  
     BEFORE **Internal Processing Fail Time**  
     CHECK (verify that the IUT did not reset and that the IUT did not send any packets)  
     VERIFY System\_Status = (the value of System\_Status read in step 1)  
   }

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

## 10. NETWORK LAYER PROTOCOL TESTS

### 10.1 General Network Layer Tests

#### 10.1.1 Processing Application Layer Messages Originating from Remote Networks

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clause: 6.5.4.

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

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

Test Steps:

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

#### 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.

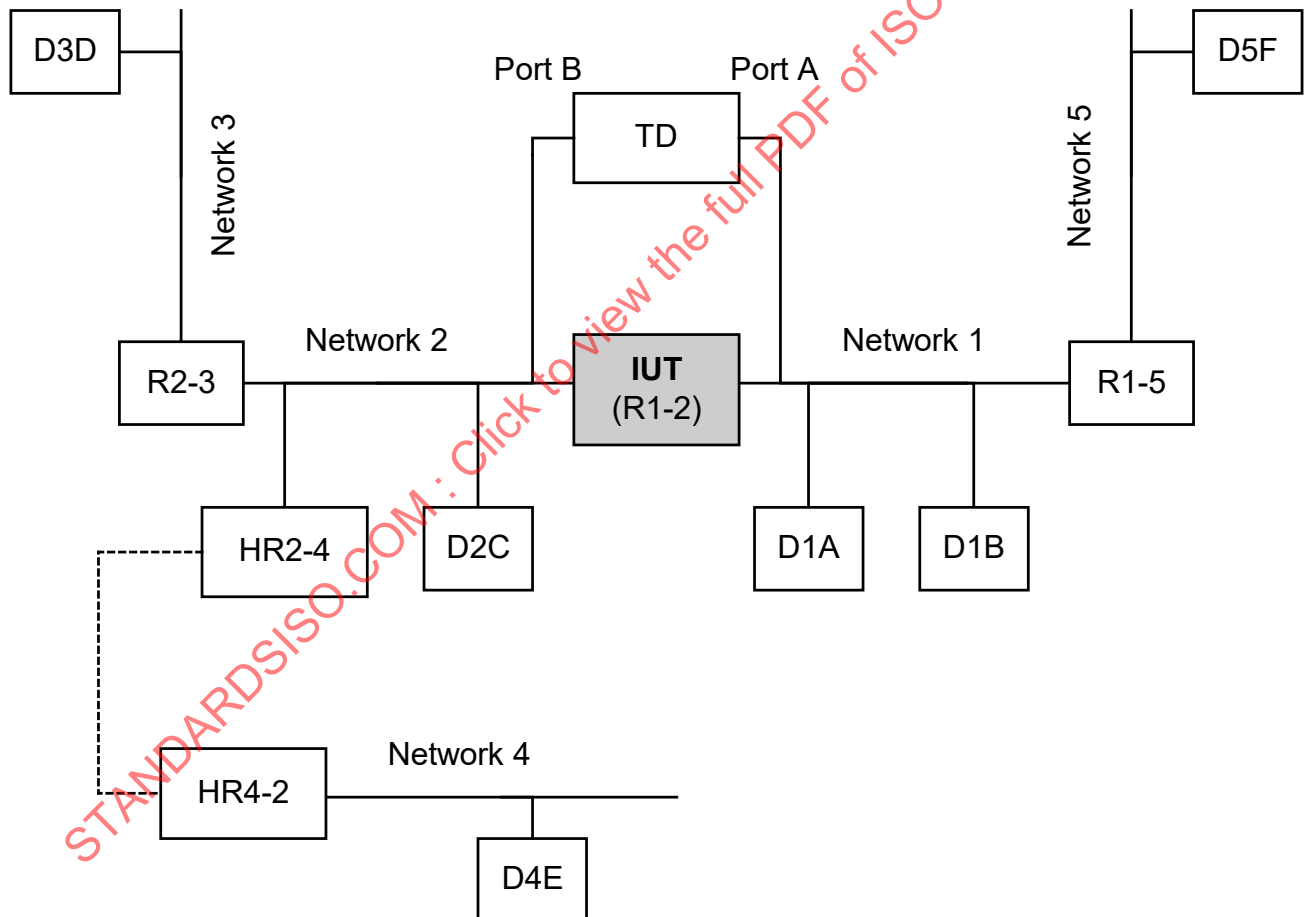


**Figure 10-1.** Logical internetwork configuration for router functionality tests

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**

BACnet Reference Clause: 6.6.2.

