INTERNATIONAL STANDARD

ISO 16484-6

Second edition
2009-03-15

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

Reference number
ISO 16484-6:2009(E)

Contents

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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.

ISO 16484-6 was prepared by Technical Committee ISO/TC 205, *Building environment design*.

This second edition cancels and replaces the first edition (ISO 16484-6:2005), of which it constitutes a minor revision.

ISO 16484 consists of the following parts, under the general title *Building automation and control systems (BACS)*:

— *Part 2: Hardware*

— *Part 3: Functions*

— *Part 5: Data communication protocol*

— *Part 6: Data communication conformance testing*

A Part 1, dealing with project implementation, and a Part 4, dealing with applications, are under development.

Building automation and control systems (BACS) —

Part 6: Data communication conformance testing

1 Scope

This part of ISO 16484 defines 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.

This part of ISO 16484 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.

2 Relationship between this part of ISO 16484 and ANSI/ASHRE 135.1-2007

This part of ISO 16484 comprises, from Clause 4 onwards, the US standard ANSI/ASHRE 135.1-2007, *Method of Test for Conformance to BACnet*, published by the American National Standards Institute and the American Society of Heating, Refrigerating and Air-Conditioning Engineers.

3 Terms, definitions and abbreviated terms

For the purposes of this document, the following terms, definitions and abbreviated terms apply.

3.1
local network
network to which a BACnet device is directly connected

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

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

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 parenthsis, 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:

DS-RP-A	DS-RP-B	
DS-RPM-A	DS-RPM-B	
DS-RPC-A	DS-RPC-B	
DS-WP-A	DS-WP-B	
DS-WPM-A	DS-WPM-B	
DS-COV-A	DS-COV-B	
DS-COVP-A	DS-COVP-B	
DS-COVU-A	DS-COVU-B	
AE-N-A	AE-N-I-B	AE-N-E-B
AE-ACK-A	AE-ACK-B	
AE-ASUM-A	AE-ASUM-B	
AE-ESUM-A	AE-ESUM-B	
AE-INFO-A	AE-INFO-B	
AE-LS-A	AE-LS-B	
SCHED-A	SCHED-I-B	SCHED-E-B
T-VMT-A	T-VMT-I-B	T-VMT-E-B
T-ATR-A	T-ATR-B	
DM-DDB-A	DM-DDB-B	
DM-DOB-A	DM-DOB-B	
DM-DCC-A	DM-DCC-B	
DM-PT-A	DM-PT-B	

DM-TM-A	DM-TM-B
DM-TS-A	DM-TS-B
DM-UTC-A	DM-UTC-B
DM-RD-A	DM-RD-B
DM-BR-A	DM-BR-B
DM-R-A	DM-R-B
DM-LM-A	DM-LM-B
DM-OCD-A	DM-OCD-B
DM-VT-A	DM-VT-B
NM-CE-A	NM-CE-B
NM-RC-A	NM-RC-B

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:

AcknowledgeAlarm	RemoveListElement	ConfirmedTextMessage
ConfirmedCOVNotification	CreateObject	UnconfirmedTextMessage
UnconfirmedCOVNotification	DeleteObject	TimeSynchronization
ConfirmedEventNotification	ReadProperty	UTCTimeSynchronization
UnconfirmedEventNotification	ReadPropertyConditional	Who-Has
GetAlarmSummary	ReadPropertyMultiple	I-Have
GetEnrollmentSummary	ReadRange	Who-Is
GetEventInformation	WriteProperty	I-Am
LifeSafetyOperation	WritePropertyMultiple	VT-Open
SubscribeCOV	DeviceCommunicationControl	VT-Close
SubscribeCOVProperty	ConfirmedPrivateTransfer	VT-Data
AtomicReadFile	UnconfirmedPrivateTransfer	RequestKey
AtomicWriteFile	ReinitializeDevice	Authenticate
AddListElement		

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:

Access Door	Binary Output	Group	Multi-state Value
Accumulator	Binary Value	Life Safety Point	Notification Class

Analog Input	Calendar	Life Safety Zone	Program
Analog Output	Command	Load Control	Pulse Converter
Analog Value	Device	Loop	Schedule
Averaging	Event Enrollment	Multi-state Input	Structured View
Binary Input	File	Multi-state Output	Trend Log

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: ❑↵
}↵
```

4.5.10 Test Database

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

```
List of Objects in Test Device: ↵
{↵
object1↵
object2↵
...
objectN↵
}↵
```

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

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

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

```
    {
        object-identifier: (proprietary ❑, instance)
        object-name: "❑"
        object-type: proprietary ❑
    }
```

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

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

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

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

The following sections show templates for each of the standard object types. To improve readability the carriage return/linefeed pairs are not explicitly shown in the examples.

4.5.10.1 Accumulator

```
{
   object-identifier: (accumulator, ❑)
   object-name: "❑"
   object-type: accumulator
   present-value: ❑
   description: "❑"
   device-type: "❑"
   status-flags: ❑
   event-state: ❑
   reliability: ❑
   out-of-service: ❑
   scale: ❑
   units: ❑
   prescale: ❑
   max-pres-value: ❑
   value-change-time: ❑
   value-before-change: ❑
   value-set: ❑
   logging-record: ❑
   logging-object: ❑
   pulse-rate: ❑
   high-limit: ❑
   low-limit: ❑
   limit-monitoring-interval: ❑
   notification-class: ❑
   time-delay: ❑
   limit-enable: ❑
   event-enable: {❑,❑,❑}
   acked-transitions: {❑,❑,❑}
   notify-type: ❑
   event-time-stamps: {❑,❑,❑}
   profile-name: "❑"
 }
```

4.5.10.2 Analog Input

```
{
   object-identifier: (analog-input, ❑)
   object-name: "❑"
   object-type: analog-input
   present-value: ?
   description: "❑"
   device-type: "❑"
   status-flags: {❑,❑,❑,❑}
   event-state: ❑
   reliability: ❑
   out-of-service: ❑
   update-interval: ❑
   units: ❑
   min-pres-value: ❑
   max-pres-value: ❑
   resolution: ❑
   cov-increment: ❑
   time-delay: ❑
   notification-class: ❑
   high-limit: ❑
   low-limit: ❑
   deadband: ❑
   limit-enable: {❑,❑}
   event-enable: {❑,❑,❑}
   acked-transitions: {❑,❑,❑}
   notify-type: ❑
   event-time-stamps: {❑,❑,❑}
   profile-name: "❑"
 }
```

4.5.10.3 Analog Output

```
 {
   object-identifier: (analog-output, ❑)
   object-name: "❑"
   object-type: analog-output
   present-value: ?
   description: "❑"
   device-type: "❑"
   status-flags: {❑,❑,❑,❑}
   event-state: ❑
   reliability: ❑
   out-of-service: ❑
   units: ❑
   min-pres-value: ❑
   max-pres-value: ❑
   resolution: ❑
   priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
```

```
  relinquish-default: ❑
  cov-increment: ❑
  time-delay: ❑
  notification-class: ❑
  high-limit: ❑
  low-limit: ❑
  deadband: ❑
  limit-enable: {❑,❑}
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.4 Analog Value

```
 {
  object-identifier: (analog-value, ❑)
  object-name: "❑"
  object-type: analog-value
  present-value: ?
  description: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  units: ❑
  priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
  relinquish-default: ❑
  cov-increment: ❑
  time-delay: ❑
  notification-class: ❑
  high-limit: ❑
  low-limit: ❑
  deadband: ❑
  limit-enable: {❑,❑}
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.5 Averaging

```
 {
  object-identifier: (averaging, ❑)
  object-name: "❑"
  object-type: averaging
  minimum-value: ❑
  minimum-value-timestamp: {❑,❑}
  average-value: ❑
  variance-value: ❑
  maximum-value: ❑
  maximum-value-timestamp: {❑,❑}
  description: "❑"
  attempted-samples: ❑
  valid-samples: ❑
  object-property-reference: ❑
  window-interval: ❑
  window-samples: ❑
  profile-name: "❑"
 }
```

4.5.10.6 Binary Input

```
 {
  object-identifier: (binary-input, ❑)
  object-name: "❑"
  object-type: binary-input
  present-value: ?
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  polarity: ❑
  inactive-text: "❑"
  active-text: "❑"
  change-of-state-time: {❑,❑}
  change-of-state-count: ❑
  time-of-state-count-reset: {❑,❑}
  elapsed-active-time: ❑
  time-of-active-time-reset: {❑,❑}
  time-delay: ❑
  notification-class: ❑
  alarm-value: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.7 Binary Output

```
 {
  object-identifier: (binary-output, ❑)
  object-name: "❑"
  object-type: binary-output
  present-value: ?
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  polarity: ❑
  inactive-text: "❑"
  active-text: "❑"
  change-of-state-time: {❑,❑}
  change-of-state-count: ❑
  time-of-state-count-reset: {❑,❑}
  elapsed-active-time: ❑
  time-of-active-time-reset: {❑,❑}
  minimum-off-time: ❑
  minimum-on-time: ❑
  priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
  relinquish-default: ❑
  time-delay: ❑
  notification-class: ❑
  feedback-value: ?
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.8 Binary Value

```
 {
  object-identifier: (binary-value, ❑)
  object-name: "❑"
  object-type: binary-value
  present-value: ?
  description: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  inactive-text: "❑"
  active-text: "❑"
  change-of-state-time: {❑,❑}
  change-of-state-count: ❑
  time-of-state-count-reset: {❑,❑}
  elapsed-active-time: ❑
  time-of-active-time-reset: {❑,❑}
  minimum-off-time: ❑
  minimum-on-time: ❑
  priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
  relinquish-default: ❑
  time-delay: ❑
  notification-class: ❑
  alarm-value: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
}
```

4.5.10.9 Calendar

```
 {
  object-identifier: (calendar, ❑)
  object-name: "❑"
  object-type: calendar
  description: "❑"
  present-value: ❑
  date-list: (❑,❑…)
  profile-name: "❑"
 }
```

4.5.10.10 Command

```
 {
  object-identifier: (command, ❑)
  object-name: "❑"
  object-type: command
  description: "❑"
  present-value: ❑
  in-process: ❑
  all-writes-successful: ❑
  action: {❑,❑…}
  action-text: {"❑","❑"...}
  profile-name: "❑"
 }
```

4.5.10.11 Device

```
 {
  object-identifier: (device, ❑)
  object-name: "❑"
  object-type: device
  system-status: ❑
  vendor-name: "❑"
  vendor-identifier: ❑
  model-name: "❑"
  firmware-revision: "❑"
  application-software-version: "❑"
  location: "❑"
  description: "❑"
  protocol-version: ❑
  protocol-revision: ❑
  protocol-services-supported: {❑,❑...}
  protocol-object-types-supported: {❑,❑...}
  object-list: {❑, ❑...}
  max-APDU-length-accepted: ❑
  segmentation-supported: ❑
  vt-classes-supported: (❑, ❑...)
  active-vt-sessions: (❑, ❑...)
  local-time: ❑
  local-date: ❑
  utc-offset: ❑
  daylight-savings-status: ❑
  apdu-segment-timeout: ❑
  apdu-timeout: ❑
  number-of-APDU-retries: ❑
  list-of-session-keys: (❑, ❑...)
  time-synchronization-recipients: (❑, ❑...)
  max-master: ❑
  max-info-frames: ❑
  device-address-binding: (❑, ❑...)
  database-revision: ❑
  configuration-files: ❑
  last-restore-time: ❑
  backup-failure-timeout: ❑
  active-cov-subscriptions: ❑
  profile-name: "❑"
 }
```

4.5.10.12 Event Enrollment

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

4.5.10.13 File

```
 {
  object-identifier: (file, ❑)
  object-name: "❑"
  object-type: file
  description: "❑"
  file-type: "❑"
  file-size: ❑
  modification-date: {❑,❑}
  archive: ❑
  read-only: ❑
  file-access-method: ❑
  profile-name: "❑"
 }
```

4.5.10.14 Group

```
 {
  object-identifier: (group, ❑)
  object-name: "❑"
  object-type: group
  description: "❑"
  list-of-group-members: (❑, ❑...)
  present-value: (❑, ❑...)
  profile-name: "❑"
 }
```

4.5.10.15 Life Safety Point

```
 {
  object-identifier: (life-safety-point, ❑)
  object-name: "❑"
  object-type: life-safety-point
  present-value: ?
  tracking-value: ❑
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability:❑
  out-of-service: ❑
  mode: ❑
  time-delay: ❑
  notification-class: ❑
  life-safety-alarm-values: (❑,❑...)
  alarm-values: (❑,❑...)
  fault-values: (❑,❑...)
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  silenced: ❑
  operation-expected: ❑
  maintenance-required: ❑
  setting: ❑
  direct-reading❑
  units: ❑
  member-of: (❑,❑...)
  profile-name: "❑"
}
```

4.5.10.16 Life Safety Zone

```
{
  object-identifier: (life-safety-zone, ❑)
  object-name: "❑"
  object-type: life-safety-zone
  present-value: ?
  tracking-value: ❑
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  mode: ❑
  time-delay: ❑
  notification-class: ❑
  life-safety-alarm-values: (❑,❑...)
  alarm-values: (❑,❑...)
  fault-values: (❑,❑...)
  event-enable: (❑,❑,❑)
  acked-transitions: (❑,❑,❑)
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  silenced: ❑
  operation-expected: ❑
  maintenance-required: ❑
  zone-members: (❑,❑...)
  member-of: (❑,❑...)
  profile-name: "❑"
}
```

4.5.10.17 Loop

```
{
  object-identifier: (loop, ❑)
  object-name: "❑"
  object-type: loop
  present-value: ?
  description: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  update-interval: ❑
  output-units: ❑
  manipulated-variable-reference: {❑}
  controlled-variable-reference: {❑}
  controlled-variable-value ❑
  controlled-variable-units ❑
  setpoint-reference: {❑}
  setpoint: ❑
  action: ❑
  proportional-constant: ❑
  proportional-constant-units: ❑
  integral-constant: ❑
  integral-constant-units: ❑
  derivative-constant: ❑
  derivative-constant-units: ❑
  bias: ❑
  maximum-output: ❑
  minimum-output: ❑
  priority-for-writing: ❑
  cov-increment: ❑
  time-delay: ❑
  notification-class: ❑
  error-limit: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.18 Multi-state Input

```
 {
  object-identifier: (multi-state-input, ❑)
  object-name: "❑"
  object-type: multi-state-input
  present-value: ?
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  number-of-states: ❑
  state-text: {"❑", "❑"...}
  time-delay: ❑
  notification-class: ❑
  alarm-values: (❑,❑…)
  fault-values: (❑,❑…)
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.19 Multi-state Output

```
{
  object-identifier: (multi-state-output, ❑)
  object-name: "❑"
  object-type: multi-state-output
  present-value: ?
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  number-of-states: ❑
  state-text: {"❑", "❑"...}
  priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
  relinquish-default: ❑
  time-delay: ❑
  notification-class: ❑
  feedback-value: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.20 Multi-state Value

```
{
  object-identifier: (multi-state-value, ❑)
  object-name: "❑"
  object-type: multi-state-value
  present-value: ?
  description: "❑"
  device-type: "❑"
  status-flags: {❑,❑,❑,❑}
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  number-of-states: ❑
  state-text: {"❑", "❑"...}
  priority-array: {❑,❑,?,?,❑,?,?,❑,?,?,?,?,?,?,?,?}
  relinquish-default: ❑
  time-delay: ❑
  notification-class: ❑
  alarm-values: (❑,❑...)
  fault-values: (❑,❑...)
  feedback-value: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
 }
```

4.5.10.21 Notification Class

```
 {
  object-identifier: (notification-class, ❑)
  object-name: "❑"
  object-type: notification-class
  description: "❑"
  notification-class: ❑
  priority: {❑, ❑, ❑}
  ack-required: {❑,❑,❑}
  recipient-list: (❑,❑…)
  profile-name: "❑"
 }
```

4.5.10.22 Program

```
{
  object-identifier: (program, ❑)
  object-name: "❑"
  object-type: program
  program-state: ❑
  program-change; ❑
  reason-for-halt: ❑
  description-of-halt: "❑"
  program-location: "❑"
  description: "❑"
  instance-of: "❑"
  status-flags: {❑,❑,❑,❑}
  reliability: ❑
  out-of-service: ❑
  profile-name: "❑"
}
```

4.5.10.23 Pulse Converter

```
{
  object-identifier: (pulse-converter, ❑)
  object-name: "❑"
  object-type: pulse-converter
  description: "❑"
  present-value: ❑
  input-reference: ❑
  status-flags: ❑
  event-state: ❑
  reliability: ❑
  out-of-service: ❑
  units: ❑
  scale-factor: ❑
  adjust-value: ❑
  count: ❑
  update-time: ❑
  count-change-time: ❑
  count-before-change: ❑
  cov-increment: ❑
  cov-period: ❑
  notification-class: ❑
  time-delay: ❑
  high-limit: ❑
  low-limit: ❑
  deadband: ❑
  limit-enable: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
}
```

4.5.10.24 Schedule

```
{
  object-identifier: (schedule, ❑)
  object-name: "❑"
  object-type: schedule
  present-value: ❑
  description: "❑"
  effective-period: {❑,❑}
  weekly-schedule: {❑,❑...}
  exception-schedule: {❑,❑...}
  list-of-object-property-references: (❑,❑…)
  priority-for-writing ❑
  profile-name: "❑"
}
```

4.5.10.25 Trend Log

```
{
   object-identifier: (trend-log, ❑)
   object-name: "❑"
   object-type: trend-log
   description: "❑"
   log-enable: ❑
   start-time: ❑
   stop-time: ❑
   log-device-object-property: {❑}
   log-interval: ❑
   cov-resubscription-interval : ❑
   client-cov-increment: ❑
   stop-when-full: ❑
   buffer-size: ❑
   log-buffer: ❑
   record-count: ❑
   total-record-count: ❑
   notification-threshold: ❑
   records-since-notification: ❑
   previous-notify-time: ❑
   current-notify-time: ❑
   event-state: ❑
   notification-class: ❑
   event-enable: {❑,❑,❑}
   acked-transitions: {❑,❑,❑}
   notify-type: ❑
   event-time-stamps: {❑,❑,❑}
   profile-name: "❑"
 }
```

4.5.10.26 Access Door

```
 {
   object-identifier: (access-door, ❑)
   object-name: "❑"
   object-type: access-door
   present-value      : ❑
   description: "❑"
   status-flags: ❑
   event-state: ❑
   reliability: ❑
   out-of-service: ❑
   priority-array: ❑
   relinquish-default: ❑
   door-status: ❑
   lock-status: ❑
   secured-status: ❑
   door-members: {❑, ❑...}
   door-pulse-time: ❑
   door-extended-pulse-time: ❑
   door-unlock-delay-time: ❑
   door-open-too-long-time: ❑
   door-alarm-state: ❑
   masked-alarm-values: {❑, ❑ ...}
   maintenance-required: ❑
   time-delay: ❑
   notification-class: ❑
   alarm-values: {❑, ❑...}
   fault-values: {❑, ❑...}
```

```
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
}
```

4.5.10.27 Load Control

```
{
  object-identifier: (load-control, ❑)
  object-name: "❑"
  object-type: load-control
  description: "❑"
  present-value: ❑
  state-description: "❑"
  status-flags: ❑
  event-state: ❑
  reliability: ❑
  requested-shed-level: ❑
  start-time: ❑
  shed-duration: ❑
  duty-window: ❑
  enabled: ❑
  full-duty-baseline: ❑
  expected-shed-level: ❑
  actual-shed-level: ❑
  shed-levels: {❑, ❑...}
  shed-level-descriptions: {"❑", "❑"...}
  notification-class: ❑
  time-delay: ❑
  event-enable: {❑,❑,❑}
  acked-transitions: {❑,❑,❑}
  notify-type: ❑
  event-time-stamps: {❑,❑,❑}
  profile-name: "❑"
}
```

4.5.10.28 Structured View

```
{
  object-identifier: (structured-view, ❑)
  object-name: "❑"
  object-type: structured-view
  description: "❑"
  node-type: ❑
  node-subtype: "❑"
  subordinate-list: {❑, ❑...}
  subordinate-annotations: {"❑", "❑"...}
  profile-name: "❑"
}
```

5 EPICS CONSISTENCY TESTS

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 Object_Types_Supported property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.

(d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.

(e) The Application_Services_Supported 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 Object_Types_Supported property of the Device object contained in the test database.

(g) For each object type listed in the Standard Object Types Supported section of the EPICS there shall be at least one object of that type in the test database.

(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. 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, that property shall be marked as 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.

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> <b> <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)
```

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>
              | <say statement> | <check statement> | <make statement>
              | <transmit statement> | <receive statement>
              | <write statement> | <verify statement> | <before 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 value> ::=         TD | IUT
<dst> ::=                    DESTINATION '=' <dst parm value>
<dst parm value> ::=         LOCAL BROADCAST | GLOBAL BROADCAST
                                  | REMOTE BROADCAST <net>
                                  | IUT | TD
<net> ::=                    (a valid BACnet network number)

<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 DESTINATION and SOURCE are not specified, the source address shall be TD and 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 defaults to IUT and DESTINATION defaults to TD.

```
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' =                          0
         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' =                          0,
                   '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 135-2004, 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>,
     '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> '=' <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' =  <property value> 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 135-1995, 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:

BEFORE **Acknowledgment Fail Time** RECEIVE BACnet-Simple-ACK

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
    }
```

6.3 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.

6.4 BACnet References

All references to BACnet clauses in this standard refer to ANSI/ASHRAE 135-2004, except as otherwise noted ("BACnet-2001" refers to ANSI/ASHRAE 135-2001).

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

Dependencies: ReadProperty Service Execution Tests, 9.18.

Purpose: To verify that all properties of all objects can be read using BACnet ReadProperty and ReadPropertyMultiple services. The test is performed once using ReadProperty and once using ReadPropertyMultiple. When verifying array properties, the whole array shall be read without using an array index, where possible.

Test Steps:

```
1.  REPEAT X = (all objects in the IUT's database) DO {
        REPEAT Y = (all properties in object X) DO {
         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.

7.2 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 each defined enumeration shall be explicitly 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 length zero, a string with the maximum supported length, and a string with some length between the two. The vendor shall provide the actual value of the maximum length string in the EPICS. See 4.4.2.

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 WritePropertyMulitiple services. The test is performed once using WriteProperty and once using WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.

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

Test Steps:

```
1.  REPEAT X = (all objects in the IUT's database) DO {
        REPEAT Y = (all writable properties in object X) DO {
            REPEAT Z = (all values meeting the functional range requirements of 7.2.1) DO {
                WRITE (X), Y = Z,
                VERIFY (X), Y = Z
                }
            }
        }
```

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.

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.  WRITE Present_Value = (any value that corresponds to an Event_State of NORMAL)
6.  IF (Reliability is writable) THEN
        REPEAT X = (all values of the Reliability enumeration appropriate to the object type except
                    NO_FAULT_DETECTED) DO {
            WRITE Reliability = X
            VERIFY Reliability = X
            VERIFY Status_Flags = (?, TRUE, ?, TRUE)
            WRITE Reliability = NO_FAULT_DETECTED
            VERIFY Reliability = NO_FAULT_DETECTED
            VERIFY Status_Flags = (?, FALSE, ?, TRUE)
            }
7.  IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service = FALSE
    ELSE
        MAKE (Out_Of_Service FALSE)
```

Notes to Tester: If the object being tested is commandable and there is an internal process writing to the Present_Value property 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 Analog Output, Analog Value, Binary Output, Binary Value, Multi-state Output, and Multi-state Value objects that are commandable.

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 binary datatypes V_{low} and V_{high} shall be the same, and V_{med} shall be different. The TD also selects three priorities P_{low}, P_{med}, and P_{high}, all between 1 and 5, such that numerically $P_{low} > P_{med} > P_{high}$. The selected values are written one at a time to Present_Value at the corresponding priority. The Present_Value and Priority_Array are checked to verify correct operation. Priorities numerically smaller than 6 (higher priority) are used to eliminate minimum on/off time considerations

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

Test Steps:

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

7.3.1.4 Minimum_Off_Time

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) that 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) that 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: 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) that 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 7.

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' =     (the current local time, $T_{LOC}$)
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' =     (the current local time, $T_{LOC}$)
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: $(T_{ELAPSED} + 30) \leq T \leq (T_{ELAPSED} + 30 + \textbf{Internal Processing Fail Time})$)
11. IF (Present_Value is writable) THEN
        WRITE Present_Value = INACTIVE
        VERIFY Present_Value = INACTIVE
    ELSE
        MAKE (Present_Value = INACTIVE)
12. IF (Elapsed_Active_Time is writable) THEN
        WRITE Elapsed_Active_Time = 0
        VERIFY Elapsed_Active_Time  = 0
    ELSE
        MAKE (Elapsed_Active_Time = 0)
13. TRANSMIT ReadProperty-Request,
        'Object Identifier' =   (the IUT's Device object),
        'Property Identifier' =     Local_Date
14. RECEIVE ReadProperty-ACK,
        'Object Identifier' =   (the IUT's Device object),
        'Property Identifier' =     Local Date,
        'Property Value' =      (the current local date, D)
15. TRANSMIT ReadProperty-Request,
        'Object Identifier' =   (the IUT's Device object),
        'Property Identifier' =     Local_Time
16. RECEIVE ReadProperty-ACK,
        'Object Identifier' =   (the IUT's Device object),
        'Property Identifier' =     Local_Time,
        'Property Value' =      (the current local time, $T_{LOC}$)
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

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.27, 12.2.23, 12.3.24, 12.4.20, 12.6.22, 12.7.26, 12.8.24, 12.12.10, 12.15.19, 12.16.19, 12.17.33, 12.18.17, 12.19.18, 12.20.18, 12.23.26, and 12.25.22.

Purpose: To verify that notification messages are transmitted only if the bit in Event_Enable corresponding to the event transition has a value of TRUE. This test applies to Event Enrollment objects and Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Loop, Multi-state Input, Multi-state Output, Multi-state Value, Pulse Converter and Trend Log objects that support intrinsic reporting.

Test Concept: The IUT is configured such that the Event_Enable property indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Event_Enable property has a value of TRUE.

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

In the test description below, "X" is used to designate the event-triggering property.

```
1.  VERIFY Event_State = NORMAL
2.  WAIT (Time_Delay + **Notification Fail Time**)
3.  IF (X is writable) THEN
        WRITE X = (a value that is OFFNORMAL)
    ELSE
        MAKE (X have a value that is OFFNORMAL)
```

```
4.  WAIT (Time_Delay)
5.  BEFORE **Notification Fail Time**
        IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object configured for this test),
                'Time Stamp' =                    (the current local time),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    NORMAL,
                'To State' =                      OFFNORMAL,
                'Event Values' =                  (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
6.  VERIFY Event_State = OFFNORMAL
7.  IF (X is writable) THEN
        WRITE X = (a value that is NORMAL)
    ELSE
        MAKE (X have a value that is NORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE **Notification Fail Time**
        IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object configured for this test),
                'Time Stamp' =                    (the current local time),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    OFFNORMAL,
                'To State' =                      NORMAL,
                'Event Values' =                  (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY Event_State = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
        MAKE (the event-triggering object change to a fault condition)
        WAIT (Time_Delay)
        BEFORE **Notification Fail Time**
            IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN

                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =            (any valid process ID),
                    'Initiating Device Identifier' =  IUT,
                    'Event Object Identifier' =       (the event-generating object configured for this test),
                    'Time Stamp' =                    (the current local time),
                    'Notification Class' =            (the class corresponding to the object being tested),
                    'Priority' =                      (the value configured to correspond to a TO-FAULT transition),
                    'Event Type' =                    (any valid event type),
                    'Notify Type' =                   EVENT | ALARM,
                    'AckRequired' =                   TRUE | FALSE,
                    'From State' =                    NORMAL,
                    'To State' =                      FAULT,
            'Event Values' =                    (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
    VERIFY Event_State = FAULT
    }
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.11 Acked_Transitions Tests

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; AcknowledgeAlarm Service Execution Tests, 9.1; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.28, 12.2.24, 12.3.25, 12.4.21, 12.6.23, 12.7.27, 12.8.25, 12.12.11, 12.15.20, 12.16.20,12.17.34, 12.18.18, 12.19.19, 12.20.19, 12.23.27 and 12.25.23.

Purpose: To verify that the Acked_Transitions property tracks whether or not an acknowledgment has been received for a previously issued event notification. It also verifies the interrelationship between Status_Flags and Event_State. This test applies to Event Enrollment objects and Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Loop, Multi-state Input, Multi-state Output, Multi-state Value, Pulse COnverter and Trend Log objects that support intrinsic reporting.

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

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

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

```
1.  WAIT (Time_Delay + **Notification Fail Time**)
2.  VERIFY Event_State = NORMAL
3.  VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
4.  VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
5.  IF (X is writable) THEN
        WRITE X = (a value that is OFFNORMAL)
    ELSE
        MAKE (X have a value that is OFFNORMAL)
6.  WAIT (Time_Delay)
7.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the event-generating object configured for this test),
            'Time Stamp' =                      (any valid time stamp),
            'Notification Class' =              (the class corresponding to the object being tested),
            'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =                      (any valid event type),
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      NORMAL,
            'To State' =                    OFFNORMAL,
            'Event Values' =                (values appropriate to the event type)
8.  VERIFY Event_State = OFFNORMAL
9.  VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
10. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
11. IF (X is writable) THEN
        WRITE X = (a value that is NORMAL)
    ELSE
        MAKE (X have a value that is NORMAL)
12. WAIT (Time_Delay)
13. BEFORE  **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the event-generating object configured for this test),
            'Time Stamp' =                  (any valid time stamp),
            'Notification Class' =          (the class corresponding to the object being tested),
            'Priority' =                    (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =                  (any valid event type),
            'Notify Type' =                 EVENT | ALARM,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  OFNORMAL,
            'To State' =                    NORMAL,
            'Event Values' =                (values appropriate to the event type)
14. VERIFY Event_State = NORMAL
15. VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
16. VERIFY Status_Flags = (FALSE, FALSE, ?,?)
17. IF (the event-triggering object can be placed into a fault condition) THEN {
18.     MAKE (the event-triggering object change to a fault condition)
19.     WAIT (Time_Delay)
20.     BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
              'Process Identifier' =          (any valid process ID),
              'Initiating Device Identifier' = IUT,
              'Event Object Identifier' =     (the event-generating object configured for this test),
              'Time Stamp' =                  (any valid time stamp),
              'Notification Class' =          (the class corresponding to the object being tested),
              'Priority' =                    (the value configured to correspond to a TO-FAULT transition),
              'Event Type' =                  (any valid event type),
              'Notify Type' =                 EVENT | ALARM,
              'AckRequired' =                 TRUE | FALSE,
              'From State' =                  NORMAL,
              'To State' =                    FAULT,
              'Event Values' =                (values appropriate to the event type)
21.     VERIFY Event_State = FAULT
22.     VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
23.     VERIFY Status_Flags = (FALSE, TRUE, ?, ?)
24.     MAKE (the event-triggering object change to a normal condition)
25.     WAIT (Time_Delay)
26.     BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
              'Process Identifier' =          (any valid process ID),
              'Initiating Device Identifier' = IUT,
              'Event Object Identifier' =     (the event-generating object configured for this test),
              'Time Stamp' =                  (any valid time stamp),
              'Notification Class' =          (the class corresponding to the object being tested),
              'Priority' =                    (the value configured to correspond to a TO-NORMAL transition),
              'Event Type' =                  (any valid event type),
              'Notify Type' =                 EVENT | ALARM,
              'AckRequired' =                 TRUE | FALSE,
                'From State' =                  FAULT,
                'To State' =                    NORMAL,
                'Event Values' =                (values appropriate to the event type)
27.     VERIFY Event_State = NORMAL
28.     VERIFY Acked_Transitions = (FALSE, FALSE, FALSE)
29.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
30.     TRANSMIT AcknowledgeAlarm-Request,
            'Acknowledging Process Identifier' =  (the value of the 'Process Identifier' in step 20),
            'Event Object Identifier' =           (the 'Event Object Identifier' in step 20),
            'Event State Acknowledged' =          FAULT,
            'Time Stamp' =                        (the 'Time Stamp in step 20),
            'Time of Acknowledgment' =            (the current time)
31.     RECEIVE BACnet-SimpleACK-PDU
32. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =             (the value of the 'Process Identifier' in step 20),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        (the 'Event Object Identifier' in step 20),
            'Time Stamp' =                     (the 'Time Stamp' in step 20),
            'Notification Class' =             (the 'Notification Class' in step 20),
            'Priority' =                       (the 'Priority' in step 20),
            'Event Type' =                     (the 'Event Type' in step 20),
            'Notify Type' =                    ACK_NOTIFICATION,
            'To State' =                       FAULT
    ELSE
.       RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =             (the value of the 'Process Identifier' in step 20),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        (the 'Event Object Identifier' in step 20),
            'Time Stamp' =                     (the 'Time Stamp' in step 20),
            'Notification Class' =             (the 'Notification Class' in step 20),
            'Priority' =                       (the 'Priority' in step 20),
            'Event Type' =                     (the 'Event Type' in step 20),
            'Notify Type' =                    ACK_NOTIFICATION
33.     VERIFY Acked_Transitions = (FALSE, TRUE, FALSE)
        }
34. TRANSMIT AcknowledgeAlarm-Request,
        'Acknowledging Process Identifier' =     (the value of the 'Process Identifier' in step 13),
        'Event Object Identifier' =              (the 'Event Object Identifier' in step 13),
        'Event State Acknowledged' =             NORMAL,
        'Time Stamp' =                           (the 'Time Stamp in step 13),
        'Time of Acknowledgment' =               (the current time)
35. RECEIVE BACnet-SimpleACK-PDU
36. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =               (the value of the 'Process Identifier' in step 13),
            'Initiating Device Identifier' =     IUT,
            'Event Object Identifier' =          (the 'Event Object Identifier' in step 13),
            'Time Stamp' =                       (the 'Time Stamp' in step 13),
            'Notification Class' =               (the 'Notification Class' in step 13),
            'Priority' =                         (the 'Priority' in step 13),
            'Event Type' =                       (the 'Event Type' in step 13),
            'Notify Type' =                      ACK_NOTIFICATION,
            'To State' =                         NORMAL
    ELSE
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =               (the value of the 'Process Identifier' in step 13),
            'Initiating Device Identifier' =     IUT,
            'Event Object Identifier' =          (the 'Event Object Identifier' in step 13),
            'Time Stamp' =                       (the 'Time Stamp' in step 13),
            'Notification Class' =              (the 'Notification Class' in step 13),
            'Priority' =                        (the 'Priority' in step 13),
            'Event Type' =                      (the 'Event Type' in step 13),
            'Notify Type' =                     ACK_NOTIFICATION
37. VERIFY Acked_Transitions = (FALSE, TRUE, TRUE)
38. TRANSMIT AcknowledgeAlarm-Request,
        'Acknowledging Process Identifier' =    (the value of the 'Process Identifier' in step 7),
        'Event Object Identifier' =             (the 'Event Object Identifier' in step 7),
        'Event State Acknowledged' =            OFFNORMAL,
        'Time Stamp' =                          (the 'Time Stamp in step 7),
        'Time of Acknowledgment' =              (the current time)
39. RECEIVE BACnet-SimpleACK-PDU
40. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (the value of the 'Process Identifier' in step 7),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the 'Event Object Identifier' in step 7),
            'Time Stamp' =                      (the 'Time Stamp' in step 7),
            'Notification Class' =              (the 'Notification Class' in step 7),
            'Priority' =                        (the 'Priority' in step 7),
            'Event Type' =                      (the 'Event Type' in step 7),
            'Notify Type' =                     ACK_NOTIFICATION,
            'To State' =                        OFFNORMAL
    ELSE
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =              (the value of the 'Process Identifier' in step 7),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the 'Event Object Identifier' in step 7),
            'Time Stamp' =                      (the 'Time Stamp' in step 7),
            'Notification Class' =              (the 'Notification Class' in step 7),
            'Priority' =                          (the 'Priority' in step 7),
            'Event Type' =                      (the 'Event Type' in step 7),
            'Notify Type' =                     ACK_NOTIFICATION
41. VERIFY Acked_Transitions = (TRUE, TRUE, TRUE)
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.1.12 Notify_Type Test

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.29, 12.2.25, 12.3.26, 12.4.22, 12.6.24, 12.7.28, 12.8.26, 12.12.6, 12.15.21, 12.16.21, 12.17.35, 12.18.19, 12.19.20, 12.20.20, 12.23.28 and 12.25.24.

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. This test applies to Event Enrollment objects and Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Loop, Multi-state Input, Multi-state Output, Multi-state Value, Pulse Converter and Trend Log objects that support intrinsic reporting.

Configuration Requirements: The IUT shall be configured with two event-generation objects, E_1 and E_2. Object E1 shall be configured with a Notify_Type of ALARM and E2 shall be configured with a Notify_Type of EVENT. Both objects shall be in a NORMAL Event_State at the beginning of the test. The Event_Enable and Acked_Transitions properties shall be configured with a value of (TRUE, TRUE, TRUE). For analog objects the Limit_Enable property shall be configured with the value (TRUE, TRUE). The value of the Transitions parameter for all recipients shall be (TRUE, TRUE, TRUE)

In the test description below X_1 and X_2 are used to designate the event-triggering property linked to E_1 and E_2 respectively.

Test Steps:

```
1.  VERIFY ($E_1$), Event_State = NORMAL
2.  VERIFY ($E_2$), Event_State = NORMAL
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  IF ($X_1$ is writable) THEN
            WRITE $X_1$ = (a value that is OFFNORMAL)
    ELSE
            MAKE ($X_1$ a value that is OFFNORMAL)
5.  WAIT (Time_Delay)
6.  BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
                    'Event Object Identifier' =         ($E_1$),
                    'Time Stamp' =                      (the current local time),
                    'Notification Class' =              (the class corresponding to the object being tested),
                    'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                      (any valid event type),
                    'Notify Type' =                     ALARM,
                    'AckRequired' =                     TRUE | FALSE,
                    'From State' =                      NORMAL,
                    'To State' =                        OFFNORMAL,
                    'Event Values' =                    (values appropriate to the event type)
7.  IF ($X_2$ is writable) THEN
            WRITE $X_2$ = (a value that is OFFNORMAL)
    ELSE
            MAKE ($X_2$ a value that is OFFNORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
                    'Event Object Identifier' =         ($E_2$),
                    'Time Stamp' =                      (the current local time),
                    'Notification Class' =              (the class corresponding to the object being tested),
                    'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                      (any valid event type),
                    'Notify Type' =                     EVENT,
                    'AckRequired' =                     TRUE | FALSE,
                    'From State' =                      NORMAL,
                    'To State' =                        OFFNORMAL,
                    'Event Values' =                    (values appropriate to the event type)
```

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

7.3.1.13 Limit_Enable Test

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clauses: 12.1.26, 12.2.22, 12.3.23, 12.4.19 and 12.23.25.

Purpose: To verify that the Limit_Enable property correctly enables or disables reporting of out of range events. This test applies to Accumulator, Analog Input, Analog Output, Analog Value and Pulse Converter objects that support intrinsic reporting. If the Limit_Enable property is not writable and cannot be reconfigured this test shall be omitted.

Test Concept: The event-triggering property is manipulated to cause both the high limit and the low limit to be exceeded for each possible combination of values for Limit_Enable. The resulting event notification messages are monitored to verify that they are transmitted only for circumstances where the associated event limit is enabled.

Configuration Requirements: The Limit_Enable property shall be configured with the value (TRUE, TRUE). If Limit_Enable is not writable there shall be additional event-generating objects of the same type that have Limit_Enable configured with the values (FALSE, TRUE), (TRUE, FALSE) and (FALSE, FALSE).

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

```
1.  VERIFY Limit_Enable  = (TRUE, TRUE)
2.  VERIFY Event_State = NORMAL
3.  WAIT (Time_Delay + Notification Fail Time)
4.  IF (X is writable) THEN
            WRITE X = (a value that exceeds High_Limit)
    ELSE
            MAKE (X a value that exceeds High_Limit)
5.  WAIT (Time_Delay)
6.  BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =            (any valid process ID),
                    'Initiating Device Identifier' =  IUT,
                    'Event Object Identifier' =       (the object configured for this test),
                    'Time Stamp' =                    (the current local time),
                    'Notification Class' =            (the class corresponding to the object being tested),
                    'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                    OUT_OF_RANGE,
                    'Notify Type' =                   ALARM | EVENT,
                    'AckRequired' =                   TRUE | FALSE,
                    'From State' =                    NORMAL,
                    'To State' =                      HIGH_LIMIT,
                    'Event Values' =                  (values appropriate to the event type)
7.  IF (X is writable) THEN
            WRITE X = (a value that is lower than Low_Limit)
    ELSE
            MAKE (X a value that is lower than Low_Limit)
8.  WAIT (Time_Delay)
9.  BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =            (any valid process ID),
                    'Initiating Device Identifier' =  IUT,
                    'Event Object Identifier' =       (the object configured for this test),
                    'Time Stamp' =                    (the current local time),
                    'Notification Class' =            (the class corresponding to the object being tested),
                    'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                    OUT_OF_RANGE,
                    'Notify Type' =                   ALARM | EVENT,
                    'AckRequired' =                   TRUE | FALSE,
                    'From State' =                    HIGH_LIMIT,
                    'To State' =                      LOW_LIMIT,
                    'Event Values' =                  (values appropriate to the event type)
10. WRITE Limit_Enable = (FALSE, TRUE)
11. VERIFY Limit_Enable = (FALSE, TRUE)
12. IF (X is writable) THEN
            WRITE X = (a value that exceeds High_Limit)
    ELSE
            MAKE (X a value that exceeds High_Limit)
13. WAIT (Time_Delay)
```

```
14. BEFORE **Notification Fail Time**
          RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
                    'Event Object Identifier' =         (the object configured for this test),
                    'Time Stamp' =                      (the current local time),
                    'Notification Class' =              (the class corresponding to the object being tested),
                    'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                      OUT_OF_RANGE,
                    'Notify Type' =                     ALARM | EVENT,
                    'AckRequired' =                     TRUE | FALSE,
                    'From State' =                      LOW_LIMIT,
                    'To State' =                        HIGH_LIMIT,
                    'Event Values' =                    (values appropriate to the event type)
15. IF (X is writable) THEN
          WRITE X = (a value that is lower than Low_Limit)
    ELSE
          MAKE (X a value that is lower than Low_Limit)
16. WAIT (Time_Delay + **Notification Fail Time**)
17. CHECK (verify that no notification message was transmitted)
18. WRITE Limit_Enable = (TRUE, FALSE)
19. VERIFY Limit_Enable = (TRUE, FALSE)
20. IF (X is writable) THEN
          WRITE X = (a value that exceeds High_Limit)
    ELSE
          MAKE (X a value that exceeds High_Limit)
21. WAIT (Time_Delay + **Notification Fail Time**)
22. CHECK (verify that no notification message was transmitted)
23. IF (X is writable) THEN
          WRITE X = (a value that is lower than Low_Limit)
    ELSE
          MAKE (X a value that is lower than Low_Limit)
24. WAIT (Time_Delay)
25. BEFORE **Notification Fail Time**
          RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
                    'Event Object Identifier' =         (the object configured for this test),
                    'Time Stamp' =                      (the current local time),
                    'Notification Class' =              (the class corresponding to the object being tested),
                    'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                      OUT_OF_RANGE,
                    'Notify Type' =                     ALARM | EVENT,
                    'AckRequired' =                     TRUE | FALSE,
                    'From State' =                      HIGH_LIMIT,
                    'To State' =                        LOW_LIMIT,
                    'Event Values' =                    (values appropriate to the event type)
26. WRITE Limit_Enable = (FALSE, FALSE)
27. VERIFY Limit_Enable = (FALSE, FALSE)
28. IF (X is writable) THEN
          WRITE X = (a value that exceeds High_Limit)
    ELSE
          MAKE (X a value that exceeds High_Limit)
29. WAIT (Time_Delay + **Notification Fail Time**)
30. CHECK (verify that no notification message was transmitted)
31. IF (X is writable) THEN
          WRITE X = (a value that is lower than Low_Limit)
    ELSE
          MAKE (X a value that is lower than Low_Limit)
32. WAIT (Time_Delay + **Notification Fail Time**)
```

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

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. If the Limit_Enable property is not writable the equivalent tests may be applied to separate or reconfigured objects that have the appropriate value for Limit_Enable.

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 .. 2^{32}-1 by setting it to the minimum, the maximum, and a randomly chosen value.

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

7.3.1.14.2 Recipient_List Test

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 .. 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.