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**

BACnet Reference Clause: 6.6.3.3

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

Dependencies: 10.2.2.1.

BACnet Reference Clauses: 6.6.3.2 and 6.6.3.3.

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020

**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

BACnet Reference Clause: 6.6.3.4.

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

**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

BACnet Reference Clause: 6.6.3.7.

##### 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

Dependencies: 10.2.2.1, 10.2.2.2.5.

BACnet Reference Clauses: 6.4.7, 6.4.8, 6.6.1, 6.6.3.8, and 6.6.3.9.

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**

Dependencies: 10.2.2.1, 10.2.2.2.5.

BACnet Reference Clauses: 6.4.7, 6.4.8, 6.6.1, 6.6.3.8, and 6.6.3.9.

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

Dependencies: 10.2.2.1, 10.2.2.2.5.

BACnet Reference Clauses: 6.4.7, 6.4.8, 6.6.1, 6.6.3.8, and 6.6.3.9.

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.

BACnet Reference Clauses: 6.4.4 and 6.6.3.5.

#### **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 directed to the IUT that contains an unknown message type in the range of message types reserved for use by ASHRAE.

Test Steps:

1. TRANSMIT PORT A,  
DESTINATION = IUT,  
SOURCE = D1A,  
Message Type = (any value in the range reserved for use by ASHRAE that is undefined in the protocol revision claimed by the device)
2. RECEIVE PORT A,  
DESTINATION = TD,  
SOURCE = IUT,  
Reject-Message-To-Network,  
Reject 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.

BACnet Reference Clause: 6.6.3.5

Test Steps:

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

BACnet Reference Clauses: 6.2.2 and 6.5.

**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 unicast message from a local device on Network 1 to a device on Network 2.

Test Steps:

1. TRANSMIT PORT A,  
DA = IUT,  
SA = D1A,  
DNET = 2,  
DADR = D2C,  
Hop Count = 255,  
BACnet-Confirmed-Request-PDU,  
'Service Choice' = ReadProperty-Request,  
'Object Identifier' = (any object identifier),  
'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,  
DA = D2C,  
SA = IUT,  
SNET = 1,  
SADR = 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 B,  
DA = IUT,  
SA = D2C,  
DNET = 1,  
DADR = D1A,  
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 A,  
DA = D1A,

SA = IUT,  
SNET = 2,  
SADR = D2C,  
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)

#### 10.2.3.3 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.

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. RECEIVE PORT A,  
DA = R1-5,  
SOURCE = IUT,  
DNET = 5,  
DADR = D5F,  
Hop Count = 255,  
Reject-Message-To-Network,  
Reject 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.

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. TRANSMIT PORT B,  
DESTINATION = LOCAL BROADCAST,  
SOURCE = R2-3,  
I-Am-Router-To-Network,  
Network Numbers = 3
4. 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.

BACnet Reference Clauses: 6.2.2 and 6.5.

##### **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

BACnet Reference Clause: 6.2.3, 6.5.4.

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

BACnet Reference Clauses: 6.1, 6.2.2, and 6.5.4.

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)

### 10.3 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.

Dependencies: PTP State Machine Tests, 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.