```
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)
```

7.3.2 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 that 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: -INF < x < INF),
2.  VERIFY Average_Value =      (a value ≠ NaN),
3.  VERIFY Maximum_Value =      (a value x: Minimum_Value ≤ x < INF),
4.  VERIFY Attempted_Samples =  (a value x > 0),
5.  VERIFY Valid_Samples =      (a value x > 0),
6.  WRITE Attempted_Samples =   0,
7.  VERIFY Attempted_Samples =  0,
8.  VERIFY Minimum_Value =      INF,
9.  VERIFY Maximum_Value =      -INF,
10. VERIFY Average_Value =      NaN,
11. VERIFY Valid_Samples =      0,
12. WAIT (at least two sample times),
13. VERIFY Minimum_Value =      (a value x: -INF < x < INF),
14. VERIFY Average_Value =      (a value ≠ NaN),
15. VERIFY Maximum_Value =      (a value x: Minimum_Value ≤ x < INF),
16. VERIFY Attempted_Samples =  (a value x ≥ 2),
17. VERIFY Valid_Samples =      (a value x ≥ 2),
18. WRITE Window_Interval =     (any new value that will result in an appropriate sample time),
19. VERIFY Attempted_Samples =  0,
20. VERIFY Minimum_Value =      INF,
21. VERIFY Maximum_Value =      -INF,
22. VERIFY Average_Value =      NaN,
23. VERIFY Valid_Samples =      0,
24. WAIT (at least two sample times),
25. VERIFY Minimum_Value =      (a value x: -INF < x < INF),
26. VERIFY Average_Value =      (a value ≠ NaN),
27. VERIFY Maximum_Value =      (a value x: Minimum_Value ≤ x < INF),
28. VERIFY Attempted_Samples =  (a value x ≥ 2),
29. VERIFY Valid_Samples =      (a value x ≥ 2),
30. WRITE Window_Samples =      (any new value that will result in an appropriate sample time),
31. VERIFY Attempted_Samples =  0,
32. VERIFY Minimum_Value =      INF,
33. VERIFY Maximum_Value =      -INF,
34. VERIFY Average_Value =      NaN,
35. VERIFY Valid_Samples =      0,
36. IF (Object_Property_Reference is writable) THEN {
        WAIT (at least two sample times),
        VERIFY Minimum_Value =              (a value x: -INF < x < INF),
        VERIFY Average_Value =              (a value ≠ NaN),
        VERIFY Maximum_Value =              (a value x: Minimum_Value ≤ x < INF),
        VERIFY Attempted_Samples =          (a value x ≥ 2),
        VERIFY Valid_Samples =              (a value x ≥ 2),
        WRITE Object_Property_Reference =   (any new value),
        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, 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

7.3.2.6.7 Tests to verify intrinsic reporting capabilities for Binary Output objects are covered in 8.4.4. 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.

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 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 + UTC_Offset – Schedule Evaluation Fail Time – 1 minute) ) |
    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.

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 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) ) |
    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) ) |
    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.

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 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 that matches the BACnetWeekNDay mask. The Present_Value is read and verified to be TRUE. The clock is reset to a value 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 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 that matches the BACnetWeekNDay mask except for the day of the week. The Present_Value is read and verified to be FALSE.

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' = (24-November-2001, 13:00:00 + UTC_Offset) |
    MAKE (the local time = 24-November-2001 at 13:00:00)
2.  WAIT Schedule Evaluation Fail Time
3.  VERIFY Present_Value = TRUE
4.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (27-October-2001, 13:00:00 + UTC_Offset) |
    MAKE (the local time = 27-October-2001 at 13:00:00)
5.  WAIT Schedule Evaluation Fail Time
6.  VERIFY Present_Value = FALSE
7.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (17-November-2001, 13:00:00 + UTC_Offset) |
```

MAKE (the local time = 17-November-2001 at 13:00:00)
8. WAIT **Schedule Evaluation Fail Time**
9. VERIFY Present_Value = FALSE
10. (TRANSMIT TimeSynchronization-Request,
    'Time' = (25-November-2001, 13:00:00 + UTC_Offset) |
    MAKE (the local time = 25-November-2001 at 13:00:00)
11. WAIT **Schedule Evaluation Fail Time**
12. 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, 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
2. VERIFY In_Process = TRUE
3. CHECK (for the externally visible actions and verify that there is a post delay)
4. VERIFY In_Process = FALSE
5. 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, 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-Simple-ACK
6.  VERIFY In_Process = FALSE
```

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, 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, then this test shall be omitted.

Configuration Requirements: The IUT shall be configured with a Command object having two distinct action lists, X and Y, that include writing to a sequence of externally visible outputs. There shall be a post delay between writes to the externally visible outputs that is long enough for the tester to observe the delay (This ensures In_Process remains TRUE long enough to command the second action list).

Test Steps:

```
1.  WRITE Present_Value = X
```

```
2.   RECEIVE Simple-ACK-PDU
3.   WRITE Present_Value = Y
4.   RECEIVE BACnet-Error-PDU
         Error Class  =   SERVICES,
         Error Code  =   SERVICE_REQUEST_DENIED | OTHER
5. CHECK (that the externally visible actions of X took place)
6. CHECK (that the externally visible actions of Y did not take place)
7. VERIFY In_Process = FALSE,
8. 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.

```
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.

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 Test

All necessary tests for functionality of the Device object are covered by tests for the application service or special functionality to which they correspond.

7.3.2.11 Event Enrollment Object Test

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.

7.3.2.12 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.

```
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.

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.

```
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

Test Steps:

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

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 7.3.2.15.5. Run the tests using a Multi-state Value object.

7.3.2.21 Notification Class Object

7.3.2.21.1 Priority Tests

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.6.

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 it. 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.

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

1. WAIT (Time_Delay + **Notification Fail Time**)
2. VERIFY Event_State = NORMAL

```
3.  IF (X is writable) THEN
            WRITE X = (a value that is OFFNORMAL)
    ELSE
            MAKE (X have a value that is OFFNORMAL)
4.  WAIT (Time_Delay)
5.  BEFORE **Notification Fail Time**
            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)
6.  VERIFY Event_State = OFFNORMAL
7.  IF (X is writable) THEN
            WRITE X = (a value that is NORMAL)
    ELSE
            MAKE (X have a value that is NORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE  **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =             (any valid process ID),
                    'Initiating Device Identifier' =   IUT,
                    'Event Object Identifier' =        (the event-generating object configured for this test),
                    'Time Stamp' =                     (any valid time stamp),
                    'Notification Class' =             (the class corresponding to the object being tested),
                    'Priority' =                       (the value configured to correspond to a TO-NORMAL transition),
                    'Event Type' =                     (any valid event type),
                    'Notify Type' =                    EVENT | ALARM,
                    'AckRequired' =                    TRUE | FALSE,
                    'From State' =                     OFNORMAL,
                    'To State' =                       NORMAL,
                    'Event Values' =                   (values appropriate to the event type)
10. VERIFY Event_State = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
12.         MAKE (the event-triggering object change to a fault condition)
13.         WAIT (Time_Delay)
14.         BEFORE **Notification Fail Time**
                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =             (any valid process ID),
                    'Initiating Device Identifier' =   IUT,
                    'Event Object Identifier' =        (the event-generating object configured for this test),
                    'Time Stamp' =                     (any valid time stamp),
                    'Notification Class' =             (the class corresponding to the object being tested),
                    'Priority' =                       (the value configured to correspond to a TO-FAULT transition),
                    'Event Type' =                     (any valid event type),
                    'Notify Type' =                    EVENT | ALARM,
                    'AckRequired' =                    TRUE | FALSE,
                    'From State' =                     NORMAL,
                    'To State' =                       FAULT,
                    'Event Values' =                   (values appropriate to the event type)
15.         VERIFY Event_State = FAULT
16.         MAKE (the event-triggering object change to a normal condition)
17.         WAIT (Time_Delay)
18.     BEFORE **Notification Fail Time**
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object configured for this test),
                'Time Stamp' =                    (any valid time stamp),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    FAULT,
                'To State' =                      NORMAL,
                'Event Values' =                  (values appropriate to the event type)
19.     VERIFY Event_State = NORMAL
        }
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.2 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

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.7.

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 it. 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 7.3.2.20.1 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 7.3.2.20.1 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. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.2.2 Ack_Required True Test

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.7.

Purpose: To verify that if the Ack_Required property indicates that event notifications require acknowledgment, then the AckRequired parameter of the notification message conveys that fact. If the IUT does not support acknowledged 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 it. 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 7.3.2.20.1 except for an additional requirement that the value of the Ack_Required property shall be B'111' indicating that acknowledgments are required.

Test Steps: The test steps are identical to those in 7.3.2.20.1 with the additional constraint that the 'AckRequired' parameter shall be TRUE in all event notification messages.

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3 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

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22; TimeSynchronization Service Execution Tests, 9.30.

BACnet Reference Clause: 12.21.8.

Purpose: To verify the operation of the Valid Days parameter of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The TD will select one instance of the Notification Class object and one instance of an event-generating object that is linked to it. The Recipient_List of the Notification Class object shall contain a single recipient with the Valid Days parameter configured so that at least one day is TRUE and at least one day is FALSE. The properties of the event-generating object will be manipulated to cause the Event_State to change from NORMAL to OFFNORMAL. The tester verifies that if the local date is one of the valid days a notification message is transmitted and the if local date is not a valid day then no notification message is transmitted.

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The event-generating object shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE and at least one day of the week has a value of FALSE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions.

In the test description below, “X” is used to designate the event-triggering property.

Test Steps:

```
1.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (any time within the window defined by From Time and To Time in the BACnet Destination that
                corresponds to one of the valid days)) |
    MAKE (the local date and time = (any time within the window defined by From Time and To Time in the
            BACnetDestination that corresponds to one of the valid days))
2.  WAIT (Time_Delay + Notification Fail Time)
```

```
3.  VERIFY Event_State = NORMAL
4.  IF (X is writable) THEN
            WRITE X = (a value that is OFFNORMAL)
    ELSE
            MAKE (X have a value that is OFFNORMAL)
5.  WAIT (Time_Delay)
6.  BEFORE Notification Fail Time
            RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =              (any valid process ID),
                    'Initiating Device Identifier' =    IUT,
                    'Event Object Identifier' =         (the event-generating object configured for this test),
                    'Time Stamp' =                      (the current local time),
                    'Notification Class' =              (the class corresponding to the object being tested),
                    'Priority' =                        (the value configured to correspond to a TO-OFFNORMAL transition),
                    'Event Type' =                      (any valid event type),
                    'Notify Type' =                     EVENT | ALARM,
                    'AckRequired' =                     TRUE | FALSE,
                    'From State' =                      NORMAL,
                    'To State' =                        OFFNORMAL,
                    'Event Values' =                    (values appropriate to the event type)
7.  VERIFY Event_State = OFFNORMAL
8.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (any time within the window defined by From Time and To time in the BACnet Destination that
                    corresponds to one of the invalid days)) |
    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))
9.  IF (X is writable) THEN
            WRITE X = (a value that is NORMAL)
    ELSE
            MAKE (X have a value that is NORMAL)
10. WAIT (Time_Delay + Notification Fail Time)
11. CHECK (verify that no notification message was transmitted)
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3.2 FromTime and ToTime Test

Dependencies: ValidDays Test, 7.3.2.21.3.1; ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; TimeSynchronization Service Execution Tests, 9.30.

BACnet Reference Clause: 12.21.8.

Purpose: To verify the operation of the From Time and To Time parameters of a BACnetDestination as used in the Recipient_List property of the Notification Class object.

Test Concept: The case where the local date and time fall within the window defined by the From Time and To Time parameters is covered by the ValidDays test in 7.3.2.21.3.1. This test uses the same IUT configuration and sets the local time to a value that is one of the ValidDays but outside of the window defined by the From Time and To Time parameters. The objective is to verify that an event notification message is not transmitted when the event is triggered.

Configuration Requirements: The configuration requirements are identical to the requirements in 7.3.2.21.3.1.

Test Steps:

```
1.  (TRANSMIT TimeSynchronization-Request,
        'Time' = (any time outside the window defined by From Time and To Time in the BACnet Destination that
                    corresponds to one of the valid days)) |
    MAKE (the local date and time = (any time outside the window defined by From Time and To Time in the
            BACnetDestination that corresponds to one of the valid days))
2.  WAIT (Time_Delay + **Notification Fail Time**)
3.  VERIFY Event_State = NORMAL
4.  IF (X is writable) THEN
          WRITE X = (a value that is OFFNORMAL)
    ELSE
          MAKE (X have a value that is OFFNORMAL)
10. WAIT (Time_Delay + **Notification Fail Time**)
11. CHECK (verify that no notification message was transmitted)
```

7.3.2.21.3.3 IssueConfirmedNotifications Test

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.8.

Purpose: To verify that ConfirmedEventNotification messages are used if the Issue Confirmed Notifications parameter has the value TRUE and UnconfirmedEventNotification messages are used if the value is FALSE. If the IUT does not support both confirmed and unconfirmed event notifications this test may be omitted.

Configuration Requirements: The IUT shall be configured with two or more instances of the Notification Class object and event-generating objects that are linked to the Notification Class objects. The event-generating objects may be objects that support intrinsic reporting or they may be Event Enrollment objects. The event-generating objects shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating objects shall be configured to be in a NORMAL event state at the start of the test. One Notification Class object, N_1, shall be configured with Issue Confirmed Notifications equal to TRUE. The other Notification Class object, N_2, shall be configured with Issue Confirmed Notifications equal to FALSE. The Valid Days parameter shall be configured so that at least one day of the week has a value of TRUE. The Transitions parameter shall be configured for the recipient to receive notifications for all event transitions. The local date and time shall be configured to be within the window defined by From Time and To Time on one of the ValidDays.

In the test description below "X_1" and "X_2" are used to designate the event-triggering property linked to Notification objects "N_1" and "N_2" respectively.

Test Steps:

```
1.  VERIFY (the event-generating object linked to N1), Event_State = NORMAL
2.  VERIFY (the event-generating object linked to N2), Event_State = NORMAL
3.  WAIT (Time_Delay + **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 (Time_Delay)
6.  BEFORE **Notification Fail Time**
          RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =           (any valid process ID),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =      (the event-generating object linked to N1),
                'Time Stamp' =                   (the current local time),
                'Notification Class' =           (the class corresponding to the object being tested),
                'Priority' =                     (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                   (any valid event type),
                'Notify Type' =                  EVENT | ALARM,
                'AckRequired' =                  TRUE | FALSE,
                'From State' =                   NORMAL,
                'To State' =                     OFFNORMAL,
                'Event Values' =                 (values appropriate to the event type)
7.  IF (X2 is writable) THEN
          WRITE X2 = (a value that is OFFNORMAL)
    ELSE
        MAKE ($X_2$ a value that is OFFNORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE **Notification Fail Time**
RECEIVE UnconfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object linked to $N_2$),
                'Time Stamp' =                    (the current local time),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    NORMAL,
                'To State' =                      OFFNORMAL,
                'Event Values' =                  (values appropriate to the event type)
```

Notes to Tester: If the Recipient_List is writable and the Issue Confirmed Notifications can be changed then this test can be performed using only one Notification Class object by writing to the Recipient_List in order to change between confirmed and unconfirmed notifications. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.21.3.4 Transitions Test

Dependencies: ConfirmedEventNotification Service Initiation Tests, 8.4; UnconfirmedEventNotification Service Initiation Tests, 8.5; ReadProperty Service Execution Tests, 9.18; WriteProperty Service Execution Tests, 9.22.

BACnet Reference Clause: 12.21.8.

Purpose: To verify that notification messages are transmitted only if the bit in the Transitions parameter corresponding to the event transition is set.

Test Concept: The IUT is configured such that the Transitions parameter indicates that some event transitions are to trigger an event notification and some are not. Each event transition is triggered and the IUT is monitored to verify that notification messages are transmitted only for those transitions for which the Transitions parameter has a value of TRUE.

Configuration Requirements: The IUT shall be configured with one or more instance of the Notification Class object and at least one event-generating object that is linked to the Notification Class object. The event-generating object may be any object that supports intrinsic reporting or it may be an Event Enrollment object. The event-generating object shall have the Event_Enable property configured to transmit notification messages for all event transitions. The event-generating object shall be configured to be in a NORMAL event state at the start of the test. The Notification Class object shall be configured with a single recipient in the Recipient_List. The Transitions parameter shall be configured with a value of TRUE for either the TO-OFFNORMAL transition or the TO-NORMAL transition and the other event transition shall have a value of FALSE. The local time shall be configured such that it represents one of the valid days in the window specified by From Time and To Time.

In the test description below, “X” is used to designate the event-triggering property.

```
1.  VERIFY Event_State = NORMAL
2.  WAIT (Time_Delay + **Notification Fail Time**)
3.  IF (X is writable) THEN
        WRITE X = (a value that is OFFNORMAL)
    ELSE
        MAKE (X have a value that is OFFNORMAL)
4.  WAIT (Time_Delay)
5.  BEFORE **Notification Fail Time**
        IF (the Transitions bit corresponding to the TO-OFFNORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object configured for this test),
                'Time Stamp' =                    (the current local time),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    NORMAL,
                'To State' =                      OFFNORMAL,
                'Event Values' =                  (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
6.  VERIFY Event_State = OFFNORMAL
7.  IF (X is writable) THEN
        WRITE X = (a value that is NORMAL)
    ELSE
        MAKE (X have a value that is NORMAL)
8.  WAIT (Time_Delay)
9.  BEFORE Notification Fail Time
        IF (the Transitions bit corresponding to the TO-NORMAL transition is TRUE) THEN
            RECEIVE ConfirmedEventNotification-Request,
                'Process Identifier' =            (any valid process ID),
                'Initiating Device Identifier' =  IUT,
                'Event Object Identifier' =       (the event-generating object configured for this test),
                'Time Stamp' =                    (the current local time),
                'Notification Class' =            (the class corresponding to the object being tested),
                'Priority' =                      (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                    (any valid event type),
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    OFFNORMAL,
                'To State' =                      NORMAL,
                'Event Values' =                  (values appropriate to the event type)
        ELSE
            CHECK (verify that the IUT did not transmit an event notification message)
10. VERIFY Event_State = NORMAL
11. IF (the event-triggering object can be placed into a fault condition) THEN {
        MAKE (the event-triggering object change to a fault condition)
        WAIT (Time_Delay)
        BEFORE Notification Fail Time
            IF (the Transitions bit corresponding to the TO-FAULT transition is TRUE) THEN
                RECEIVE ConfirmedEventNotification-Request,
                    'Process Identifier' =            (any valid process ID),
                    'Initiating Device Identifier' =  IUT,
                    'Event Object Identifier' =       (the event-generating object configured for this test),
                    'Time Stamp' =                    (the current local time),
                    'Notification Class' =            (the class corresponding to the object being tested),
                    'Priority' =                      (the value configured to correspond to a TO-FAULT transition),
                    'Event Type' =                    (any valid event type),
                    'Notify Type' =                   EVENT | ALARM,
                    'AckRequired' =                   TRUE | FALSE,
                    'From State' =                    NORMAL,
                    'To State' =                      FAULT,
                    'Event Values' =                  (values appropriate to the event type)
            ELSE
                CHECK (verify that the IUT did not transmit an event notification message)
        VERIFY Event_State = FAULT
        }
```

Notes to Tester: The UnconfirmedEventNotification service may be substituted for the ConfirmedEventNotification service. The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages.

7.3.2.22 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.

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) | 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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V_2

7.3.2.23.2 Weekly_Schedule Property Test

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

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 V_1. 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 V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V_2
8. (TRANSMIT TimeSynchronization-Request, 'Time' = D_3) | MAKE (the local date and time = D_3)
9. WAIT **Schedule Evaluation Fail Time**
10. VERIFY Present_Value = V_3
11. (TRANSMIT TimeSynchronization-Request, 'Time' = D_4) | MAKE (the local date and time = D_4)
12. WAIT **Schedule Evaluation Fail Time**
13. VERIFY Present_Value = V_4
14. (TRANSMIT TimeSynchronization-Request, 'Time' = D_5) | MAKE (the local date and time = D_5)
15. WAIT **Schedule Evaluation Fail Time**
16. VERIFY Present_Value = V_5

17. (TRANSMIT TimeSynchronization-Request, 'Time' = D_6) | MAKE (the local date and time = D_6)
18. WAIT **Schedule Evaluation Fail Time**
19. VERIFY Present_Value = V_6
20. (TRANSMIT TimeSynchronization-Request, 'Time' = D_7) | 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 7-2) DO {
    (TRANSMIT TimeSynchronization-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

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a date appearing in a referenced Calendar object 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 that references a Calendar object with a non-empty Date_List. The criteria for the dates used 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 V_1.

Table 7-3. 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, and 2C. Higher eventPriority than any other 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 BACnetDailtySchedule in the referencing Schedule object.	Other than V_1

Test Steps:

1. VERIFY Present_Value = (any value other than V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

7.3.2.23.3.2 Calendar Entry Date Test

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

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 7-4. 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 specific date. 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 V_1. 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 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 BACnetDailtySchedule in the referencing Schedule object.	Other than V_1

Test Steps:

1. VERIFY Present_Value = (any value other than V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

7.3.2.23.3.3 Calendar Entry DateRange Test

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

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 7-5. 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 date range. 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 V_1. 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 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 BACnetDailtySchedule in the referencing Schedule object.	Other than V_1

Test Steps:

1. VERIFY Present_Value = (any value other than V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

7.3.2.23.3.4 Calendar Entry WeekNDay Month Test

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

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 7-6. 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 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 V_1. 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 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 2.D. 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 BACnetDailtySchedule in the referencing Schedule object.	Other than V_1

Test Steps:

1. VERIFY Present_Value = (any value other than V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

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.

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 a 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 V_1. 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 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 V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

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.

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 7-8. 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 last week of the month. 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 V_1. 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 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 V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

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.

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 7-9. 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 7-9. The local date and time shall be set such that the Present_Value property has a value other than V_1. 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 7-9. 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 V_1)
2. (TRANSMIT TimeSynchronization-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) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = (any value other than $V_{1)}$

7.3.2.23.3.8 Event Priority Test

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

BACnet Reference Clause: 12.24.8.

Purpose: To verify that a BACnetSpecialEvent of a higher priority takes precedence over one of lower priority when both 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 D_1 represents a date and time where all of the special events are active.

Test Steps:

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

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) | 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.

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.

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

Test Steps:

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

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

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.

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. 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' = $D_1$) | 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' = COLDSTART,
          'Password' =                    (any valid password)
         RECEIVE BACnet-Simple-ACK-PDU
     ELSE
         MAKE (the IUT reinitialize)
5.   CHECK (Did the IUT perform a COLDSTART reboot?)
6.   WAIT **Schedule Evaluation Fail Time**
7.   IF (Protocol_Revision is present and Protocol_Revision $\geq$ 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.

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

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.

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' = $D_t$) | MAKE (the local date and time = $D_t$)
2.  WAIT **Schedule Evaluation Fail Time**
```

3. VERIFY Present_Value = V_1
4. VERIFY (value of referenced property in IUT) = V_1
5. (TRANSMIT TimeSynchronization-Request, 'Time' = D_2) | MAKE (the local date and time = D_2)
6. WAIT **Schedule Evaluation Fail Time**
7. VERIFY Present_Value = V_2
8. VERIFY (value of referenced property in IUT) = V_2

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.

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. 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' = $D_t$) | MAKE (the local date and time = $D_t$)
2. WAIT Schedule Evaluation Fail Time
3. VERIFY Present_Value = $V_1$
4. (TRANSMIT TimeSynchronization-Request, 'Time' = $D_2$) | MAKE (the local date and time = $D_2$)
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' =       $V_2$
6. WAIT Schedule Evaluation Fail Time
7. VERIFY Present_Value = $V_2$
```

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
```

7.3.2.24 Trend Log Object Tests

The Trend Log object has only a few properties required to be writable or otherwise configurable. The Trend Log object 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 the Trend Log object center upon the collection of (time, value) records in its Log_Buffer and its issuance of notifications when a predetermined number of records have been collected since startup or the last preceding notification.

7.3.2.24.1 Log_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 Log_Enable property enables and disables the logging of data by the Trend Log object.

Test Concept: The Trend Log is configured to acquire data by each means (polling and COV subscription) available to the implementation. Log_Enable is enabled and the collection of one or more records in the Log_Buffer is confirmed. Log_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. WRITE Log_Enable = FALSE
2. WRITE Record_Count = 0
3. WAIT  Internal Processing Fail Time
4. TRANSMIT ReadProperty-Request,
        'Object Identifier' =   (the object being tested),
        'Property Identifier' =     Total_Record_Count
5. RECEIVE ReadProperty-ACK,
        'Object Identifier' =   (the object being tested),
        'Property Identifier' =     Total_Record_Count
        'Property Value' =    (any valid value, X)
6. WRITE Log_Enable = TRUE
7. WAIT  Internal Processing Fail Time
8. IF (COV subscription in use) THEN
     MAKE (monitored value change more than Client_COV_Increment)
   ELSE
     WAIT (Log_Interval)
9. WAIT (Notification Fail Time + Internal Processing Fail Time)
10. VERIFY Total_Record_Count > (value X returned in step 5)
11. WRITE Log_Enable = FALSE
12. WAIT  Internal Processing Fail Time
13. TRANSMIT ReadProperty-Request,
        'Object Identifier' =   (the object being tested),
        'Property Identifier' =     Total_Record_Count
14. RECEIVE ReadProperty-ACK,
        'Object Identifier' =   (the object being tested),
        'Property Identifier' =     Total_Record_Count
        'Property Value' =    (any valid value, X)
15. IF (COV subscription in use) THEN
     MAKE (monitored value change more than Client_COV_Increment)
   ELSE
     WAIT (Log_Interval)
16. WAIT (Notification Fail Time + Internal Processing Fail Time)
17. VERIFY Total_Record_Count = (value X returned in step 14)
```

7.3.2.24.2 Start_Time Test

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

BACnet Reference Clause: 12.25.6.

Purpose: To verify that logging is enabled at the time specified by Start_Time.

Test Concept: The Trend Log is configured to acquire data by each means (polling and COV subscription) 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; Log_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. IF (COV subscription in use) THEN
      MAKE (monitored value change more than Client_COV_Increment)
   ELSE
      WAIT (Log_Interval)
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. IF (COV subscription in use) THEN
      MAKE (monitored value change more than Client_COV_Increment)
   ELSE
      WAIT (Log_Interval)
11. WAIT (Notification Fail Time + Internal Processing Fail Time)
12. VERIFY Total_Record_Count > (value X returned in step 4)
```

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 Trend Log is configured to acquire data by each means (polling and COV subscription) 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 Log_Enable = FALSE
2. WAIT  Internal Processing Fail Time
3. WRITE Record_Count = 0
4. TRANSMIT ReadProperty-Request,
        'Object Identifier' =    (the object being tested),
        'Property Identifier' =  Total_Record_Count
5. RECEIVE ReadProperty-ACK,
        'Object Identifier' =    (the object being tested),
        'Property Identifier' =  Total_Record_Count
        'Property Value' =       (any valid value, X)
6. WRITE Log_Enable = TRUE.
7. WAIT  Internal Processing Fail Time
8. IF (COV subscription in use) THEN
      MAKE (monitored value change more than Client_COV_Increment)
   ELSE
      WAIT (Log_Interval)
9. WAIT (Notification Fail Time + Internal Processing Fail Time)
10. VERIFY Total_Record_Count > (value X returned in step 5)
11. WHILE (IUT clock is earlier than Stop_Time) DO {}
```

```
12. WAIT (Notification Fail Time + Internal Processing Fail Time)
13. TRANSMIT ReadProperty-Request,
        'Object Identifier' =    (the object being tested),
        'Property Identifier' =      Total_Record_Count
14. RECEIVE ReadProperty-ACK,
        'Object Identifier' =    (the object being tested),
        'Property Identifier' =      Total_Record_Count
        'Property Value' =     (any valid value, X)
15. IF (COV subscription in use) THEN
      MAKE (monitored value change more than Client_COV_Increment)
    ELSE
      WAIT (Log_Interval)
16. WAIT (Notification Fail Time + Internal Processing Fail Time)
17. VERIFY Total_Record_Count = (value X returned in step 14)
```

7.3.2.24.4 Log_Interval Test

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

BACnet Reference Clause: 12.25.9.

Purpose: To verify that the logging period is controlled by Log_Interval.

Test Concept: The Trend Log 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. Log_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; Unconfirmed Notifications Subscription, 8.10.2.

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. Log_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 or FALSE),
            'Lifetime' =                           (2 * COV_Resubscription_Interval)
   ELSE
      RECEIVE SubscribeCOVProperty-Request,
            'Subscriber Process Identifier' =      (any value),
            'Monitored Object Identifier' =        (the object to be monitored),
            '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. IF  ('Issue Confirmed Notification' = TRUE) THEN
      TRANSMIT ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =      (corresponding value in step 1),
            'Initiating Object 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' =                     (2 * COV_Resubscription_Interval),
            'List of Values' =                     (appropriate BACnetPropertyValue(s))
   ELSE
      TRANSMIT UnconfirmedCOVNotification,
            'Subscriber Process identifier' =      (corresponding value in step 1),
            'Initiating Object 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' =                     (2 * COV_Resubscription_Interval),
            'List of Values' =                (appropriate BACnetPropertyValue(s))
3. WAIT (COV_Resubscription_Interval – Notification Fail Time)
4. BEFORE (2 * Notification Fail Time)
      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' =      (corresponding value in step 1),
                  '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' =      (corresponding value in step 1),
                  'Lifetime' =                           (corresponding value in step 1),
                  'Monitored Property Identifier' =      (corresponding value in step 1),
                  'COV Increment' =                      (corresponding value in step 1)
```

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.

BACnet Reference Clause: 12.25.12.

Purpose: To verify that Stop_When_Full set to TRUE properly indicates that the Trend Log ceases collecting data when its Log_Buffer acquires Buffer_Size data items.

Test Concept: The Trend Log is configured to acquire data by whatever means. Data is collected until more than Buffer_Size records have been collected and Log_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 FALSE. Log_Enable shall be set to FALSE.

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. WRITE Log_Enable = TRUE
6. WHILE ( (Total_Record_Count – (value X returned in step 4))  modulo $2^{32}$ < Buffer_Size ) DO { }
7. WAIT  **Internal Processing Fail Time**
8. VERIFY Log_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.

BACnet Reference Clause: 12.25.12.

Purpose: To verify that Stop_When_Full set to FALSE properly indicates that the Trend Log continues collecting data after its Log_Buffer acquires Buffer_Size data items.

Test Concept: The Trend Log is configured to acquire data by whatever means. Data is collected until more than Buffer_Size records have been collected and Log_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. Log_Enable shall be set to FALSE.

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. WRITE Log_Enable = TRUE
6. WHILE ( (Total_Record_Count – (value X returned in step 4))  modulo $2^{32}$ < (Buffer_Size+1) ) DO { }
7. WAIT  **Internal Processing Fail Time**
8. VERIFY Log_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 Trend Log 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 occur after the end of the test. Log_Enable shall be set to TRUE.

Test Steps:

```
1. WHILE ( Record_Count < Buffer_Size ) DO { }
2. WRITE Log_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.

BACnet Reference Clause: 12.25.15.

Purpose: To verify that the Record_Count property indicates the number of records that are stored in the Log_Buffer.

Test Concept: The Trend Log is configured to acquire data by whatever means. Record_Count is set to zero and Log_Buffer is read to verify no records are present. 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 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 occur after the end of the test. Log_Enable shall be set to FALSE.

Test Steps:

```
1. WRITE Record_Count = 0
2. WAIT Internal Processing Fail Time
3. CHECK ( that Log_Buffer has no records )
4. WRITE Log_Enable = TRUE
5. WHILE ( Record_Count < Buffer_Size/2 ) DO { }
6. WRITE Log_Enable = FALSE
7. WAIT Internal Processing Fail Time
8. VERIFY ( that Log_Buffer has the number of records indicated by Record_Count )
9. WRITE Log_Enable = TRUE
10. WHILE ( Record_Count < Buffer_Size ) DO { }
11. WRITE Log_Enable = FALSE
12. WAIT Internal Processing Fail Time
13. VERIFY ( 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 12.23.16 that the value wrap from 2^{32}-1 to one; even if a record was collected every 100th of a second it could take more than 497 days to complete the test.)

Test Concept: The Trend Log 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. Log_Enable shall be set to FALSE.

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. WRITE Log_Enable = TRUE
6. WHILE ( Record_Count = 0 ) DO { }
7. WRITE Log_Enable = FALSE
8. WAIT  Internal Processing Fail Time
9. IF ( Record_Count = Buffer_Size ) THEN
        ERROR "Buffer full; cannot verify Total_Record_Count value."
   ELSE {
      IF ( Total_Record_Count != Record_Count + (value X returned in step 4)) THEN
         ERROR "Total_Record_Count has incorrect value."
         }
```

7.3.2.24.10 Notification_Threshold Test

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

BACnet Reference Clause: 12.25.17.

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 Trend Log is configured to acquire data by whatever means. Record_Count is set to zero. Collection of data proceeds until a notification is seen, collection is halted and the value of Record_Count is checked. Collection resumes until the second notification, when collection is again halted and Record_Count verified. If, for whatever reason, the IUT cannot be configured such that the TD is able to halt collection before another record is collected after issuing the notification 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. Log_Enable shall be set to FALSE.

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. WRITE Log_Enable = TRUE
6.  MAKE ( Trend Log object collect number of records specified by Notification_Threshold)
```

```
7.  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 appropriate BACnetTimeStamp value),
        '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)
8.  TRANSMIT BACnet-SimpleACK-PDU
9. WRITE Log_Enable = FALSE
10. IF ( Total_Record_Count – (value read in step 4) != Notification_Threshold ) THEN
      ERROR "Notification_Threshold value is incorrect."
11.  WRITE Log_Enable = TRUE
12.  MAKE ( Trend Log object collect number of records specified by Notification_Threshold)
13.  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 appropriate BACnetTimeStamp value),
        '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 7),
                            (current local BACnetDateTime)
14.  TRANSMIT BACnet-SimpleACK-PDU
15. WRITE Log_Enable = FALSE
16. IF ( Total_Record_Count – (value X returned in step 4) != 2 * Notification_Threshold ) THEN
      ERROR "Notification_Threshold value is incorrect."
```

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 properties 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 Notification_Threshold)
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 appropriate BACnetTimeStamp value),
            '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 Notification_Threshold)
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 appropriate BACnetTimeStamp value),
            '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: The Trend Log is configured to acquire data by COV subscription from the TD. After it attempts to subscribe with the TD the Trend Log is halted and Event_State is 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. VERIFY Event_State = NORMAL
2. WRITE Log_Enable = TRUE
3. IF (the IUT uses SubscribeCOV for this Trend Log)
       RECEIVE  SubscribeCOV-Request,
           'Subscriber Process Identifier' =    (any value),
           'Monitored Object Identifier' =      (any object),
           'Issue Confirmed Notifications' =    (TRUE|FALSE),
           'Lifetime' =                         (2 * COV_Resubscription_Interval)
   ELSE
       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)
4. TRANSMIT BACnet-SimpleACK-PDU
5. WAIT COV_Resubscription_Interval
6. VERIFY Event_State = FAULT
```

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

Dependencies: None.

BACnet Reference Clause: 13.5.

Purpose: To verify that the IUT can initiate an AcknowledgeAlarm service request.

Test Steps:

```
1.  TRANSMIT   ConfirmedEventNotification-Request | UnconfirmedEventNotification-Request,
     'Subscriber Process Identifier' =     (any value selected by the TD),
     'Initiating Device Identifier' =      (any value selected by the TD),
     'Event Object Identifier' =           (any value selected by the TD),
     'Time Stamp' =                        (any value selected by the TD),
     'Notification Class' =                (any value selected by the TD),
     'Priority' =                          (any value selected by the TD),
     'Event Type' =                        (any value selected by the TD),
     'Notify Type' =                       ALARM,
     'AckRequired' =                       TRUE,
     'From State' =                        NORMAL
     'To State' =                          (any offnormal state appropriate to the event type),
     '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 initiate an AcknowledgeAlarm service request with parameters appropriate to the event
notification)
4.  RECEIVE AcknowledgeAlarm-Request,
     Acknowledging Process Identifier =    (any process identifier),
     Event Object Identifier =             (the 'Event Object Identifier' from the notification),
     Time Stamp =                          (the 'Time Stamp' from the notification),
     Acknowledgement Source =              (any CharacterString),
     Time of Acknowledgement =             (the current time as determined by the IUT)
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. 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' =                            0
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
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' =                  0,
                '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 **NotificationFailTime**
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 **NotificationFailTime**
        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 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. 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, 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' =                            0
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

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 NotificationFailTime
        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 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. 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' =                             0
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
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' =                0,
                '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  NotificationFailTime
        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 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. 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, 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' =                             0
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
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 NotificationFailTime
        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 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. 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' =                             0
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
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' =                0,
                 '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  NotificationFailTime
         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 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. 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, 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' =                               0
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
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  NotificationFailTime
        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 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. 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 L in the test steps below.

Test Steps:

```
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =       (any value > 0 chosen by the TD),
        'Monitored Object Identifier' =         L,
        'Issue Confirmed Notifications' =       TRUE,
        'Lifetime' =                            0
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' =         L,
        'Time Remaining' =                      0,
        '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' =                   L,
        'Property Identifier' =                 COV_Increment
6.  RECEIVE BACnet-ComplexACK-PDU,
        'Object Identifier' =                   L,
        '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' =     L,
                'Time Remaining' =                  0,
                'List of Values' =                  (the initial Present_Value, new Status_Flags, initial Setpoint, and
initial
                                                    Controlled_Variable_Value)
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  IF (Present_Value is now writable) THEN
        WRITE 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")
10. WAIT **NotificationFailTime**
11. CHECK (verify that no COV notification was transmitted)
12. 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")


13. BEFORE **NotificationFailTime**
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =   (the same value used in step 1),
            'Initiating Device Identifier' =    IUT,
            'Monitored Object Identifier' =     L,
            'Time Remaining' =                  0,
            'List of Values' =                  (the new Present_Value, new Status_Flags, initial Setpoint, and initial
                                                Controlled_Variable_Value)
```

```
14. TRANSMIT BACnet-SimpleACK-PDU
15. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =     (the same value used in step 1),
        'Monitored Object Identifier' =       L
16. RECEIVE BACnet-SimpleACK-PDU
17. IF (Out_Of_Service is writable) THEN
        WRITE L, Out_Of_Service =             FALSE
        RECEIVE BACnet-SimpleACK-PDU
```

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. 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, the vendor shall provide an alternative trigger mechanism to accomplish this task. All of these methods are equally acceptable.

The object identifier of the Loop object being tested is designated as L 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' =       L,
        'Issue Confirmed Notifications' =     TRUE,
        'Lifetime' =                          0
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' =       L,
        'Time Remaining' =                    0,
        'List of Values' =                    (the initial Present_Value, initial Status_Flags, initial Setpoint, and
                                              initial Controlled_Variable_Value)
4.  TRANSMIT BACnet-SimpleACK-PDU
5.  WRITE L, Out_Of_Service = TRUE | WRITE L, 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 4) THEN
        RECEIVE BACnet-SimpleACK-PDU
7.  BEFORE NotificationFailTime
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' = (the same value used in step 1),
            'Initiating Device Identifier' =  IUT,
            'Monitored Object Identifier' =   L,
            'Time Remaining' =                0,
            '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' =       L
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
```

8.3 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 SafetyZone 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.

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 ConfirmedCOVNotification-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)
```

8.4 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.

8.4.1 CHANGE_OF_BITSTRING Tests

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

BACnet Reference Clauses: 12.12.5, 12.12.7, 13.3.1, and 13.8.

Purpose: To verify the correct operation of the Change of Bitstring event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_BITSTRING.

Test Concept: The object begins the test in a NORMAL state. The referenced property is changed to a value that is one of the values designated in List_Of_Bitstring_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The 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.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x = one of the List_Of_Bitstring_Values after the bitmask is
                                   applied)
    ELSE
        MAKE (the referenced property have a value x: x = one of the List_Of_Bitstring_Values after the
                bitmask is applied)
3.  WAIT (Time_Delay)
4.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' = (the Event Enrollment object being tested),
            'Time Stamp' =           (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =             (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =           CHANGE_OF_BITSTRING,
            'Notify Type' =          EVENT | ALARM,
            'AckRequired' =          TRUE | FALSE,
            'From State' =           NORMAL,
            'To State' =             OFFNORMAL,
            ' Event Values' =               referenced-bitstring, Status_Flags
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
7.  VERIFY Event_State = OFFNORMAL
8.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 4, *, *)
9.  IF (Present_Value is writable) THEN
        WRITE (referenced property) = (a value x: x corresponds to a NORMAL state)
    ELSE
        MAKE (the referenced property have a value x: x corresponds to a NORMAL state)
10. WAIT (Time_Delay)
```

```
11. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' = (the Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =         (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          CHANGE_OF_BITSTRING,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          OFFNORMAL,
            'To State' =            NORMAL,
            ' Event Values' =              referenced-bitstring, Status_Flags
12. TRANSMIT BACnet-SimpleACK-PDU
13. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
14. VERIFY Event_State = NORMAL
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 4, *, the timestamp in step 11)
```

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in steps 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

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

BACnet Reference Clauses: 12.6, 12.8, 12.18, 12.20, 13.2,13.3.2, and 13.8.

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_STATE and to intrinsic event reporting for Binary Input, Binary Value, Multi-state Input and Multi-state Value objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to a value that is one of the values designated in List_Of_Values. After the time delay expires the object should enter the OFFNORMAL state and transmit an event notification message. The Present_Value (referenced property) is then changed to a value corresponding to a NORMAL state. After the time delay the object should enter the NORMAL state and transmit an event notification message. For Multi-state Input and Multi-state Value objects there is a special case of the CHANGE_OF_STATE algorithm that applies to transitions to the FAULT state. The test procedure includes a test for this special case.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: x = Alarm_Value for binary objects or one of the Alarm_Values for
                          multi-state objects)
    ELSE
        MAKE (Present_Value have a value x: x = Alarm_Value for binary objects or one of the Alarm_Values
                    for multi-state objects)
3.  WAIT (Time_Delay)
```

```
4.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the Event Enrollment object
                                    being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          CHANGE_OF_STATE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            OFFNORMAL,
            ' Event Values' =               Present_Value, Status_Flags
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
7.  VERIFY Event_State = OFFNORMAL
8.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 4, *, *)
9.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: x corresponds to a NORMAL state)
    ELSE
        MAKE (Present_Value have a value x: x corresponds to a NORMAL state)
10. WAIT (Time_Delay)
11. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                    Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          CHANGE_OF_STATE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          OFFNORMAL,
            'To State' =            NORMAL,
            ' Event Values' =               Present_Value, Status_Flags
12. TRANSMIT BACnet-SimpleACK-PDU
13. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
14. VERIFY Event_State = NORMAL
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 4, *, the timestamp in step 11)
16. IF (the object being tested is a multi-state object that supports intrinsic reporting) THEN
17.     IF (Present_Value is writable) THEN
            WRITE Present_Value = (a value x: x = one of the Fault_Values)
        ELSE
            MAKE (Present_Value have a value x: x = one of the Fault_Values)
18.     WAIT (Time_Delay)
19.     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' =                      (the current local time),
                'Notification Class' =              (the configured notification class),
                'Priority' =                        (the value configured to correspond to a TO-FAULT transition),
                'Event Type' =                      CHANGE_OF_STATE,
            'Notify Type' =                 EVENT | ALARM,
            'AckRequired' =                 TRUE | FALSE,
            'From State' =                  NORMAL,
            'To State' =                    FAULT,
            ' Event Values' =               Present_Value, Status_Flags
20.  TRANSMIT BACnet-SimpleACK-PDU
21.     VERIFY Status_Flags = (FALSE, TRUE, ?, ?)
22.     VERIFY Event_State = FAULT
23.     IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
            VERIFY Event_Time_Stamps = (the timestamp in step 4, the timestamp in step 19, the timestamp in step 11)
24.     IF (the object being tested is a multi-state object that supports intrinsic reporting and Protocol_Revision is
        present and Protocol_Revision ≥ 1) THEN
            VERIFY Reliability = MULTI_STATE_FAULT
25.     IF (Present_Value is writable) THEN
            WRITE Present_Value = (a value x: x corresponds to a NORMAL state)
        ELSE
            MAKE (Present_Value have a value x: x corresponds to a NORMAL state)
26.     WAIT (Time_Delay)
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' =                   (the current local time),
                'Notification Class' =           (the configured notification class),
                'Priority' =                     (the value configured to correspond to a TO-NORMAL transition),
                'Event Type' =                   CHANGE_OF_STATE,
                'Notify Type' =                  EVENT | ALARM,
                'AckRequired' =                  TRUE | FALSE,
                'From State' =                   FAULT,
                'To State' =                     NORMAL,
                ' Event Values' =                Present_Value, Status_Flags
28.  TRANSMIT BACnet-SimpleACK-PDU
29.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
30.     VERIFY Event_State = NORMAL
31.     IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
            VERIFY Event_Time_Stamps = (the timestamp in step 4, the timestamp in step 19, the timestamp in step 26)
```

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in steps 8 and 15 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 4.