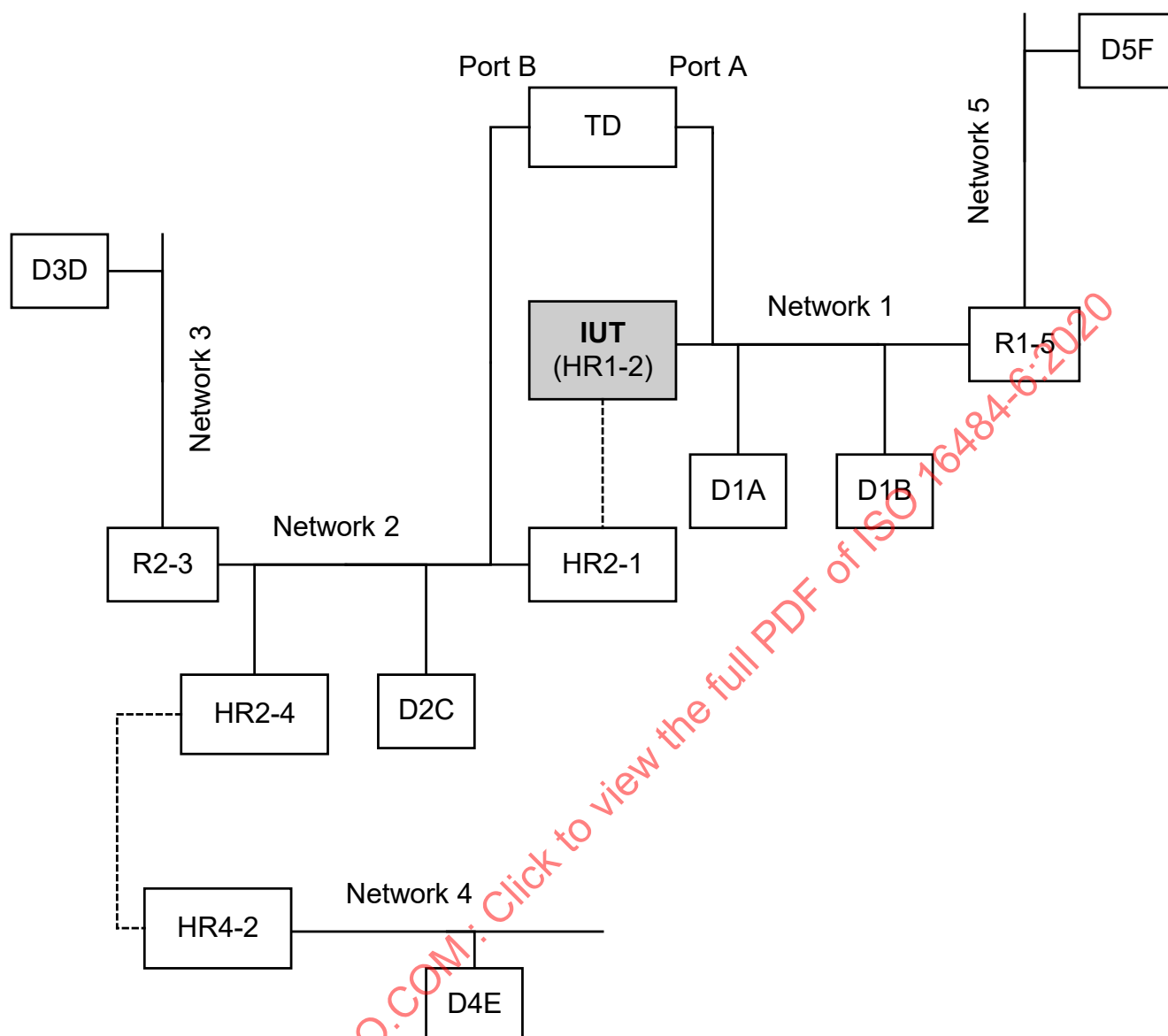


**Figure 10-2.** Logical internetwork configuration for half-router functionality tests

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.

BACnet Reference Clause: 6.6.3.4.

#### 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

BACnet Reference Clause: 6.7.4.1.1.

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**

BACnet Reference Clause: 6.7.1.

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**

Dependencies: 10.3.3.

BACnet Reference Clauses: 6.7.1.4 and 6.7.2.2.

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

BACnet Reference Clause: 6.7.1.3.

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

BACnet Reference Clause: 6.7.4.2.2.



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**

Dependencies: 10.3.5.

BACnet Reference Clause: 6.7.2.1.

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

BACnet Reference Clause: 6.7.4.2.

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

BACnet Reference Clause: 6.6.

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.

BACnet Reference Clause: Annex H.

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'.