8.4.3 CHANGE_OF_VALUE Tests

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

The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Integer or Real datatypes.

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

BACnet Reference Clauses: 12.12, 13.3.2, and 13.8.

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm as applied to numerical datatypes. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.

Test Concept: The object begins the test in a NORMAL state. The referenced property is changed by a value that is less than the Referenced_Property_Increment. The tester verifies that no event notification is transmitted. The referenced property is changed again to a value that differs from the original value by an amount greater than the Referenced_Property_Increment. 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 property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value by less than
                                Referenced_Property_Increment)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value by less than
                Referenced_Property_Increment)
3.  WAIT (Time_Delay + Notification Fail Time)
4.  CHECK (verify that no event notification message is transmitted)
5.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value in step 1 by more than
                                Referenced_Property_Increment)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value in step 1 by more than
                Referenced_Property_Increment)
6.  WAIT (Time_Delay)
7.  BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' = (the Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          CHANGE_OF_VALUE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            NORMAL,
            ' Event Values' =               changed-value, Status_Flags
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10. VERIFY Event_State = 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 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in 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

The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Bitstring datatypes.

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

BACnet Reference Clauses: 12.12, 13.3.2, and 13.8.

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm as applied to Bitstring datatypes. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.

Test Concept: The object begins the test in a NORMAL state. The referenced property is changed to a new value such that none of the bits in the Bitmask are changed. The tester verifies that no event notification is transmitted. The referenced property is changed again to a value that differs in one or more bits that are included in the Bitmask. 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 property shall have a value of TRUE. The Bitmask shall be configured so that at least one but not all bits of the referenced property 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 Event_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value but only in bits that are not
                                      included in Bitmask)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value but only in bits that are not
              included in Bitmask)
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no event notification message is transmitted)
5.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x differs from the initial value in one or more bits included in
                                      Bitmask)
    ELSE
        MAKE (the referenced property have a value x: x differs from the initial value one or more bits included
              in Bitmask)
6.  WAIT (Time_Delay)
7.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =      (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          CHANGE_OF_VALUE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            NORMAL,
            ' Event Values' =           changed-value, Status_Flags
8.  TRANSMIT BACnet-SimpleACK-PDU
9.  VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10. VERIFY Event_State = 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 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in 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

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

BACnet Reference Clauses: 12.7, 12.12, 12.19, 13.2, 13.3.4, and 13.8.

Purpose: To verify the correct operation of the COMMAND_FAILURE algorithm. This test applies to Event Enrollment objects with an Event_Type of COMMAND_FAILURE and to intrinsic event reporting for Binary Output and Multi-State Output objects.

Test Concept: The Feedback_Value (Feedback_Property_Reference) shall be decoupled from the input signal that is normally used to verify the output. Initially Present_Value (referenced property) and Feedback_Value (Feedback_Property_Reference) are in agreement. Present_Value (the referenced property) is changed and an event notification should be transmitted indicating a transition to an OFFNORMAL state. The Feedback_Value (Feedback_Property_Reference) is changed to again agree with the Present_Value (referenced property). A second event notification is transmitted indicating a return to a NORMAL state.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test. The Feedback_Value property shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

In the test description below Present_Value is used as the referenced property and Feedback_Value is used to verify the output. If an Event Enrollment object is being tested these properties shall be replaced by the appropriate property reference.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  VERIFY Status_Flags = (FALSE, FALSE, FALSE, FALSE)
3.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a different value)
    ELSE
        MAKE (Present_Value take on a different value)
4.  WAIT (Time_Delay)
5.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          COMMAND_FAILURE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            OFFNORMAL,
            'Event Values' =                Present_Value, Status_Flags, Feedback_Value
6.  TRANSMIT BACnet-SimpleACK-PDU
7.  VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
8.  VERIFY Event_State = OFFNORMAL
9.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
10. IF (Feedback_Value is writable) THEN
        WRITE Feedback_Value = (a value consistent with Present_Value)
    ELSE
        MAKE (Feedback_Value take on a value consistent with Present_Value)
11. WAIT (Time_Delay)
12. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                  Event Enrollment object being tested),
      'Time Stamp' =          (the current local time),
      'Notification Class' =  (the configured notification class),
      'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
      'Event Type' =          COMMAND_FAILURE,
      'Notify Type' =         EVENT | ALARM,
      'AckRequired' =         TRUE | FALSE,
      'From State' =          OFFNORMAL,
      'To State' =            NORMAL,
      ' Event Values' =       Present_Value, Status_Flags, Feedback_Value
13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15. VERIFY Event_State = 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 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in steps 9 and 16 can have a value that indicates an unspecified time or a time that precedes the timestamp in step 5.

8.4.5 FLOATING_LIMIT Tests

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

BACnet Reference Clauses: 12.12, 12.17, 12.21, 13.2, 13.3.5, and 13.8.

Purpose: To verify the correct operation of the Floating Limit event algorithm. This test applies to Event Enrollment objects with an Event_Type of FLOATING_LIMIT and to Loop objects that support intrinsic reporting. When testing Loop objects both High_Diff_Limit and Low_Diff_Limit shall be replaced by Error_Limit in the test description below.

Test Concept: The object begins the test in a NORMAL state. The referenced property is raised to a value that is below but within Deadband of the high limit. At this point the object should still be in a NORMAL state. The referenced property is raised to a value that is above the high limit. After the time delay expires the object should enter the HIGH_LIMIT state and transmit an event notification message. The referenced property is lowered to a value that is below the high limit but still within Deadband of the limit. The object should remain in the HIGH_LIMIT state. The referenced property is lowered further to a normal value that is not within Deadband of a limit. After the time delay expires the object should enter the NORMAL state and issue an event notification. The same process is repeated to test the low limit.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < (Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < (Setpoint_Reference + High_Diff_Limit))
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no notification message has been transmitted)
5.  VERIFY Event_State = NORMAL
6.  IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x: x > (Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x: x > (Setpoint_Reference + High_Diff_Limit))
7.  WAIT (Time_Delay)
```

```
8.  BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' = (the Loop object being tested or the object referenced by the Event Enrollment
                                         object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          FLOATING_LIMIT,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            HIGH_LIMIT,
            ' Event Values' =               reference-value, Status_Flags, setpoint-value, error-limit,
9.  TRANSMIT BACnet-SimpleACK-PDU
10. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY Event_State = HIGH_LIMIT
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 8, *,  *)
13. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < Setpoint_Reference + High_Diff_Limit))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference + High_Diff_Limit – Deadband)< x < Setpoint_Reference + High_Diff_Limit))
14. WAIT (Time_Delay + Notification Fail Time)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY Event_State = HIGH_LIMIT
17. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
            (Setpoint_Reference - Low_Diff_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit –
Deadband))
    ELSE
        MAKE (the referenced property have a value x:
            (Setpoint_Reference - Low_Diff_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit –
Deadband))
18. WAIT (Time_Delay)
19. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =  IUT,
            'Event Object Identifier' = (the Loop object being tested or the object referenced by the Event Enrollment
                                         object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          FLOATING_LIMIT,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          HIGH_LIMIT,
            'To State' =            NORMAL,
            ' Event Values' =               reference-value, Status_Flags, setpoint-value, error-limit,
20. TRANSMIT BACnet-SimpleACK-PDU
21. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
22. VERIFY Event_State = NORMAL
23. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
```

```
24. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference - Low_Diff_Limit < x < (Setpoint_Reference - Low_Diff_Limit + Deadband))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference - Low_Diff_Limit < x < (Setpoint_Reference - Low_Diff_Limit + Deadband))
25. WAIT (Time_Delay + Notification Fail Time)
26. CHECK (verify that no notification message has been transmitted)
27. VERIFY Event_State = NORMAL
28. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x such x < (Setpoint_Reference -  Low_Diff_Limit))
    ELSE
        MAKE (referenced property have a value x: x < (Setpoint_Reference - Low_Diff_Limit))
29. WAIT (Time_Delay)
30. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' = (the Loop object being tested or the Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          FLOATING_LIMIT,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            LOW_LIMIT,
            ' Event Values' =               reference-value, Status_Flags, setpoint-value, error-limit,
31. TRANSMIT BACnet-SimpleACK-PDU
32. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
33. VERIFY Event_State = LOW_LIMIT
34. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
                (Setpoint_Reference - Low_Limit) < x < (Setpoint_Reference - Low_Limit + Deadband))
    ELSE
        MAKE (the referenced property have a value x:
                (Setpoint_Reference - Low_Limit) < x < (Setpoint_Reference - Low_Limit + Deadband))
36. WAIT (Time_Delay + Notification Fail Time)
37. CHECK (verify that no notification message has been transmitted)
38. VERIFY Event_State = Low_Limit
39. IF (the referenced property is writable) THEN
        WRITE (referenced property) = (a value x:
            (Setpoint_Reference - Low_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit – Deadband))
    ELSE
        MAKE (the referenced property have a value x:
            (Setpoint_Reference - Low_Limit + Deadband) < x < (Setpoint_Reference + High_Diff_Limit – Deadband))
40. WAIT (Time_Delay)
41. BEFORE Notification Fail Time
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' = (the Loop object being tested or the Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          OUT_OF_RANGE,
            'Notify Type' =        EVENT | ALARM,
            'AckRequired' =        TRUE | FALSE,
            'From State' =         LOW_LIMIT,
            'To State' =           NORMAL,
            ' Event Values' =          reference-value, Status_Flags, setpoint-value, error-limit,
42.  TRANSMIT BACnet-SimpleACK-PDU
43.  VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
44.  VERIFY Event_State = NORMAL
45.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
         VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)
```

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in steps 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

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

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 12.23, 13.2, 13.3.6, and 13.8.

Purpose: To verify the correct operation of the OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of OUT_OF_RANGE and to intrinsic event reporting for Analog Input, Analog Output, and Analog Value objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is raised to a value that is below but within Deadband of the high limit. At this point the object should still be in a NORMAL state. The Present_Value (referenced property) is raised to a value that is above the high limit. After the time delay expires the object should enter the HIGH_LIMIT state and transmit an event notification message. The Present_Value (referenced property) is lowered to a value that is below the high limit but still within Deadband of the limit. The object should remain in the HIGH_LIMIT state. The Present_Value (referenced property) is lowered further to a normal value that is not within Deadband of a limit. After the time delay expires the object should enter the NORMAL state and issue an event notification. The same process is repeated to test the low limit.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. For objects using intrinsic reporting the Limit_Enable property shall have a value of TRUE for both HighLimit and LowLimit events. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (High_Limit – Deadband)< x < High_Limit)
    ELSE
        MAKE (Present_Value have a value x: (High_Limit – Deadband)< x < High_Limit)
3.  WAIT (Time_Delay + **Notification Fail Time**)
4.  CHECK (verify that no notification message has been transmitted)
5.  VERIFY Event_State = NORMAL
6.  IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x such x > High_Limit)
    ELSE
        MAKE (Present_Value have a value x: x > High_Limit)
7.  WAIT (Time_Delay)
8.  BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                      Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          OUT_OF_RANGE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            HIGH_LIMIT,
            ' Event Values' =               Present_Value, Status_Flags, Deadband, High_Limit
9.  TRANSMIT BACnet-SimpleACK-PDU
10. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
11. VERIFY Event_State = HIGH_LIMIT
12. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        VERIFY Event_Time_Stamps = (the timestamp in step 8, *, *)
13. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (High_Limit – Deadband)< x < High_Limit)
    ELSE
        MAKE (Present_Value have a value x: (High_Limit – Deadband)< x < High_Limit)
14. WAIT (Time_Delay + **Notification Fail Time**)
15. CHECK (verify that no notification message has been transmitted)
16. VERIFY Event_State = HIGH_LIMIT
17. IF (Present_Value is writable) THEN
        WRITE Present_Value = (a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
    ELSE
        MAKE (Present_Value have a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
18. WAIT (Time_Delay)
19. BEFORE **Notification Fail Time**
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                      Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =          OUT_OF_RANGE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          HIGH_LIMIT,
            'To State' =            NORMAL,
            ' Event Values' =            Present_Value, Status_Flags, Deadband, High_Limit
20.  TRANSMIT BACnet-SimpleACK-PDU
21.  VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
22.  VERIFY Event_State = NORMAL
23.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
         VERIFY Event_Time_Stamps = (the timestamp in step 8, *, the timestamp in step 19)
24.  IF (Present_Value is writable) THEN
         WRITE Present_Value = (a value x: Low_Limit < x < (Low_Limit + Deadband))
     ELSE
         MAKE (Present_Value have a value x: Low_Limit < x < (Low_Limit + Deadband))
25.  WAIT (Time_Delay + **Notification Fail Time**)
26.  CHECK (verify that no notification message has been transmitted)
27.  VERIFY Event_State = NORMAL
28.  IF (Present_Value is writable) THEN
         WRITE Present_Value = (a value x such x < Low_Limit)
     ELSE
         MAKE (Present_Value have a value x: x < Low_Limit)
29.  WAIT (Time_Delay)
30.  BEFORE **Notification Fail Time**
         RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                   Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =        (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          OUT_OF_RANGE,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          NORMAL,
            'To State' =            LOW_LIMIT,
            ' Event Values' =            Present_Value, Status_Flags, Deadband, Low_Limit
31.  TRANSMIT BACnet-SimpleACK-PDU
32.  VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
33.  VERIFY Event_State = LOW_LIMIT
34.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
         VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 19)
35.  IF (Present_Value is writable) THEN
         WRITE Present_Value = (a value x: Low_Limit < x < (Low_Limit + Deadband))
     ELSE
         MAKE (Present_Value have a value x: Low_Limit <  x < (Low_Limit + Deadband))
36.  WAIT (Time_Delay + **Notification Fail Time**)
37.  CHECK (verify that no notification message has been transmitted)
38.  VERIFY Event_State = LOW_LIMIT
39.  IF (Present_Value is writable) THEN
         WRITE Present_Value = (a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
     ELSE
         MAKE (Present_Value have a value x: (Low_Limit + Deadband) < x < (High_Limit – Deadband))
40.  WAIT (Time_Delay)
41.  BEFORE **Notification Fail Time**
         RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (any valid process ID),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' = (the intrinsic reporting object being tested or the object referenced by the
                                   Event Enrollment object being tested),
            'Time Stamp' =          (the current local time),
            'Notification Class' =        (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-NORMAL transition),
```

'Event Type' =	OUT_OF_RANGE,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	LOW_LIMIT,
'To State' =	NORMAL,
' Event Values' =	Present_Value, Status_Flags, Deadband, Low_Limit

42. TRANSMIT BACnet-SimpleACK-PDU
43. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
44. VERIFY Event_State = NORMAL
45. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
    VERIFY Event_Time_Stamps = (the timestamp in step 30, *, the timestamp in step 41)

Notes to Tester: The 'Message Text' parameter is omitted in the test description because it is optional. The IUT may include this parameter in the notification messages. The time stamps indicated by "*" in steps 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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 12.12, 12.25, 13.2, 13.3.7, and 13.8.

Purpose: To verify the correct operation of the BUFFER_READY event algorithm. This test applies to Trend Log objects that support intrinsic notification and to Event Enrollment objects with an Event_Type of BUFFER_READY.

Test Concept: The object that performs the notification ("the notifying object") begins the test in a NORMAL state, with no records stored in the object containing the buffer ("the buffer object"). The buffer object acquires the number of records specified by Records_Since_Notification, at which time the notifying object performs a TO-NORMAL transition and sends BUFFER_READY notifications.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-NORMAL transition. The notifying 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 IUT shall be set to TRUE.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (buffer object collect number of records specified by Notification_Threshold)
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 object referenced by the Event Enrollment object being tested),
'Time Stamp' =	(any appropriate BACnetTimeStamp value),
'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 buffer object), (any BACnetDateTime), (current local BACnetDateTime)

4. TRANSMIT BACnet-SimpleACK-PDU
5. MAKE (buffer object collect number of records specified by Notification_Threshold)

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 object referenced by the Event Enrollment object being tested),
'Time Stamp' =	(any appropriate BACnetTimeStamp value),
'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 buffer object), (current local BACnetDateTime sent in step 3), (current local BACnetDateTime)

7. TRANSMIT BACnet-SimpleACK-PDU

8.4.8 CHANGE_OF_LIFE_SAFETY Tests

8.4.8.1 NORMAL to OFFNORMAL Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

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. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to one of the values designated in List_Of_Alarm_Values. After the time delay expires, the object should enter the OFFNORMAL state and transmit an event notification message.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (Present_Value have a value x such that x corresponds to an OFFNORMAL state)
3. WAIT Time_Delay
4. BEFORE **Notification Fail Time**
5. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,

```
          'To State' =                  OFFNORMAL,
          ' Event Values' =             Present_Value,Mode,Status_Flags, Operation_Expected
6.   TRANSMIT BACnet-SimpleACK-PDU
7.   VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
8.   VERIFY Event_State = OFFNORMAL
9.   VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
```

8.4.8.2 OFFNORMAL to NORMAL Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and NORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in an OFFNORMAL state. The Present_Value (referenced property) 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.

Test Steps:

```
1.   MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
2.   WAIT Time_Delay
3.   IF (latching is supported) THEN
4.        CHECK (Event_State = OFFNORMAL)
5         MAKE (the object reset)
6.        BEFORE **Notification Fail Time**
7.        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' =                  (the current local time),
               'Notification Class' =          (the configured notification class),
               'Priority' =                    (the value configured to correspond to a TO-OFFNORMAL transition),
               'Event Type' =                  CHANGE_OF_LIFE_SAFETY,
               'Notify Type' =                 EVENT | ALARM,
               'AckRequired' =                 TRUE | FALSE,
               'From State' =                  OFFNORMAL,
               'To State' =                    NORMAL,
               ' Event Values' =               Present_Value, Mode, Status_Flags, Operation_Expected
8.        TRANSMIT BACnet-SimpleACK-PDU,
9.        VERIFY Status_Flags = (FALSE, FALSE, ?, ?),
10.       VERIFY Event_State = NORMAL,
11.       VERIFY Event_Time_Stamps = (the timestamp in step 7, *, *)
     ELSE
12.       BEFORE **Notification Fail Time**
13.            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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,
'To State' =	NORMAL,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

14. TRANSMIT BACnet-SimpleACK-PDU
15. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
16. VERIFY Event_State = NORMAL
17. VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)

8.4.8.3 NORMAL to LIFE_SAFETY_ALARM Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13.9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and LIFE_SAFETY_ALARM event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to one of the values designated in Life_Safety_Alarm_Values. 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (Present_Value have a value x such that x corresponds to a LIFE_SAFETY_ALARM state)
3. WAIT Time_Delay
4. BEFORE **Notification Fail Time**
5. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	LIFE_SAFETY_ALARM,
' Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)

```
8.    VERIFY Event_State = LIFE_SAFETY_ALARM
9.    VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
```

8.4.8.4 LIFE_SAFETY_ALARM to NORMAL Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13.9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and NORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in an LIFE_SAFETY_ALARM state. The Present_Value (referenced property) 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference.

Test Steps:

```
1.    MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
2.    WAIT Time_Delay
3.    IF (latching is supported) THEN
4.        CHECK (Event_State = LIFE_SAFETY_ALARM)
5.        MAKE (the object reset)
6.        BEFORE **Notification Fail Time**
7.            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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    LIFE_SAFETY_ALARM,
                'To State' =                      NORMAL,
                ' Event Values' =                 Present_Value, Mode, Status_Flags, Operation_Expected
8.        TRANSMIT BACnet-SimpleACK-PDU
9.        VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10.       VERIFY Event_State = NORMAL
11.       VERIFY Event_Time_Stamps = (the timestamp in step 7, *, *)
      ELSE
12.       BEFORE **Notification Fail Time**
13.           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' =                    (the current local time),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                  CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  LIFE_SAFETY_ALARM,
                'To State' =                    NORMAL,
                ' Event Values' =               Present_Value, Mode, Status_Flags, Operation_Expected
14.     TRANSMIT BACnet-SimpleACK-PDU
15.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
16.     VERIFY Event_State = NORMAL
17.     VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)
```

8.4.8.5 LIFE_SAFETY_ALARM to OFFNORMAL Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and OFFNORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in an LIFE_SAFETY_ALARM state. The Present_Value (referenced property) 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 property shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested Present_Value should be replaced by the appropriate property reference. 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. 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.  MAKE (Present_Value have a value x such that x corresponds to an OFFNORMAL state)
2.  WAIT Time_Delay
3.  IF (latching is supported) THEN
4.      CHECK (Event_State = OFFNORMAL)
5.      MAKE (the object reset)
6.      BEFORE **Notification Fail Time**
7.   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' =          (the current local time),
            'Notification Class' =          (the configured notification class),
            'Priority' =            (the value configured to correspond to a TO-OFFNORMAL transition),
            'Event Type' =          CHANGE_OF_LIFE_SAFETY,
            'Notify Type' =         EVENT | ALARM,
            'AckRequired' =         TRUE | FALSE,
            'From State' =          LIFE_SAFETY_ALARM,
            'To State' =            OFFNORMAL,
            ' Event Values' =               Present_Value, Mode, Status_Flags, Operation_Expected
8.      TRANSMIT BACnet-SimpleACK-PDU
```

```
9.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10.     VERIFY Event_State = OFFNORMAL
11.     VERIFY Event_Time_Stamps = (the timestamp in step 7, *, *)
    ELSE
12.     BEFORE **Notification Fail Time**
13.         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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    LIFE_SAFETY_ALARM,
                'To State' =                      OFFNORMAL,
                ' Event Values' =                 Present_Value, Mode, Status_Flags, Operation_Expected
14.     TRANSMIT BACnet-SimpleACK-PDU
15.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
16.     VERIFY Event_State = OFFNORMAL
17.     VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)
```