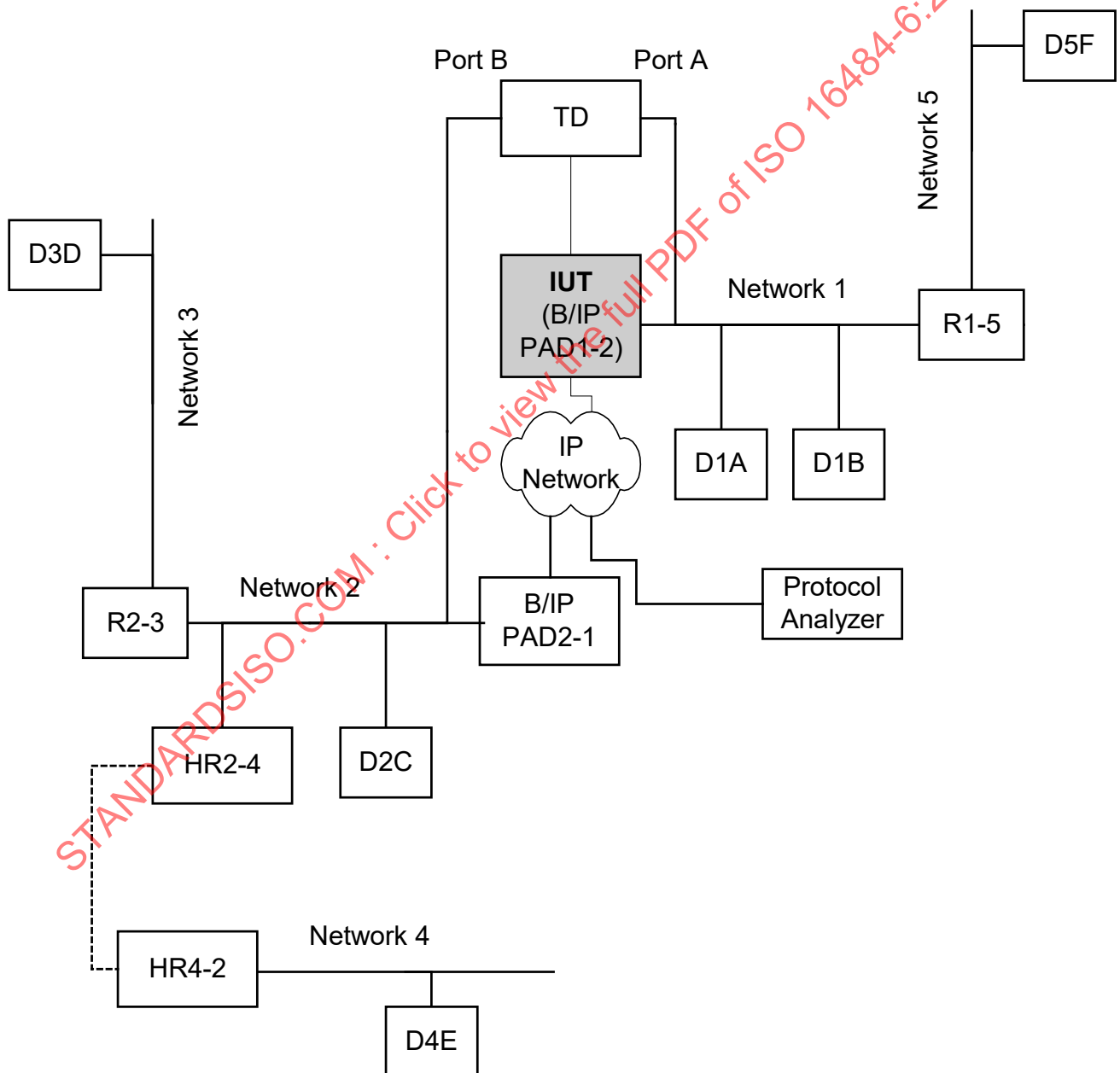


Figure 10-3. Logical internetwork configuration for B/IP router functionality tests

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.

Dependencies: None.

#### **10.5.1 Locating Routers**

BACnet Reference Clause: 6.4.1.

##### **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**

BACnet Reference Clause: 6.4.7.

##### **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.

### 10.5.3 Initiating and Terminating PTP Connections

#### 10.5.3.1 Establish-Connection-To-Network

BACnet Reference Clause: 6.4.9.

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

BACnet Reference Clause: 6.4.10.

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

BACnet Reference Clauses: 6.5.2.1, 6.5.4

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

BACnet Reference Clauses: 6.6, 6.6.3.2

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

BACnet Reference Clauses: 6.6, 6.6.3.8, 6.6.3.10, 6.6.3.11

Purpose: This test case verifies that the non-router IUT will quietly accept and discard network layer router services.

Test Concept: The TD transmits the Initialize-Routing-Table, Establish-Connection-To-Network, and Disconnect-Connection-To-Network services. The IUT is required to silently drop the requests because it is not a router.

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
Initialize-Routing-Table  
Number of Ports = 0
2. WAIT **Internal Processing Fail Time**
3. CHECK (that the IUT did not send any packets in response to the Initialize-Routing-Table)
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)
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)

### **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**

Dependencies: ReadProperty Service Initiation Tests, 8.15, ReadProperty Service Execution Tests, 9.15

BACnet Reference Clause: 6.5.3

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.

Test Configuration: 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

Dependencies: ReadProperty Service Initiation Tests, 8.18, ReadProperty Service Execution Tests, 9.18, Who-Is Service Initiation Tests, 8.34

BACnet Reference Clause: 6.5.3

Purpose: To verify that the IUT can initiate requests to a remote network and respond to requests from a remote network after the IUT uses the Who-Is and I-Am Application Layer services to discover the MAC address of the router to that remote network.

Test Concept: The IUT broadcasts a Who-Is request to discover device D2A and notes the MAC address of the intervening router in the corresponding I-Am reply. The TD transmits a request to a device on the remote network and responds to a request from the remote network without performing any further form of dynamic router binding. If the IUT does not support application layer router binding, then this test shall be omitted. If the IUT cannot initiate a ReadProperty request, then another confirmed service can be substituted. The IUT may use the deviceInstanceRange form of Who-Is.

Clause 6.5.3 specifically mentions router binding via Who-Is and does not mention router binding by initiating other application layer services (such as Who-Has) or by lurking and noting the router MAC addresses for incoming application layer requests. For this reason the test only allows for router binding via Who-Is.

Test Steps:

1. MAKE (IUT transmit Who-Is to discover the device on the remote network)
2. RECEIVE
  - DA = BROADCAST,
  - SA = IUT,
  - DNET = GLOBAL BROADCAST,
  - Hop Count = 255,
  - BACnet-Unconfirmed-Request-PDU,
  - 'Service Choice' = who-Is
  - (DA = BROADCAST,
  - SA = IUT,
  - DNET = DNET2,
  - DADR = BROADCAST,
  - 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**

Dependencies: ReadProperty Service Initiation Tests, 8.15, ReadProperty Service Execution Tests, 9.15, Locating Routers, 10.5.1

BACnet Reference Clause: 6.5.3

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

Dependencies: ReadProperty Service Initiation Tests, 8.15, ReadProperty Service Execution Tests, 9.15

BACnet Reference Clause: 6.5.3

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.

BACnet Reference Clause: 6.5.3.

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 = | Simple-ACK   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.



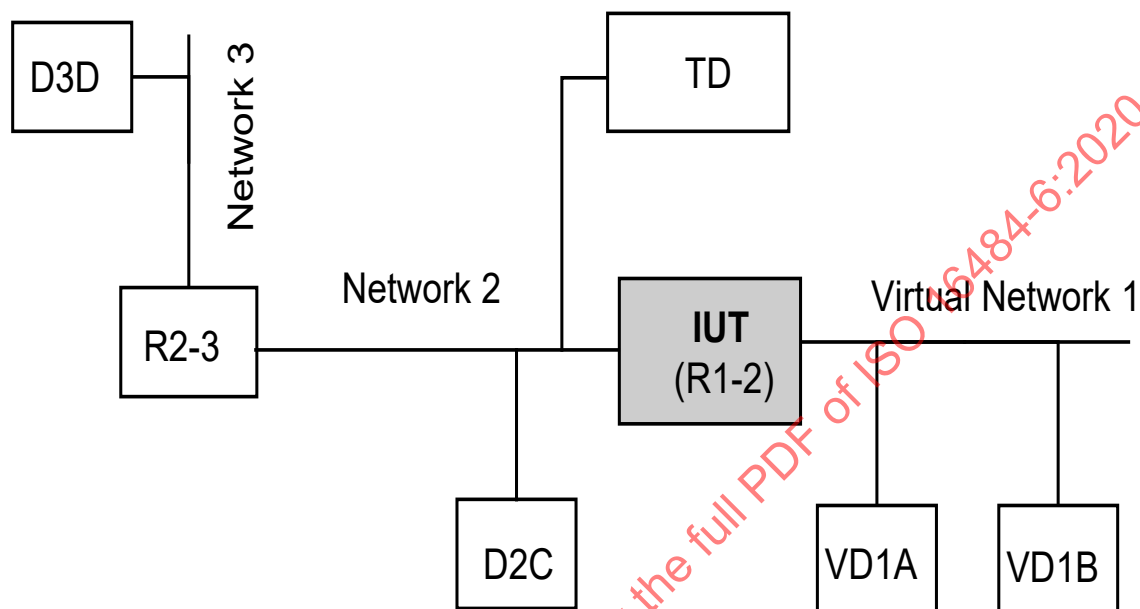


**Figure 10-4.** Logical internetwork configuration for virtual routing functionality tests.

#### 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.

Test Steps:

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

DA = TD,  
 SA = IUT,  
 SNET = 1,  
 SADR = VD1A,  
 ReadProperty-ACK,  
 'Object Identifier' = (the object identifier used in step 3),  
 'Property Identifier' = (the property identifier used in step 3),  
 'Property Value' = (the contents of the specified property)

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

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

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

Test Steps:

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

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

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

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

Test Steps:

1. TRANSMIT
  - DA = IUT,
  - SA = R2-3,
  - DNET = 1,
  - DADR = VD1A,
  - SNET = 3,
  - SADR = D3D,
  - Hop Count = 254,
  - ReadProperty-Request,
  - 'Object Identifier' = (the object identifier of any object in the target device),
  - 'Property Identifier' = (any property of the specified object containing a value small enough so that the response will not need to be segmented)
2. RECEIVE
  - DA = R2-3,
  - SA = IUT,
  - DNET = 3,
  - DADR = D3D,
  - SNET = 1,
  - SADR = VD1A,
  - Hop Count = (any integer x: 1 < x < 255),
  - ReadProperty-Request,

'Object Identifier' = (the object identifier used in step 1),  
'Property Identifier' = (the property identifier used in step 1),  
'Property Value' = (the contents of the specified property)

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

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

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

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

Test Steps:

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

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

#### 10.8.3.5 Unicast Messages that Should Not Be Routed

##### 10.8.3.5.1 Unknown Network

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

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

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 ReadProperty request on Network 2)

##### 10.8.3.5.2 Network Reachable Through the Same Port

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

Test Steps:

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

#### **10.8.4 Routing of Broadcast APDUs to Virtual Devices**

##### **10.8.4.1 Broadcasts that Should Be Ignored**

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

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

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

##### **A. Who-Is Option**

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

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' =

'Device Instance Range High Limit' =

'Object Identifier' =

(the Device object instance number of the virtual device),

(the Device object instance number of the virtual device),

(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' =

'Object Identifier' =

'Object Name' =

( 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' =

'Object Identifier' =

'Object Name' =

(the Device object instance number of the virtual device),

(the object identifier specified in step 1),

(any valid value)

(the Device object instance number of the virtual device),

(the object identifier specified in step 1),

(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.

**Test Steps:**

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

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

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

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

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

**Test Steps:**

1. MAKE (the virtual device generate a remote broadcast message to the network 3)
2. RECEIVE  
DA = R2-3,  
SA = IUT,  
DNET = 3,  
DLEN = 0,  
SNET = 1,

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

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

### 10.8.5 Hop Count Protection

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

BACnet Reference Clause: 6.5.4.

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

Test Steps:

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

### 10.8.6 Network Layer Priority

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

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

### **10.8.7 Multiple Devices on a Single Virtual Network**

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

#### **10.8.7.1 Who-Is Specifying Different Device Identifier**

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

Test Steps:

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

#### **10.8.7.2 Who-Has Specifying Different Device Identifier**

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

Test Steps:

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

#### **10.8.7.3 Read of Object Not Contained by Virtual Device**

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

Test Steps:

1. TRANSMIT  
DA = IUT,  
SA = TD,  
DNET = 1,  
DADR = VD1A,  
Hop Count = 255,  
ReadProperty-Request,  
'Object Identifier' = (Device object identifier of VD1B),  
'Property Identifier' = Object\_Identifier
2. RECEIVE  
DA = TD,

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

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

## 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

Dependencies: None.

BACnet Reference Clauses: 7.1 and 8.1.

ISO/IEC 8802-2 Reference Clauses: 3.3, 4.1, 5.4.1.1, and 5.4.1.2.

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:

1. TRANSMIT ReadProperty-Request,  
 DSAP = BACnet\_Individual,  
 SSAP = BACnet\_Command,  
 CTL = X'03',  
 'Object Identifier' = (the device's Device object),  
 'Property Identifier' = Object\_Identifier
2. 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

Dependencies: None.

BACnet Reference Clauses: 7.1 and 8.1.

ISO/IEC 8802-2 Reference Clauses: 3.3, 4.1, 4.2.1, 5.3.1, 5.4.1.1, 5.4.1.2, and 6.6.

Purpose: To verify that the LLC correctly responds to Exchange Identification (XID) requests.

Test Steps:

1. 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

Dependencies: None.

BACnet Reference Clauses: 7.1 and 8.1.

ISO/IEC 8802-2 Reference Clauses: 3.3, 4.1, 5.3.1, 5.4.1.1, 5.4.1.2, and 6.7.

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,
 }
 }
 }
 }

}

STANDARDSISO.COM : Click to view the full PDF of ISO 16484-6:2020



## 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                                                     |

Dependencies: None.

BACnet Reference Clauses: 9.3.2, 9.5.5, and 9.5.6.

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                                                     |

BACnet Reference Clauses: 9.3.4, 9.3.5, 9.5.4, and 9.5.6.

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.

Dependencies: None.

##### 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.

BACnet Reference Clause: 9.5.4.

##### 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

BACnet Reference Clause: 9.5.1.3.