8.4.8.6 OFFNORMAL to LIFE_SAFETY_ALARM Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and LIFE_SAFETY_ALARM event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in an OFFNORMAL state. The Present_Value (referenced property) 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in an OFFNORMAL state at the start of the test. In the test description below Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference. 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.  MAKE (Present_Value have a value x such that x corresponds to an LIFE_SAFETY state)
2.  WAIT Time_Delay
3.  IF (latching is supported) THEN
4.      CHECK (Event_State = LIFE_SAFETY_ALARM)
5.      MAKE (the object reset)
6.      BEFORE **Notification Fail Time**
7.          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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    OFFNORMAL,
                'To State' =                      LIFE_SAFETY_ALARM,
                ' Event Values' =                 Present_Value, Mode, Status_Flags, Operation_Expected
8.      TRANSMIT BACnet-SimpleACK-PDU
9.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
10.     VERIFY Event_State = LIFE_SAFETY_ALARM
11.     VERIFY Event_Time_Stamps = (the timestamp in step 7, *, *)
    ELSE
12.     BEFORE **Notification Fail Time**
13.         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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    OFFNORMAL,
                'To State' =                      LIFE_SAFETY_ALARM,
                ' Event Values' =                 Present_Value, Mode, Status_Flags, Operation_Expected
14.     TRANSMIT BACnet-SimpleACK-PDU
15.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
16.     VERIFY Event_State = LIFE_SAFETY_ALARM
17.     VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)
```

8.4.8.7 Mode Transition Tests when Event State is Maintained

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

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. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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 Present_Value designated in the List_Of_Alarm_Values. The object should immediately 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, 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.

Test Steps:

```
1.  CHECK (Event_State = NORMAL)
2.  MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
3.  BEFORE **Notification Fail Time**
4.      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' =                      (the current local time),
            'Notification Class' =              (the configured notification class),
            'Priority' =                        (the value configured to correspond to a TO-NORMAL transition),
            'Event Type' =                      CHANGE_OF_LIFE_SAFETY,
            'Notify Type' =                     EVENT | ALARM,
            'AckRequired' =                     TRUE | FALSE,
            'From State' =                      NORMAL,
            'To State' =                        NORMAL,
            'Event Values' =                    Present_Value, Mode, Status_Flags, Operation_Expected
5.  TRANSMIT BACnet-SimpleACK-PDU
6.  VERIFY Status_Flags =                       (FALSE, FALSE, ?, ?)
7.  VERIFY Event_State =                        NORMAL
8.  VERIFY Event_Time_Stamps =                  (the timestamp in step 4, *, *)
9.  MAKE (Present_Value have a value x such that x corresponds to an OFFNORMAL state)
10. MAKE (Mode = different value that maintains Event_State as OFFNORMAL)
11. BEFORE **Notification Fail Time**
12.     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' =                          (the current local time),
            'Notification Class' =                  (the configured notification class),
            'Priority' =                            (the value configured to correspond to a TO-OFFNORMAL
                                                    transition),
            'Event Type' =                          CHANGE_OF_LIFE_SAFETY,
            'Notify Type' =                         EVENT | ALARM,
            'AckRequired' =                         TRUE | FALSE,
            'From State' =                          OFFNORMAL,
            'To State' =                            OFFNORMAL,
            'Event Values' =                        Present_Value, Mode, Status_Flags, Operation_Expected
13. TRANSMIT BACnet-SimpleACK-PDU
```

14. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
15. VERIFY Event_State = OFFNORMAL
16. VERIFY Event_Time_Stamps = (the timestamp in step 12, *, *)
17. MAKE (Present_Value have a value x such that x corresponds to a LIFE_SAFETY_ALARM state)
18. MAKE (Mode = different value that maintains Event_State = LIFE_SAFETY_ALARM)
19. BEFORE **Notification Fail Time**
20. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,
'To State' =	OFFNORMAL,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

21. TRANSMIT BACnet-SimpleACK-PDU
22. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
23. VERIFY Event_State = OFFNORMAL
24. VERIFY Event_Time_Stamps = (the timestamp in step 20, *, *)

8.4.8.8 NORMAL to OFFNORMAL Mode Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

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 by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (Mode a different value that forces the state to OFFNORMAL)
3. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,

```
                'Notify Type' =                EVENT | ALARM,
                'AckRequired' =                TRUE | FALSE,
                'From State' =                 NORMAL,
                'To State' =                   OFFNORMAL,
                'Event Values' =               Present_Value, Mode, Status_Flags, Operation_Expected
4.  TRANSMIT BACnet-SimpleACK-PDU
5.  VERIFY Status_Flags =                      (TRUE, FALSE, ?, ?)
6.  VERIFY Event_State =                       OFFNORMAL
7.  VERIFY Event_Time_Stamps =                 (the timestamp in step 3, *, *)
```

8.4.8.9 OFFNORMAL to NORMAL Mode Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and NORMAL event states by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a OFFNORMAL state at the start of the test. In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference. 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.  MAKE (Mode a different value that forces the state to NORMAL)
2.  IF (latching is supported) THEN
3.      CHECK (Event_State = OFFNORMAL)
4.      MAKE (the object reset)
5.      BEFORE **Notification Fail Time**
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' =                  (the current local time),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                  CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  OFFNORMAL,
                'To State' =                    NORMAL,
                'Event Values' =                Present_Value, Mode, Status_Flags, Operation_Expected
7.      TRANSMIT BACnet-SimpleACK-PDU
8.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
9.      VERIFY Event_State = NORMAL
10.     VERIFY Event_Time_Stamps = (the timestamp in step 6, *, *)
    ELSE
11.     BEFORE **Notification Fail Time**
```

12. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	OFFNORMAL,
'To State' =	NORMAL,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

13. TRANSMIT BACnet-SimpleACK-PDU
14. VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15. VERIFY Event_State = NORMAL
16. VERIFY Event_Time_Stamps = (the timestamp in step 12, *,*)

8.4.8.10 NORMAL to LIFE_SAFETY_ALARM Mode Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and LIFE_SAFETY_ALARM event states by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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.

In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (Mode a different value that forces the state to LIFE_SAFETY_ALARM)
3. BEFORE **Notification Fail Time**
4. 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' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-OFFNORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	LIFE_SAFETY_ALARM,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

5. TRANSMIT BACnet-SimpleACK-PDU
6. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
7. VERIFY Event_State = LIFE_SAFETY_ALARM
8. VERIFY Event_Time_Stamps = (the timestamp in step 4, *, *)

8.4.8.11 LIFE_SAFETY_ALARM to NORMAL Mode Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and NORMAL event states by changing the Mode property. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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 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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference. 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.  MAKE (Mode a different value that forces the state to NORMAL)
2.  IF (latching is supported) THEN
3.      CHECK (Event_State = LIFE_SAFETY_ALARM)
4.      MAKE (the object reset)
5.      BEFORE **Notification Fail Time**
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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    LIFE_SAFETY_ALARM,
                'To State' =                      NORMAL,
                'Event Values' =                  Present_Value, Mode, Status_Flags, Operation_Expected
7.      TRANSMIT BACnet-SimpleACK-PDU
8.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
9.      VERIFY Event_State = NORMAL
10.     VERIFY Event_Time_Stamps = (the timestamp in step 6, *, *)
    ELSE
11.     BEFORE **Notification Fail Time**
12.         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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =               EVENT | ALARM,
                'AckRequired' =               TRUE | FALSE,
                'From State' =                LIFE_SAFETY_ALARM,
                'To State' =                  NORMAL,
                'Event Values' =              Present_Value, Mode, Status_Flags, Operation_Expected
13.     TRANSMIT BACnet-SimpleACK-PDU
14.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15.     VERIFY Event_State = NORMAL
16.     VERIFY Event_Time_Stamps = (the timestamp in step 12, *, *)}
```

8.4.8.12 LIFE_SAFETY_ALARM to OFFNORMAL Mode Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and OFFNORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference. 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.  MAKE (Mode a different value that forces the state to OFFNORMAL)
2.  IF (latching is supported) THEN
3.      CHECK (Event_State = OFFNORMAL)
4.      MAKE (the object reset)
5.      BEFORE **Notification Fail Time**
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' =                 (the current local time),
                'Notification Class' =         (the configured notification class),
                'Priority' =                   (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                 CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                EVENT | ALARM,
                'AckRequired' =                TRUE | FALSE,
                'From State' =                 LIFE_SAFETY_ALARM,
                'To State' =                   OFFNORMAL,
                'Event Values' =               Present_Value, Mode, Status_Flags, Operation_Expected
7.      TRANSMIT BACnet-SimpleACK-PDU
8.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
9.      VERIFY Event_State = OFFNORMAL
10.     VERIFY Event_Time_Stamps = (the timestamp in step 6, *, *)
    ELSE
11.     BEFORE **Notification Fail Time**
```

```
12.         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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                    LIFE_SAFETY_ALARM,
                'To State' =                      OFFNORMAL,
                'Event Values' =                  Present_Value, Mode, Status_Flags, Operation_Expected
13.     TRANSMIT BACnet-SimpleACK-PDU
14.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15.     VERIFY Event_State = OFFNORMAL
16.     VERIFY Event_Time_Stamps = (the timestamp in step 12, *, *)
```

8.4.8.13 OFFNORMAL to LIFE_SAFETY_ALARM Mode Transition Test

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

BACnet Reference Clauses: 12.21.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning from the OFFNORMAL and LIFE_SAFETY_ALARM event states by changing the Mode property. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

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 property shall have a value of TRUE. The event-generating objects shall be in a LIFE_SAFETY_ALARM state at the start of the test. In the test description below, Present_Value is used as the referenced property. If an Event Enrollment object is being tested, Present_Value should be replaced by the appropriate property reference. 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.  MAKE (Mode a different value that forces the state to LIFE_SAFETY_ALARM)
2.  IF (latching is supported) THEN
3.      CHECK (Event_State = OFFNORMAL)
4.      MAKE (the object reset)
5.      BEFORE **Notification Fail Time**
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' =                    (the current local time),
                'Notification Class' =            (the configured notification class),
                'Priority' =                      (the value configured to correspond to a TO OFFNORMAL transition),
                'Event Type' =                    CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                   EVENT | ALARM,
                'AckRequired' =                   TRUE | FALSE,
                'From State' =                  LIFE_SAFETY_ALARM,
                'To State' =                    OFFNORMAL,
                'Event Values' =                Present_Value, Mode, Status_Flags, Operation_Expected
7.      TRANSMIT BACnet-SimpleACK-PDU
8.      VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
9.      VERIFY Event_State = OFFNORMAL
10.     VERIFY Event_Time_Stamps = (the timestamp in step 6, *, *)
    ELSE
11.     BEFORE Notification Fail Time
12.         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' =                  (the current local time),
                'Notification Class' =          (the configured notification class),
                'Priority' =                    (the value configured to correspond to a TO-OFFNORMAL transition),
                'Event Type' =                  CHANGE_OF_LIFE_SAFETY,
                'Notify Type' =                 EVENT | ALARM,
                'AckRequired' =                 TRUE | FALSE,
                'From State' =                  LIFE_SAFETY_ALARM,
                'To State' =                    OFFNORMAL,
                'Event Values' =                Present_Value, Mode, Status_Flags, Operation_Expected
13.     TRANSMIT BACnet-SimpleACK-PDU
14.     VERIFY Status_Flags = (FALSE, FALSE, ?, ?)
15.     VERIFY Event_State = OFFNORMAL
16.     VERIFY Event_Time_Stamps = (the timestamp in step 12, *, *)
```

8.4.8.14 TO-FAULT Transition Tests

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.8 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and FAULT event states. It applies to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

Test Concept: The object begins the test in a NORMAL state. The Present_Value (referenced property) is changed to one of the values designated in List_Of_Fault_Values. After the time delay expires, the object should enter the FAULT state and transmit an event notification message. This test also incorporates the return transition from FAULT to NORMAL.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL, TO-FAULT and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating objects shall be in a NORMAL state at the start of the test.

In the test description below, Present_Value is used as the referenced property.

Test Steps:

```
1.  VERIFY Event_State = NORMAL
2.  MAKE (Present_Value have a value x such that x corresponds to a FAULT state)
3.  WAIT Time_Delay
4.  BEFORE Notification Fail Time
5.      RECEIVE ConfirmedEventNotification-Request,

            'Process Identifier' =              (any valid process ID),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the intrinsic reporting object being tested),
            'Time Stamp' =                      (the current local time),
```

'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-FAULT transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	NORMAL,
'To State' =	FAULT,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

6. TRANSMIT BACnet-SimpleACK-PDU
7. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
8. VERIFY Event_State = FAULT
9. VERIFY Event_Time_Stamps = (the timestamp in step 5, *, *)
10. MAKE (Present_Value have a value x such that x corresponds to a NORMAL state)
11. WAIT Time_Delay
12. BEFORE **Notification Fail Time**
13. RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the intrinsic reporting object being tested),
'Time Stamp' =	(the current local time),
'Notification Class' =	(the configured notification class),
'Priority' =	(the value configured to correspond to a TO-NORMAL transition),
'Event Type' =	CHANGE_OF_LIFE_SAFETY,
'Notify Type' =	EVENT | ALARM,
'AckRequired' =	TRUE | FALSE,
'From State' =	FAULT,
'To State' =	NORMAL,
'Event Values' =	Present_Value, Mode, Status_Flags, Operation_Expected

14. TRANSMIT BACnet-SimpleACK-PDU
15. VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
16. VERIFY Event_State = NORMAL
17. VERIFY Event_Time_Stamps = (the timestamp in step 13, *, *)

8.4.9 EXTENDED Tests

Dependencies: None

BACnet Reference Clauses: 13.2 and 13.3

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

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

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for whichever transition shall be used for the test. The Issue_Confirmed_Notifications property shall have a value of TRUE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps:

1. VERIFY Event_State = NORMAL
2. MAKE (the event generating object transition)
3. WAIT (Time_Delay)
4. BEFORE **Notification Fail Time**
   RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(any valid process ID),
'Initiating Device Identifier' =	IUT,

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

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

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.

8.5.1 CHANGE_OF_BITSTRING Tests

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

BACnet Reference Clauses: 12.12, 13.3.1, and 13.9.

Purpose: To verify the correct operation of the Change of Bitstring event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_BITSTRING.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.1 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

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

BACnet Reference Clauses: 12.6, 12.8, 12.18, 12.20, 13.2,13.3.2, and 13.9.

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_STATE and to intrinsic event reporting for Binary Input, Binary Value, Multi-state Input and Multi-state Value objects.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.2 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

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

The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Integer or Real datatypes.

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

BACnet Reference Clauses: 12.12, 13.3.3, and 13.9.

Purpose: To verify the correct operation of the CHANGE_OF_VALUE event algorithm as applied to numerical datatypes. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.

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 property shall have a value of FALSE. The event-generating object shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.1 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

The test in this clause applies to use of the CHANGE_OF_VALUE algorithm applied to Bitstring datatypes.

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

BACnet Reference Clauses: 12.12, 13.3.3, and 13.9.

Purpose: To verify the correct operation of the CHANGE_OF_STATE event algorithm. This test applies to Event Enrollment objects with an Event_Type of CHANGE_OF_VALUE.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.3.2 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

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

BACnet Reference Clauses: 12.7, 12.12, 12.19 13.2, 13.3.4, and 13.9.

Purpose: To verify the correct operation of the COMMAND_FAILURE algorithm. It applies to Event Enrollment objects with an Event_Type of COMMAND_FAILURE and to intrinsic event reporting for Binary Output and Multi-State Output objects.

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating object shall be in a NORMAL state at the start of the test. The Feedback_Value property shall be decoupled from the input signal that is normally used to verify the output so that it can be independently manipulated.

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

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

BACnet Reference Clauses: 12.12, 12.17, 13.2, 13.3.5, and 13.9.

Purpose: To verify the correct operation of the Floating Limit event algorithm. This test applies to Event Enrollment objects with an Event_Type of FLOATING_LIMIT and to Loop objects that support intrinsic reporting. When testing Loop objects both High_Diff_Limit and Low_Diff_Limit shall be replaced by Error_Limit in the test description below

Configuration Requirements: The IUT shall be configured such that the Event_Enable property has a value of TRUE for the TO-OFFNORMAL and TO-NORMAL transitions. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.5 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

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

BACnet Reference Clauses: 12.2, 12.3, 12.4, 12.12, 13.2, 13.3.6, and 13.9.

Purpose: To verify the correct operation of the OUT_OF_RANGE event algorithm. This test applies to Event Enrollment objects with an Event_Type of OUT_OF_RANGE and to intrinsic event reporting for Analog Input, Analog Output, and Analog Value objects.

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 HighLimit and LowLimit events. The Issue_Confirmed_Notifications property shall have a value of FALSE. The event-generating objects shall be in a NORMAL state at the start of the test.

Test Steps: The test steps for this test case are identical to the test steps in 8.4.6 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

Dependencies: ReadProperty Service Execution Tests, 9.18.

BACnet Reference Clauses: 12.12, 12.25, 13.2, 13.3.7, and 13.9.

Purpose: To verify the correct operation of the BUFFER_READY event algorithm. This test applies to Trend Log objects that support intrinsic notification and to Event Enrollment objects with an Event_Type of BUFFER_READY.

The test concept, configuration requirements and test steps are identical to those in 8.4.7, 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 CHANGE_OF_LIFE_SAFETY TESTS

8.5.8.1 NORMAL to OFFNORMAL Transition Test

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.13 and Figure 13-9.

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. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.1, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and NORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.2, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and LIFE_SAFETY_ALARM event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.3, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and NORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.4, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and OFFNORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.5, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and LIFE_SAFETY_ALARM event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.6, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

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. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.7, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

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 by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.8, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the OFFNORMAL and NORMAL event states by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.9, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and LIFE_SAFETY_ALARM event states by changing the Mode. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.10, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and NORMAL event states by changing the Mode property. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.11, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the LIFE_SAFETY_ALARM and OFFNORMAL event states. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.12, except that the Issue_Confirmed_Notifications property 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

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning from the OFFNORMAL and LIFE_SAFETY_ALARM event states by changing the Mode property. It applies to Event Enrollment objects with an Event_Type of CHANGE_OF_LIFE_SAFETY and to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.13, except that the Issue_Confirmed_Notifications property shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.5.8.14 TO-FAULT Transition Tests

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

BACnet Reference Clauses: 12.12.7, 12.15, 12.16, 13.3.8, 13.9 and Figure 13-9.

Purpose: This test case verifies the correct operation of the CHANGE_OF_LIFE_SAFETY event algorithm for objects transitioning between the NORMAL and FAULT event states. It applies to intrinsic event reporting for Life Safety Point and Life Safety Zone objects.

The test concept, configuration requirements and test steps are identical to those in 8.4.8.14, except that the Issue_Confirmed_Notifications property shall have a value of FALSE, UnconfirmedEventNotification requests are issued instead of ConfirmedEventNotification requests and the TD does not acknowledge receiving the notifications.

8.6 GetAlarmSummary Service Initiation Tests

Purpose: To verify that the IUT can initiate GetAlarmSummary service requests.

Dependencies: None.

BACnet Reference Clause: 13.10.

Test Steps:

1. RECEIVE GetAlarmSummary-Request
2. TRANSMIT BACnet-ComplexACK-PDU,
   'List of Alarm Summaries' = (an empty list)

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)

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)

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
```

8.11 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 ≥ 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 ≤ '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 ≥ 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 ≤ 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 ≥ -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 ≥ -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' = (two 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' = (two 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' = (two or more elements with the correct datatype to removefrom 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' = (two or more elements with the correct datatype to remove from the list)
2. TRANSMIT BACnet-SimpleACK-PDU

8.16CreateObject 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 more than one BACnetPropertyValues consistent with the specified object type)
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 more than one BACnetPropertyValues consistent with the specified object type)
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.

Test Steps:

1. RECEIVE ReadProperty-Request,
   'Object Identifier' = (any object),
   'Property Identifier' = (any valid non-array property of the specified object)

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.

Test Steps:

```
1.  RECEIVE ReadProperty-Request,
        'Object Identifier' = (any object),
        'Property Identifier' = (any valid array property of the specified object),
        'Array Index' = (any valid array index for the specified property)
```

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),
                                         (any array property identifier, any array index, any relation specifier, any valid
                                          comparison value)
                                        )
```

8.19.2 Reading Specific Properties of Objects that Meet the Selection Criteria

Purpose: To verify that the IUT can initiate a ReadPropertyConditional service request in which objects are chosen based on some selection criteria and a set of properties to be read is specified.

Test Steps:

```
1.  RECEIVE ReadPropertyConditional-Request,
        'Selection Logic' =             OR,
        'List of Selection Criteria' =  ((any non-array property identifier, any relation specifier, any valid comparison
                                          value),
                                         (any array property identifier, any array index, any relation specifier, any valid
                                          comparison value)
                                        ),
        'List of Property References' = (two or more properties that correspond to objects meeting the selection criteria)
```

8.20 ReadPropertyMultiple Service Initiation Tests

Purpose: This clause defines the tests necessary to demonstrate support for initiating ReadPropertyMultiple service requests. At least one of the combinations defined in 8.20.2 through 8.20.4 needs to be supported in order to claim the ability to initiate the ReadPropertyMultiple service.

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.)
```

8.21 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),
            '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)
```

8.22 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)
```

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),
        'Array Index' =          (any valid array index for the specified property),
        'Property Value' =       (any value appropriate to the specified property)
```

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 ≤ X ≤ 16)
```

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 non-array property of the specified object),
 'Property Value' = (any value appropriate to the specified property)

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 non-array property of the specified object that was not previously used),
 'Property Value' = (any value appropriate to the specified property)

-- … (Any number of properties ≥ 2 is acceptable.)

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 non-array property of the specified object),
 'Property Value' = (any value appropriate to the specified property)

-- … (Any number of (object, property, value) tuples ≥ 2 is acceptable.)

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 non-array 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 non-array 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.)

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),
            '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.

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)

8.25ConfirmedPrivateTransfer 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 ConfirmedTextMessage-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 ConfirmedTextMessage-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 ≤ X ≤ '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)
```

8.33I-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.

8.34Who-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: $1 \le X \le$ 'Device Instance Range High Limit'),
   'Device Instance Range High Limit' = (any integer Y: 'Device Instance Range Low Limit' $\le Y \le$ 4,194,303)

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_{IUT})
2. TRANSMIT VT-Open-ACK,
   'Remote VT Session Identifier' = (any valid session identifier, S_{TD})
3. VERIFY (the IUT's Device object), Active_VT_Sessions = (the newly established session)

If the IUT is capable of having more than one active VT session the tester may optionally decide to leave the session created by this test active while proceeding with the tests in 8.36.2. Otherwise the VT session should be closed. This may be accomplished as part of the test for verifying the initiation of the VT-Close service request (8.37) or execution of the VT-Close service (9.35.1).

8.36.2 Other VT-classes

Purpose: To verify that the IUT can initiate VT-Open service requests for all supported optional VT-classes and that the open VT sessions are reflected in the Active_VT_Sessions property of the IUT's Device object. If the IUT does not support VT classes other than DEFAULT_TERMINAL then this test shall be omitted.

Test Concept: An attempt is made to open a new VT session for each supported VT-class. The previously opened sessions are left open until all supported VT-classes are tested or until the maximum number of open sessions supported is reached. If the maximum number of open sessions is reached before all of the VT-Classes have been tested, a session is closed before moving on to the next VT-class. Either the IUT or the TD can initiate the closure of a session.

Test Steps:

```
1.  REPEAT X = (all supported VT classes except DEFAULT_TERMINAL) DO {
        RECEIVE VT-Open-Request,
            'VT-class' =                        X,
            'Local VT Session Identifier' =     (any valid unique session identifier, $S_{IUT,i}$)
        TRANSMIT VT-Open-ACK,
            'Remote VT Session Identifier' =    (any valid unique session identifier, $S_{TD,i}$)
        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
            )
    }
```

8.37VT-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, $S_{TD}$, 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 8.38.1applies. 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, $S_{IUT}$)
2.  TRANSMIT VT-Open-ACK,
        'Remote VT Session Identifier' =    (any valid session identifier, $S_{TD}$)
3.  MAKE (the IUT initiate a VT-Data-Request)
4.  RECEIVE VT-Data-Request,
        'VT-session Identifier' =           $S_{TD,}$
        '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' =           $S_{TD,}$
        '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' =           $S_{TD,}$
        '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' =  $S_{IUT}$
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_{TD})
2. RECEIVE VT-Open-ACK,
   'Remote VT Session Identifier' = (any valid session identifier, S_{IUT})
3. TRANSMIT VT-Data-Request,
   'VT-session Identifier' = S_{IUT},
   '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' = $S_{TD,}$
   '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' = $S_{TD,}$
   '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' = $S_{TD,}$
    '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_{IUT}
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_{IUT}.

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 step1)

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send 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:

1. MAKE (a change that triggers the detection of an alarm event in the IUT)
2. RECEIVE ConfirmedEventNotification-Request,

'Process Identifier' =	(the process identifier configured for this event),
'Initiating Device Identifier' =	IUT,
'Event Object Identifier' =	(the object detecting the alarm),
'Time Stamp' =	(the current time or sequence number),
'Notification Class' =	(the notification class configured for this event),
'Priority' =	(the priority configured for this event),
'Event Type' =	(any valid event type),
'Notify Type' =	ALARM,
'AckRequired' =	TRUE,
'From State' =	NORMAL,
'To State' =	(any appropriate non-normal event state),
'Event Values' =	(the values appropriate to the event type)

```
3.  RECEIVE
        DESTINATION =                       (at least one device other than the TD),
        SOURCE =                            IUT,
        ConfirmedEventNotification-Request,
        'Process Identifier' =              (the process identifier configured for this event),
        'Initiating Device Identifier' =    IUT,
        'Event Object Identifier' =         (the object detecting the alarm),
        'Time Stamp' =                      (the current time or sequence number),
        'Notification Class' =              (the notification class configured for this event),
        'Priority' =                        (the priority configured for this event),
        'Event Type' =                      (any valid event type),
        'Notify Type' =                     ALARM,
        'AckRequired' =                     TRUE,
        'From State' =                      NORMAL,
        'To State' =                        (any appropriate non-normal event state),
        'Event Values' =                    (the values appropriate to the event type)
4.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
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' =        (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)
6.  RECEIVE BACnet-Simple-ACK-PDU
7.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the time or sequence number from the notification in step 2),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event),
            'Event Type' =                  (the event type included in step 2),
            'Notify Type' =                 ACK_NOTIFICATION,
            'To State' =                    (the 'To State' used in step 3)
    ELSE
        RECEIVE ConfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the time or sequence number from the notification in step 2),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event),
            'Event Type' =                  (the event type included in step 2),
            'Notify Type' =                 ACK_NOTIFICATION,
            'To State' =                    (the 'To State' used in step 3)
8.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE
            DESTINATION =                   (at least one device other than the TD),
            SOURCE =                        IUT,
            ConfirmedEventNotification-Request,
            'Process Identifier' =          (the process identifier configured for this event),
            'Initiating Device Identifier' = IUT,
            'Event Object Identifier' =     (the object detecting the alarm),
            'Time Stamp' =                  (the time or sequence number from the notification in step 2),
            'Notification Class' =          (the notification class configured for this event),
            'Priority' =                    (the priority configured for this event),
            'Event Type' =                  (the event type included in step 2),
            'Notify Type' =                 ACK_NOTIFICATION,
            'To State' =                        (the 'To State' used in step 3)
    ELSE
        RECEIVE
            DESTINATION =                       (at least one device other than the TD),
            SOURCE =                            IUT,
            ConfirmedEventNotification-Request,
            'Process Identifier' =              (the process identifier configured for this event),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the object detecting the alarm),
            'Time Stamp' =                      (the time or sequence number from the notification in step 2),
            'Notification Class' =              (the notification class configured for this event),
            'Priority' =                        (the priority configured for this event),
            'Event Type' =                      (the event type included in step 2),
            'Notify Type' =                     ACK_NOTIFICATION
9.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'111'
```

Notes to Tester: The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. 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.

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send 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 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.

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send 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 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.

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

```
1.  MAKE (a change that triggers the detection of an alarm event in the IUT)
2.  RECEIVE UnconfirmedEventNotification-Request,
        'Process Identifier' =                 (the process identifier configured for this event),
        'Initiating Device Identifier' =       IUT,
        'Event Object Identifier' =            (the object detecting the alarm),
        'Time Stamp' =                         (the current time or sequence number),
        'Notification Class' =                 (the notification class configured for this event),
        'Priority' =                           (the priority configured for this event type),
        'Event Type' =                         (any valid event type),
        'Notify Type' =                        ALARM,
        'AckRequired' =                        TRUE,
        'From State' =                         NORMAL,
        'To State' =                           (any appropriate non-normal event state),
        'Event Values' =                       (the values appropriate to the event type)
3.  IF (the notification in step 2 was not a broadcast) THEN
    RECEIVE
        DESTINATION =                          (at least one device other than the TD),
        SOURCE =                               IUT,
        UnconfirmedEventNotification-Request,
        'Process Identifier' =                 (the process identifier configured for this event),
        'Initiating Device Identifier' =       IUT,
        'Event Object Identifier' =            (the object detecting the alarm),
        'Time Stamp' =                         (the current time or sequence number),
        'Notification Class' =                 (the notification class configured for this event),
        'Priority' =                           (the priority configured for this event type),
        'Event Type' =                         (any valid event type),
        'Notify Type' =                        ALARM,
        'AckRequired' =                        TRUE,
        'From State' =                         NORMAL,
        'To State' =                           (any appropriate non-normal event state),
        'Event Values' =                       (the values appropriate to the event type)
4.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
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' =           (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)
6.  RECEIVE BACnet-Simple-ACK-PDU
7.  IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE
            DESTINATION =                      LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE =                           IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =              (the process identifier configured for this event),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the object detecting the alarm),
            'Time Stamp' =                      (the time or sequence number from the notification in step 2),
            'Notification Class' =              (the notification class configured for this event),
            'Priority' =                        (the priority configured for this event type),
            'Event Type' =                      (any valid event type),
            'Notify Type' =                     ACK_NOTIFICATION,
            'To State' =                        (the 'To State' used in step 2 or 3)
    ELSE
        RECEIVE
            DESTINATION =                       LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE =                            IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =              (the process identifier configured for this event),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the object detecting the alarm),
            'Time Stamp' =                      (the time or sequence number from the notification in step 2),
            'Notification Class' =              (the notification class configured for this event),
            'Priority' =                        (the priority configured for this event type),
            'Event Type' =                      (any valid event type),
            'Notify Type' =                     ACK_NOTIFICATION
8.  IF (the notification in step 7 was not broadcast) THEN
        IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
            RECEIVE
                DESTINATION =                   (at least one device other than the TD),
                SOURCE =                        IUT,
                UnconfirmedEventNotification-Request,
                'Process Identifier' =          (the process identifier configured for this event),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =     (the object detecting the alarm),
                'Time Stamp' =                  (the time or sequence number from the notification in step 2),
                'Notification Class' =          (the notification class configured for this event),
                'Priority' =                    (the priority configured for this event type),
                'Event Type' =                  (any valid event type),
                'Notify Type' =                 ACK_NOTIFICATION,
                'To State' =                    (the 'To State' used in step 2 or 3)
        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 time or sequence number from the notification in step 2),
                'Notification Class' =          (the notification class configured for this event),
                'Priority' =                    (the priority configured for this event type),
                'Event Type' =                  (any valid event type),
                'Notify Type' =                 ACK_NOTIFICATION,
                'To State' =                    (the 'To State' used in step 2 or 3)
9.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'111'
```

Notes to Tester: The destination address used for the acknowledgment notification in step 7 shall be the same address used in step 2. The destination address used for the acknowledgment notification in step 8 shall be the same address used in step 3. 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.

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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.

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 at least one other device. The TD acknowledges the alarm and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm has been acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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.

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. If the IUT cannot be configured to generate such a notification, then this test shall be skipped.

Test Steps:

```
1.  MAKE (a change that triggers the detection of an alarm event in the IUT)
2.  RECEIVE UnConfirmedEventNotification-Request,
        'Process Identifier' =            (the process identifier configured for this event),
        'Initiating Device Identifier' =  IUT,
        'Event Object Identifier' =       (the object detecting the alarm),
        'Time Stamp' =                    (the current time or sequence number),
          'Notification Class' =            (the notification class configured for this event),
          'Priority' =                      (the priority configured for this event),
          'Event Type' =                    (any valid event type),
          'Notify Type' =                   (any valid notify type),
          'AckRequired' =                   TRUE,
          'From State' =                    (any valid event-state),
          'To State' =                      (any "offnormal" event state other than offnormal itself),
          'Event Values' =                  (the values appropriate to the event type)
3.   VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = {0,?,?}
4.   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),
          'Time of Acknowledgment' =             (any valid timestamp)
5.   RECEIVE BACnet-Simple-ACK-PDU
6.   IF (Protocol_Revision is present and Protocol_Revision $\geq$ 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' =                    (the current time or sequence number),
               'Notification Class' =            (the notification class configured for this event),
               'Priority' =                      (the priority configured for this event),
               'Event Type' =                    (the event type included in step 2),
               'Notify Type' =                   ACK_NOTIFICATION,
               'To State' =                      (offnormal or the 'To State' from step 2)
     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' =                    (the current time or sequence number),
               'Notification Class' =            (the notification class configured for this event),
               'Priority' =                      (the priority configured for this event),
               'Event Type' =                    (the event type included in step 2),
               'Notify Type' =                   ACK_NOTIFICATION,
7.   VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = {1,?,?}
```

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 at least one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

```
1.  MAKE (a change that triggers the detection of an alarm event in the IUT)
2.  RECEIVE ConfirmedEventNotification-Request,
        'Process Identifier' =                 (the process identifier configured for this event),
        'Initiating Device Identifier' =       IUT,
        'Event Object Identifier' =            (the object detecting the alarm),
        'Time Stamp' =                         (the current time or sequence number),
        'Notification Class' =                 (the notification class configured for this event),
        'Priority' =                           (the priority configured for this event type),
        'Event Type' =                         (any valid event type),
        'Notify Type' =                        ALARM,
        'AckRequired' =                        TRUE,
        'From State' =                         NORMAL,
        'To State' =                           (any appropriate non-normal event state),
        'Event Values' =                       (the values appropriate to the event type)
3.  RECEIVE
        DESTINATION =                          (at least one device other than the TD),
        SOURCE =                               IUT,
        ConfirmedEventNotification-Request,
        'Process Identifier' =                 (the process identifier configured for this event),
        'Initiating Device Identifier' =       IUT,
        'Event Object Identifier' =            (the object detecting the alarm),
        'Time Stamp' =                         (the current time or sequence number),
        'Notification Class' =                 (the notification class configured for this event),
        'Priority' =                           (the priority configured for this event type),
        'Event Type' =                         (any valid event type),
        'Notify Type' =                        ALARM,
        'AckRequired' =                        TRUE,
        'From State' =                         NORMAL,
        'To State' =                           (any appropriate non-normal event state),
        'Event Values' =                       (the values appropriate to the event type)
4.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
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' =           (the state specified in the 'To State' parameter of the notification),
        'Time Stamp' =                         (a time stamp older than the one conveyed in the notification),
        'Time of Acknowledgment' =             (the current time using a Time format)
6.  RECEIVE BACnet-Error-PDU
        Error Class  =                         SERVICES,
        Error Code  =                          INVALID_TIME_STAMP
7.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
8.  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)
9.  RECEIVE BACnet-Simple-ACK-PDU
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE
            ConfirmedEventNotification-Request,
            'Process Identifier' =             (the process identifier configured for this event),
            'Initiating Device Identifier' =   IUT,
            'Event Object Identifier' =        (the object detecting the alarm),
            'Time Stamp' =                     (the time or sequence number from the notification in step 2),
            'Notification Class' =             (the notification class configured for this event),
            'Priority' =                       (the priority configured for this event type),
            'Event Type' =                     (any valid event type),
            'Notify Type' =                    ACK_NOTIFICATION,
            'To State' =                          (the 'To State' used in step 2 or 3)
    ELSE
        RECEIVE
            ConfirmedEventNotification-Request,
            'Process Identifier' =                (the process identifier configured for this event),
            'Initiating Device Identifier' =      IUT,
            'Event Object Identifier' =           (the object detecting the alarm),
            'Time Stamp' =                        (the time or sequence number from the notification in step 2),
            'Notification Class' =                (the notification class configured for this event),
            'Priority' =                          (the priority configured for this event type),
            'Event Type' =                        (any valid event type),
            'Notify Type' =                       ACK_NOTIFICATION
11. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE
            ConfirmedEventNotification-Request,
            'Process Identifier' =                (the process identifier configured for this event),
            'Initiating Device Identifier' =      IUT,
            'Event Object Identifier' =           (the object detecting the alarm),
            'Time Stamp' =                        (the time or sequence number from the notification in step 2),
            'Notification Class' =                (the notification class configured for this event),
            'Priority' =                          (the priority configured for this event type),
            'Event Type' =                        (any valid event type),
            'Notify Type' =                       ACK_NOTIFICATION,
            'To State' =                          (the 'To State' used in step 2 or 3)
    ELSE
        RECEIVE
            DESTINATION =                         (at least one device other than the TD),
            SOURCE =                              IUT,
            ConfirmedEventNotification-Request,
            'Process Identifier' =                (the process identifier configured for this event),
            'Initiating Device Identifier' =      IUT,
            'Event Object Identifier' =           (the object detecting the alarm),
            'Time Stamp' =                        (the time or sequence number from the notification in step 2),
            'Notification Class' =                (the notification class configured for this event),
            'Priority' =                          (the priority configured for this event type),
            'Event Type' =                        any valid event type),
            'Notify Type' =                       ACK_NOTIFICATION
12. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'111'
```

Notes to Tester: The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3.

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

Purpose: To verify that an alarm remains unacknowledged if the 'Acknowledging Process Identifier' is inconsistent with the other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm using an invalid process 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 process identifier and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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 'Acknowledging Process Identifier' shall have a value that is different from the 'Process Identifier' in the event notification.

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the Error Code in step 7 shall be INCONSISTENT_PARAMETERS.

9.1.2.3 Unsuccessful Alarm Acknowledgment of Confirmed Event Notifications Because the Referenced Object Does Not Exist

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm using an improper 'Event Object Identifier' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event Object Identifier' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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 have a value that is different from the 'Event Object Identifier' in the event notification and for which no object exists in the IUT.

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the Error Class in step 7 shall be OBJECT and the Error Code in step 7 shall be UNKNOWN_OBJECT. For devices that claim a Protocol_Revision of 5 or prior, an Error Class of SERVICES with an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.

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 other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm using an invalid event state and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event state and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send confirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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 State Acknowledged' shall have an off-normal value other than OFFNORMAL and other than the value of the 'To State' parameter in the event notification.

Notes to Tester: A passing result is the same message sequence described in 9.1.2.1 except that the Error Code in step 7 shall be INVALID_EVENT_STATE. For devices that claim a Protocol_Revision of 5 or prior, an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.

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 at least one other device. The TD acknowledges the alarm using an old time stamp and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper time stamp and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

```
1.  MAKE (a change that triggers the detection of an alarm event in the IUT)
2.  RECEIVE UnconfirmedEventNotification-Request,
        'Process Identifier' =                  (the process identifier configured for this event),
        'Initiating Device Identifier' =        IUT,
        'Event Object Identifier' =             (the object detecting the alarm),
        'Time Stamp' =                          (the current time or sequence number),
        'Notification Class' =                  (the notification class configured for this event),
        'Priority' =                            (the priority configured for this event type),
        'Event Type' =                          (any valid event type),
        'Notify Type' =                         ALARM,
        'AckRequired' =                         TRUE,
        'From State' =                          NORMAL,
        'To State' =                            (any appropriate non-normal event state),
        'Event Values' =                        (the values appropriate to the event type)
3.  IF (the notification in step 2 was not a broadcast) THEN
    RECEIVE
        DESTINATION =                           (at least one device other than the TD),
        SOURCE =                                IUT,
        UnconfirmedEventNotification-Request,
        'Process Identifier' =                  (the process identifier configured for this event),
        'Initiating Device Identifier' =        IUT,
        'Event Object Identifier' =             (the object detecting the alarm),
        'Time Stamp' =                          (the current time or sequence number),
        'Notification Class' =                  (the notification class configured for this event),
        'Priority' =                            (the priority configured for this event type),
        'Event Type' =                          (any valid event type),
        'Notify Type' =                         ALARM,
        'AckRequired' =                         TRUE,
        'From State' =                          NORMAL,
        'To State' =                            (any appropriate non-normal event state),
        'Event Values' =                        (the values appropriate to the event type)
4.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
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' =            (the state specified in the 'To State' parameter of the notification),
        'Time Stamp' =                          (a time stamp older than the one conveyed in the notification),
        'Time of Acknowledgment' =              (the current time using a Time format)
6.  RECEIVE BACnet-Error-PDU
        Error Class  =                          SERVICES,
        Error Code  =                           INVALID_TIME_STAMP
7.  VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'011'
8.  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)
9.  RECEIVE BACnet-Simple-ACK-PDU
```

```
10. IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
        RECEIVE
            DESTINATION =                       LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE =                            IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =              (the process identifier configured for this event),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the object detecting the alarm),
            'Time Stamp' =                      (the time or sequence number from the notification in step 2),
            'Notification Class' =              (the notification class configured for this event),
            'Priority' =                        (the priority configured for this event type),
            'Event Type' =                      (any valid event type),
            'Notify Type' =                     ACK_NOTIFICATION,
            'To State' =                        (the 'To State' used in step 2 or 3)
    ELSE
        RECEIVE
            DESTINATION =                       LOCAL BROADCAST | GLOBAL BROADCAST | TD,
            SOURCE =                            IUT,
            UnconfirmedEventNotification-Request,
            'Process Identifier' =              (the process identifier configured for this event),
            'Initiating Device Identifier' =    IUT,
            'Event Object Identifier' =         (the object detecting the alarm),
            'Time Stamp' =                      (the time or sequence number from the notification in step 2),
            'Notification Class' =              (the notification class configured for this event),
            'Priority' =                        (the priority configured for this event type),
            'Event Type' =                      (any valid event type),
            'Notify Type' =                     ACK_NOTIFICATION
11. IF (the notification in step 10 was not broadcast) THEN
        IF (Protocol_Revision is present and Protocol_Revision ≥ 1) THEN
            RECEIVE
                DESTINATION =                   (at least one device other than the TD),
                SOURCE =                        IUT,
                UnconfirmedEventNotification-Request,
                'Process Identifier' =          (the process identifier configured for this event),
                'Initiating Device Identifier' = IUT,
                'Event Object Identifier' =     (the object detecting the alarm),
                'Time Stamp' =                  (the time or sequence number from the notification in step 2),
                'Notification Class' =          (the notification class configured for this event),
                'Priority' =                    (the priority configured for this event type),
                'Event Type' =                  (any valid event type),
                'Notify Type' =                 ACK_NOTIFICATION,
                'To State' =                    (the 'To State' used in step 2 or 3)
        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 time or sequence number from the notification in step 2),
                'Notification Class' =          (the notification class configured for this event),
                'Priority' =                    (the priority configured for this event type),
                'Event Type' =                  (any valid event type),
                'Notify Type' =                 ACK_NOTIFICATION
12. VERIFY (the 'Event Object Identifier' from the event notification), Acked_Transitions = B'111'
```

Notes to Tester: The destination address used for the acknowledgment notification in step 10 shall be the same address used in step 2. The destination address used for the acknowledgment notification in step 11 shall be the same address used in step 3.

9.1.2.6 Unsuccessful Alarm Acknowledgment of Unconfirmed Event Notifications Because the Referenced Object Does Not Exist

Purpose: To verify that an alarm remains unacknowledged if the 'Event Object Identifier' represents an object that does not exist.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm using an invalid event object identifier and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper event object identifier and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device shall be recipients of the alarm notification.

Test Steps: The test steps defined in 9.1.2.5 shall be followed except that in the first AcknowledgeAlarm request the 'Time Stamp' shall have the same value as the 'Time Stamp' from the event notification and the 'Event Object Identifier' shall have a value that is different from the 'Event Object Identifier' in the event notification and for which no object exists in the IUT.

Notes to Tester: A passing result is the same message sequence described as the passing result in 9.1.2.5 except that the Error Class in step 7 shall be OBJECT and the Error Code in step 7 shall be UNKNOWN_OBJECT. For devices that claim a Protocol_Revision of 5 or prior, an Error Class of SERVICES with an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.

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 other parameters that define the alarm being acknowledged.

Test Concept: An alarm is triggered that causes the IUT to notify the TD and at least one other device. The TD acknowledges the alarm using an invalid 'Event State Acknowledged' and verifies that the acknowledgment is not accepted by the IUT and that the IUT does not notify other devices that the alarm was acknowledged. The TD then acknowledges the alarm using the proper 'Event State Acknowledged' and verifies that the acknowledgment is properly noted by the IUT. The IUT notifies all other recipients that the alarm was acknowledged.

Configuration Requirements: The IUT shall be configured with at least one object that can detect alarm conditions and send unconfirmed notifications. The Acked_Transitions property shall have the value B'111' indicating that all transitions have been acknowledged. The TD and at least one other BACnet device 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 State Acknowledged' shall have an off-normal value other than OFFNORMAL and other than the value of the 'To State' parameter in the event notification.

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 that claim a Protocol_Revision of 5 or prior, an Error Code of INCONSISTENT_PARAMETERS shall also be accepted.

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 Notification from Analog, Binary, Multi-state, and Life Safety Objects

COV notifications convey the value of the Present_Value and Status_Flags properties when initiated by Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, and Life Safety Zone objects. Since the ability to subscribe to COV notifications is general and can be applied to any of these object types, the IUT shall demonstrate that it correctly responds to COV notifications from objects representing each of these object types. The test procedure defined in this clause shall be applied once for each object type.

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from analog, binary, and multi-state objects.

Test Steps:

```
    REPEAT X = (one object of each type in the set {Analog Input, Analog Output, Analog Value, Binary Input, Binary
    Output, Binary Value, Multi-state Input, Multi-state Output, Multi-state Value, Life Safety Point, Life Safety Zone})
    DO {
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 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)
```

9.2.2 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.  RECEIVE BACnet-Error-PDU,
         Error Class  =               SERVICES,
         Error Code  =                (any valid error code for class SERVICES)
```

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.  RECEIVE BACnet-Error-PDU,
         Error Class  =               SERVICES,
         Error Code  =                (any valid error code for class SERVICES)
```

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. RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = (any valid error code for class SERVICES)

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. RECEIVE BACnet-Error-PDU,
   Error Class = SERVICES,
   Error Code = (any valid error code for class SERVICES)

9.2.2.5 Change of Value Notifications with an Invalid List of Values

Purpose: To verify that an appropriate reject is returned if a COV notification arrives that contains a list of values that is not appropriate for the object type of the monitored object.

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' = X,
   'Time Remaining' = (any amount of time greater than 0),
   'List of Values' = (a list of values that is not appropriate to object X)
4. RECEIVE BACnet-Reject-PDU,
   'Reject Reason' = INCONSISTENT_PARAMETERS | INVALID_PARAMETER_DATATYPE | INVALID_TAG

9.3 UnconfirmedCOVNotification Service Execution Tests

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

9.4 ConfirmedEventNotification Service Execution Tests

Dependencies: None.

BACnet Reference Clause: 13.8.

Purpose: To verify that the IUT can execute the ConfirmedEventNotification service request.

Test Concept: BACnet does not define any action to be taken upon receipt of a ConfirmedEventNotification except to return an acknowledgement. Although returning an acknowledgment is sufficient to conform to the standard, a vendor may specify additional actions that can be observed. Any vendor-defined actions that do not occur during the tests shall be noted in the test report. No negative tests are included.

9.4.1 ConfirmedEventNotification Using the Time Form of the 'Timestamp' Parameter and Conveying a Text Message

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 Text Message

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 Text Message

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

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),
        '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)
```

9.5 UnconfirmedEventNotification Service Execution Tests

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

9.6 GetAlarmSummary Service Execution Tests

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

BACnet Reference Clause: 13.10.