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 ( $T_{\text{reply\_delay}}$ ) {  
 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

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 (Treply_delay) {
 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

BACnet Reference Clause: 9.5.1.3.

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<sub>reply\_delay</sub>) {  
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<sub>reply\_delay</sub>) {  
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

BACnet Reference Clause: 9.5.4.4.

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<sub>reply\_delay</sub>) {  
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

BACnet Reference Clause: 9.5.4.2.

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

BACnet Reference Clause: 9.5.4.4.

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)                                 |

BACnet Reference Clauses: 9.3.1, 9.3.3, and 9.5.6.5.

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) |
|------------------|---------------|

BACnet Reference Clause: 9.5.6.5, 9.5.6.8.

Dependencies: None.

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)  |

BACnet Reference Clause: 9.5.6.6.

Dependencies: None.

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)

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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) |
|------------------|---------------|

BACnet Reference Clauses: 9.5.6.3 and 9.5.6.5.

Dependencies: None.

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)    |

BACnet Reference Clauses: 9.3.8 and 9.5.6.5.

Dependencies: None.

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

```

#### 12.1.1.4.7 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)

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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  $T_{usage\_timeout}$
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

BACnet Reference Clause: 9.5.6.5.

Dependencies: None.

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 **T<sub>usage\_timeout</sub>**
    - 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 **T<sub>usage\_timeout</sub>**
- 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

```

BACnet Reference Clause: 9.5.6.6.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.7.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.7, 9.5.6.8.

Dependencies: None.

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.

BACnet Reference Clause: 9.5.6.7.

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 ( $T_{no\_token} + T_{slot}$ )
5. TRANSMIT
  - DA = 127,
  - SA = 1,
  - Poll For Master
6. BEFORE ( $T_{no\_token} + T_{slot}$ ) {
  - 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                    |

BACnet Reference Clause: 9.5.6.9.

Dependencies: None.

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           |

BACnet Reference Clause: 9.5.6.9.

Dependencies: Answer Data Request, 12.1.1.5.1.

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

```

BACnet Reference Clause: 9.5.6.2.

Dependencies: None.

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<sub>reply\_delay</sub>) {
 

```

 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

```

BACnet Reference Clause: 9.5.6.2.

Dependencies: None.

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<sub>reply\_delay</sub>) {  
    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          |

BACnet Reference Clause: 9.5.6.2.

Dependencies: None.

#### 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 (Treply_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 |

BACnet Reference Clause: 9.5.6.7.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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)

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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
 }

```

#### 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 |

BACnet Reference Clause: 9.5.6.5.

Dependencies: None.

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
    
```

12.1.1.7.5 New Master Enters

Purpose: To verify that the IUT recognizes the presence of another master.

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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<sub>usage\_timeout</sub>) {  
        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<sub>usage\_timeout</sub>) {  
    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<sub>usage\_timeout</sub>) {  
    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) |

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.8.

Dependencies: None.

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

BACnet Reference Clause: 9.5.6.8

Dependencies: None.

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
  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.

BACnet Reference Clause: 9.5.6.4.

Dependencies: None.

##### 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<sub>reply\_timeout</sub>

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  $T_{reply\_timeout}$
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  $T_{reply\_timeout}$
  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.

BACnet Reference Clause: 9.5.6.

Dependencies: None.

##### 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          |

Dependencies: None.

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 |

Dependencies: None.

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 |

Dependencies: None.

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 |

Dependencies: None.

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

```

Dependencies: None.

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

Dependencies: None

BACnet Reference Clauses: 9.5.3 and 9.5.6.5

**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.

Dependencies: None.

BACnet Reference Clauses: 9.3.6, 9.5.5, 9.5.7 and 9.5.4.

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

Dependencies: None.

BACnet Reference Clause: 9.5.7.2.

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<sub>reply\_delay</sub>)  
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)

Dependencies: Confirmed Service Request Transitions, 12.1.2.1.1.

BACnet Reference Clause: 9.5.4.4.

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<sub>reply\_delay</sub>)  
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.

Dependencies: None.

##### 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

BACnet Reference Clauses: 9.5.4 and 9.5.7.2.

##### 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

BACnet Reference Clause: 9.5.1.3.

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

BACnet Reference Clause: 9.5.1.3.

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  $T_{\text{reply\_delay}}$
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  $T_{\text{reply\_delay}}$



### 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\_Type
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

BACnet Reference Clause: 9.5.4.4.

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)

BACnet Reference Clause: 9.5.7.2.

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 (Treply_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 (Treply_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)

BACnet Reference Clauses: 9.1.3 and 9.5.7.2.

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 (Treply_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 (Treply_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.

BACnet Reference Clause: 9.5.7.2.

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<sub>reply\_delay</sub>)  
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<sub>reply\_delay</sub>) {  
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<sub>reply\_delay</sub>)  
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

BACnet Reference Clause: 9.5.4.2.

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<sub>reply\_delay</sub>)  
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

BACnet Reference Clause: 9.3.4.

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 (Treply_delay)
 RECEIVE
 Test_Response,
 'Length' = 0

```

Notes to Tester: In the response in step 4, there shall be no inter-frame gaps greater than  $T_{\text{frame\_gap}}$ .

#### 12.1.2.2.3 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  $T_{\text{frame\_gap}}$ .

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 (Treply_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.

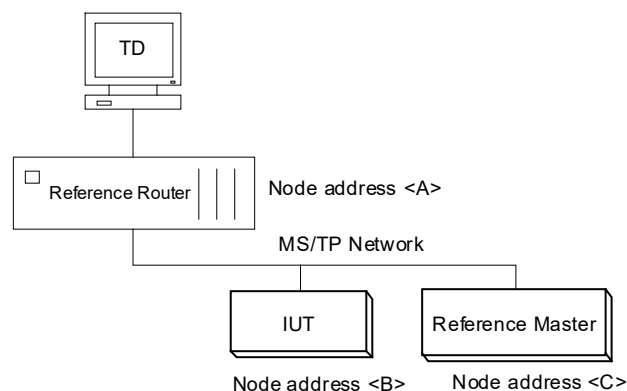


Figure 12-1. General MS/TP Test Network Setup.

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  $\pm 1$  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:

$T_{\text{pos\_err}} = 2$  milliseconds

$T_{\text{neg\_err}} = 1$  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.

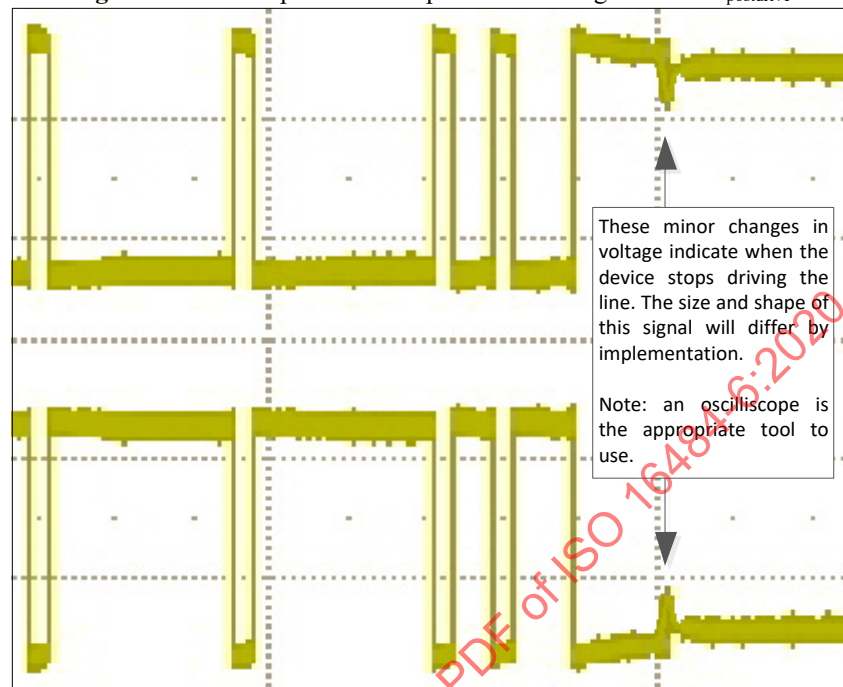
Configuration Requirements: Connect an oscilloscope to the MS/TP network.

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 data octets when transmitting a frame is 20 bit times or less.

Test Steps:

1. Elicit the transmission of any frame from the IUT.
2. Measure the longest EIA-485 idle time that appears between octets within the data frame transmitted by the IUT. If there is no idle time between octets, pass the IUT.
3. Fail the IUT if the time measured in step 2 is greater than the time intervals shown below for each baud rate.

|              |                                                     |
|--------------|-----------------------------------------------------|
| 9600 baud:   | fail if interval is greater than 2,083 microseconds |
| 19200 baud:  | fail if interval is greater than 1,042 microseconds |
| 38400 baud:  | fail if interval is greater than 521 microseconds   |
| 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