Dependencies: None.

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' = ALL
2. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (an empty list)

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 Event_State = 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 Event_State = 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 Event_State = 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 Event_State = 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 the event types CHANGE_OF_BITSTRING, CHANGE_OF_STATE, CHANGE_OF_VALUE, COMMAND_FAILURE, FLOATING_LIMIT, and OUT_OF_RANGE. If only a subset of these event types are supported as many of them as possible shall be configured.

Test Steps:

1. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,
   'Event Type Filter' = CHANGE_OF_BITSTRING
2. RECEIVE GetEnrollmentSummary-ACK,
   'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_BITSTRING)
3. TRANSMIT GetEnrollmentSummary-Request,
   'Acknowledgment Filter' = ALL,

'Event Type Filter' = CHANGE_OF_STATE
4. RECEIVE GetEnrollmentSummary-ACK,
'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_STATE)
5. TRANSMIT GetEnrollmentSummary-Request,
'Acknowledgment Filter' = ALL,
'Event Type Filter' = CHANGE_OF_VALUE
6. RECEIVE GetEnrollmentSummary-ACK,
'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = CHANGE_OF_VALUE)
7. TRANSMIT GetEnrollmentSummary-Request,
'Acknowledgment Filter' = ALL,
'Event Type Filter' = FLOATING_LIMIT
8. RECEIVE GetEnrollmentSummary-ACK,
'List of Enrollment Summaries' = (all configured event-generating objects with Event_Type = FLOATING_LIMIT)

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. 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.

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 9.6.4 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 50 bytes.

Test Concept: In steps 1-4, the test first tests proper chaining by requesting two lists from the IUT and verifying that the second list is properly distinct from the first. In steps 5-9, to test the “fixed object processing order” as defined in BACnet 13.12.1.1.1, it requests the first list again, and then, before requesting the second list, the tester makes the last object in the first list no longer have any active event states. When the TD requests the second list using the object identifier of the now-normal device, the IUT should respond with the same second list as it did before.

Test Steps:

1. TRANSMIT GetEventInformation-Request,
   'max-APDU-length-accepted' = B'0000',
   'segmented-response-accepted' = FALSE
2. RECEIVE GetEventInformation-ACK,
   'List of Event Summaries' = (an arbitrary list),
   'More Events' = TRUE
3. TRANSMIT GetEventInformation-Request,
   'Last Received Object Identifier' = the last object identifier of the list received in step 2)

```
4.  RECEIVE GetEventInformation-ACK,
        'List of Event Summaries' =          (a list of object identifiers not including any received in step 2)
5.  TRANSMIT GetEventInformation-Request,
        'max-APDU-length-accepted'  =      B'0000',
        'segmented-response-accepted'  =   FALSE
6.  RECEIVE GetEventInformation-ACK,
        'List of Event Summaries' =          (an arbitrary list),
        'More Events' =                      TRUE
7.  MAKE (the object identified by the last object identifier in the list received in step 6 have no active event states)
8.  TRANSMIT GetEventInformation-Request,
        'Last Received Object Identifier' =  (the last object identifier of the list received in step 6)
9.  RECEIVE GetEventInformation-ACK,
        'List of Event Summaries' =          (the same list received in step 4)
```

9.9 LifeSafetyOperation Service Execution Test

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

9.9.1 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.  WAIT Notification Fail Time
4.  IF (the IUT supports 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' =       (any value > 0 if automatic cancellation is supported, otherwise 0),
          'List of Values' =            (values appropriate to the object type of the monitored object)
    ELSE
          RECEIVE BACnet-Error PDU,
                 Error Class  =           SERVICES,
                 Error Code  =            SERVICE_REQUEST_DENIED | OTHER
```

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.  WAIT Notification Fail Time
4.  IF (the IUT supports confirmed notifications) THEN
     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)
    ELSE
          RECEIVE BACnet-Error PDU,
                 Error Class  =           SERVICES,
                 Error Code  =            SERVICE_REQUEST_DENIED | OTHER
```

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.  WAIT Notification Fail Time
4.  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' =       0,
          'List of Values' =             (values appropriate to the object type of the monitored object)
    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' =       0,
          'List of Values' =             (values appropriate to the object type of the monitored object)
5.  MAKE (a change to the monitored object that should cause a COV notification)
6.  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' =       0,
          'List of Values' =             (values appropriate to the object type of the monitored object
                                including the changed value that triggered the notification)
    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' =       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.

```
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.  WAIT Notification Fail Time
4.  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)
    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)
5.  MAKE (a change to the monitored object that should cause a COV notification)
6.  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)
     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
                                     including the changed value of that triggered the notification)
7.   WAIT (the lifetime of the subscription)
8.   MAKE (a change to the monitored object that would cause a COV notification if there were an active subscription)
9.   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.

```
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.   WAIT Notification Fail Time
4.   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' =        60,
          'List of Values' =         (values appropriate to the object type of the monitored object)
     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' =        60,
          'List of Values' =         (values appropriate to the object type of the monitored object)
5.   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)
6.   RECEIVE BACnet-SimpleACK-PDU
7.   WAIT Notification Fail Time
```

```
8.  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)
    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)
```

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.  RECEIVE BACnet-Error PDU,
          Error Class  =                  SERVICES,
          Error Code  =                   SERVICE_REQUEST_DENIED | OTHER
```

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.

```
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)
5.  MAKE (a change to the monitored object that should cause a COV notification)
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' =                 (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)
7.  WAIT (the lifetime of the subscription)
8.  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.

```
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.  RECEIVE BACnet-Error-PDU,
        Error Class  =                    SERVICES,
        Error Code  =                     SERVICE_REQUEST_DENIED | OTHER
```

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.   RECEIVE BACnet-Error PDU,
          Error Class  =                     SERVICES,
          Error Code  =                      SERVICE_REQUEST_DENIED | OTHER
```

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 < 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 < 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 < (the number of records in the test file – 2)),
   'Requested Record Count' = (any number m: 0 < m < 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 ≤ n < 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: $0 \leq n <$ the number of records in the test file),
'Requested Record Count' = (any number m: $m >$ 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: $0 < n <$ 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: $0 < n <$ 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: $0 < n <$ (the number of octets in the test file – 2)),
   'Requested Octet Count' = (any number m: $0 < m <$ 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: $0 \le n <$ 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: $0 \le n <$ the number of octets in the test file),
   'Requested Octet Count' = (any number m: $m >$ 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 ≥ the number of records in the test file),
     'Requested Record Count' =   (any number > 0)
2.  RECEIVE BACnet-Error PDU,
        Error Class  =              SERVICES,
        Error Code  =               INVALID_FILE_START_POSITION
```

9.12.2.1.2 Attempting to Read Data from a Nonexistent File

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified file does not exist.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =          (any non existent file),
     'File Start Record' =        (any number ≥ 0),
     'Requested Record Count' =   (any number > 0)
2.  RECEIVE BACnet-Error PDU,
        Error Class  =              OBJECT,
        Error Code  =               UNKNOWN_OBJECT
```

9.12.2.1.3 Attempting to Read Data Using the Wrong File Access Type

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the file access type is inappropriate for the specified file.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =          R,
     'File Start Position' =      0,
     'Requested Octet Count' =        1
2.  RECEIVE BACnet-Error PDU,
        Error Class  =              SERVICES,
        Error Code  =               INVALID_FILE_ACCESS_METHOD
```

9.12.2.1.4 Attempting to Read Data Beginning with a Record Number Less Than Zero

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified record range is invalid.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =          R,
     'File Start Record' =        (any number n: n <0),
     'Requested Record Count' =   1
```

```
2.  RECEIVE
        (BACnet-Error PDU,
                Error Class  =          SERVICES,
                Error Code  =           INVALID_START_POSITION) |
       (BACnet-Reject-PDU,
                Reject Reason =         PARAMETER_OUT_OF_RANGE)
```

9.12.2.2 Reading Stream Based Files

The purpose of this test group is to verify the correct execution of the AtomicReadFile service requests from stream-based files under circumstances where the service is expected to fail.

9.12.2.2.1 Attempting to Read Data from a Range of Records Outside the File Boundaries

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified octets are outside of the boundaries of the file.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =            S,
     'File Start Position' =        (any number n: n ≥ the number of octets in the test file),
     'Requested Octet Count' =          (any number > 0)
2.  RECEIVE BACnet-Error PDU,
        Error Class  =                  SERVICES,
        Error Code  =                   INVALID_FILE_START_POSITION
```

9.12.2.2.2 Attempting to Read Data from a Nonexistent File

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified file does not exist.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =            (any non existent file),
     'File Start Position' =        (any number ≥ 0),
     'Requested Octet Count' =          (any number > 0)
2.  RECEIVE BACnet-Error PDU,
        Error Class  =                  OBJECT,
        Error Code  =                   UNKNOWN_OBJECT
```

9.12.2.2.3 Attempting to Read Data Using the Wrong File Access Type

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the file access type is inappropriate for the specified file.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =            S,
     'File Start Record' =          0,
     'Requested Record Count' =     1
2.  RECEIVE BACnet-Error PDU,
        Error Class  =                  SERVICES,
        Error Code  =                   INVALID_FILE_ACCESS_METHOD
```

9.12.2.2.4 Attempting to Read Data Beginning with a Start Position Less Than Zero

Purpose: To verify the correct execution of the AtomicReadFile service request under circumstances where the specified record range is invalid.

Test Steps:

```
1.  TRANSMIT AtomicReadFile-Request,
     'File Identifier' =           S,
     'File Start Position' =       (any number n: n <0),
     'Requested Octet Count' =         1
2.  RECEIVE
    (BACnet-Error PDU,
         Error Class  =              SERVICES,
         Error Code  =               INVALID_START_POSITION) |
    (BACnet-Reject-PDU,
        Reject Reason =              PARAMETER_OUT_OF_RANGE)
```

9.13 AtomicWriteFile Service Execution Tests

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

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 ≥ 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 ≤ 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 < 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 ≥ 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 < 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 < 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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-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' = 0

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. 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 or more elements to be added to the list with the second element having the wrong datatype)
4. RECEIVE AddListElement-Error,
   Error Class = SERVICES,
   Error Code = INVALID_PARAMETER_DATATYPE
   'First Failed Element' = 2
5. 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-Simple-ACK-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-Simple-ACK-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. 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' = (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)
4. RECEIVE RemoveListElement-Error,
   Error Class = SERVICES | PROPERTY,
   Error Code = OTHER
   'First Failed Element' = 2
5. VERIFY (L), ListProp = InitialList

9.16CreateObject 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. TRANSMIT CreateObject-Request,
   'Object Type' = (any creatable object type)
2. RECEIVE CreateObject-ACK,
   'Object Identifier' = (any unique object identifier of the specified type)
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)

9.16.1.2 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 Identifier' =        (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.  TRANSMIT CreateObject-Request,
        'Object Type' =              (any creatable object type),
        'List Of Initial Values' =   (a list of one or more properties and their initial values)
2.  RECEIVE CreateObject-ACK,
        'Object Identifier' =        (any unique object identifier of the specified type)
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)
```

9.16.1.4 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 Identifier' =            (any unique object identifier of a type that is creatable )
        'List Of Initial Values' =       (a list of two or more properties and their initial values)
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 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
4.  VERIFY (the IUT's Device object),
        Object_List =                 (any object list that does not contain the object specified 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.  TRANSMIT CreateObject-Request,
        'Object Type' =               (any creatable object type),
        'List Of Initial Values' =    (a list of two or more properties and their initial values with one of the values
                                       being out of range)
2.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE CreateObject-Error PDU,
            Error Class  =                  PROPERTY,
            Error Code  =                   VALUE_OUT_OF_RANGE
            'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
    ELSE
        RECEIVE CreateObject-Error,
            Error Class =                   PROPERTY,
            Error Code =                    VALUE_OUT_OF_RANGE |
                                            OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED | OTHER
            'First Failed Element Number' = (the position in the 'List Of Initial Values' with the offending value)
3.  CHECK(Verify that the new object was not created)
4.  TRANSMIT CreateObject-Request,
        'Object Type' =               (any creatable object type),
        'List Of Initial Values' =    (a list of two or more properties and their initial values with one of the values
                                        being an inappropriate datatype)
```

```
5.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE CreateObject-Error PDU,
            Error Class  =                    PROPERTY,
            Error Code  =                     INVALID_DATATYPE
            'First Failed Element Number' =   (the position in the 'List Of Initial Values' with the offending value)
    ELSE
        RECEIVE CreateObject-Error,
            Error Class =                     PROPERTY,
            Error Code =                      VALUE_OUT_OF_RANGE | INVALID_DATATYPE |
                                              OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED | OTHER
            'First Failed Element Number' =   (the position in the 'List Of Initial Values' with the offending value)
6.  CHECK(Verify that the new object was not created)
```

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

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

Test Steps:

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

```
6.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE BACnet-Error PDU,
            Error Class =     OBJECT,
            Error Code =      UNKNOWN_OBJECT
    ELSE
            RECEIVE BACnet-Error PDU
            Error Class =     OBJECT,
            Error Code =      UNKNOWN_OBJECT | NO_OBJECTS_OF_SPECIFIED_TYPE | OTHER
```

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 = (the Object_Name specified in the EPICS)
2.  TRANSMIT DeleteObject-Request,
     'Object Identifier' = X
3.  RECEIVE BACnet-Simple-ACK-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
```

9.18ReadProperty 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,
        '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.

```
1.  TRANSMIT ReadProperty-Request,
         'Object Identifier' =    (Device, X),
         'Property Identifier' =  Object_List,
         '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 that is not defined for the specified object.

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.

9.20ReadPropertyMultiple 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 object, Object2, not supported in 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' =        Object1,
          REPEAT P = (each property supported by Object1) DO {
               'Property Identifier' =      P,
               'Property Value' =   (the value of P specified in the EPICS)
          }
     }
```

Notes to Tester: Any proprietary properties that are supported for the object-type shall also be returned (see BACnet 15.7.3.1.2).

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' = Object1,
          'Property Identifier' =      OPTIONAL
     RECEIVE ReadPropertyMultiple-ACK,
          'Object Identifier' =        Object1,
          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.

Test Steps:

```
1.  REPEAT ObjectX = (one instance of each supported object type) DO {
     TRANSMIT ReadPropertyMultiple-Request,
          'Object Identifier' =        Object1,
          'Property Identifier' =           REQUIRED
     RECEIVE ReadPropertyMultiple-ACK,
          'Object Identifier' =             Object1,
          REPEAT P = (each required property defined for Object1) DO {
               'Property Identifier' =      P,
               'Property Value' =    (the value of P specified in the EPICS)
          }
     }
```

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 =
(MaxAPDU - NonSegRqstHdrSize - ObjIdSize - TagsSize) / PropIdSize =
(MaxAPDU - 11) / 2

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

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

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

NumPropertiesToUse = min(MaxPropsPerRqst, MaxPropsPerResp)

Test Steps:

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

9.20.2 Negative ReadPropertyMultiple Service Execution Tests

The purpose of this test group is to verify correct execution of ReadPropertyMultiple service requests under circumstances where the service is expected to fail.

9.20.2.1 Reading a Single, Unsupported Property from a Single Object

Purpose: To verify the ability to correctly execute a ReadPropertyMultiple service request for which the 'List of Read Access Specifications' contains specifications for a single unsupported property.

Test Steps:

```
1.   TRANSMIT ReadPropertyMultiple-Request,
      'Object Identifier' = Object1 | Object2,
      'Property Identifier' =     (any property, P1, that is not supported in the selected object)
2.   RECEIVE BACnet-Error-PDU,
      'Error Class' =       PROPERTY,
      'Error Code' =        UNKNOWN_PROPERTY
```

9.20.2.2 Reading Multiple Properties with Access Errors for Every Property

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

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

Test Steps:

```
1.   TRANSMIT ReadPropertyMultiple-Request,
      'Object Identifier' = Object1,
      'Property Identifier' =     P1,
      'Property Identifier' =     P2,
      'Property Identifier' =     P3,
      'Object Identifier' =       Object2,
      'Property Identifier' =     P4,
      'Property Identifier' =     P5,
      'Property Identifier' =     P6
2.   RECEIVE
          (BACnet-Error-PDU,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class) ) |
          (ReadPropertyMultiple-ACK,
               'Object Identifier' =   Object1,
               'Property Identifier' =P1,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class),
               'Property Identifier' =P2,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class),
               'Property Identifier' =P3,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class),
               'Object Identifier' =   Object2,
               'Property Identifier' =P4,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class),
               'Property Identifier' =P5,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class),
               'Property Identifier' =P6,
               'Error Class' =   OBJECT | PROPERTY,
               'Error Code' =    (any valid error code for the returned error class) )
```

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

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

Test Steps:

```
1.   TRANSMIT ReadPropertyMultiple-Request,
          'Object Identifier' =          (Device, X),
          'Property Identifier' =        Vendor_Name,
          'Array Index' =                1
2.   RECEIVE
          (BACnet-Error-PDU,
               'Error Class'  =   PROPERTY,
               'Error Code'   =   PROPERTY_IS_NOT_AN_ARRAY) |
          (ReadPropertyMultiple-ACK,
               'Object Identifier' =        (Device, X),
               'Property Identifier' =      Vendor_Name,
               'Array Index' =              1,
               'Error Class' =              PROPERTY,
               'Error Code' =               PROPERTY_IS_NOT_AN_ARRAY)
```

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 Log_Enable property shall be FALSE so that the Log_Buffer does not change during the tests.

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 Trend Log object configured for this test),
          'Property Identifier' =  Log_Buffer
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 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 Trend Log object configured for this test),
   'Property Identifier' = Log_Buffer,
   'Reference Index' = (any value x: $1 < x <$ (the number of trend records in the buffer - the 'Count' used below)),
   'Count' = (any value x: $0 < x \leq$ the number of trend records remaining beyond "Reference Index')
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 same value used in the 'Count' parameter in step 1),
   'Item Data' = (all of the specified trend records)

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 Trend Log object configured for this test),
   'Property Identifier' = Log_Buffer,
   'Reference Index' = (any value x: $2 < x <$ the number of trend records in the buffer),
   'Count' = (any value x: $x < 0$ AND $|x| <$ the 'Reference Index')
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 same value used in the 'Count' parameter in step 1),
   'Item Data' = (all of the specified trend records)

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 Trend Log object configured for this test),
   'Property Identifier' = Log_Buffer,
   'Reference Time' = (any value older than (earlier time) the last time in the buffer),
   'Count' = (any value > 0)
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 same value used in the 'Count' parameter in step 1),
   'Item Data' = (all of the specified trend records)

Notes to Tester: The first item returned shall be the entry in the Log_Buffer with a timestamp newer (later time) than the time specified by the 'Reference Time' parameter.

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)
```

9.22 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 ≤ N ≤ 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 ≥ 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 that the supported size if 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 an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

```
1.  VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =    Object1,
        'Property Identifier' =  P1,
        'Property Value' =       (any value with an invalid datatype)
3.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE BACnet-Error PDU,
            Error Class =        PROPERTY,
            Error Code =         INVALID_DATATYPE
    ELSE
        RECEIVE (BACnet-Error PDU,
            Error Class  =       PROPERTY,
            Error Code  =        INVALID_DATATYPE) |
        (BACnet-Reject-PDU
            Reject Reason =      INVALID_PARAMETER_DATATYPE) |
        (BACnet-Reject-PDU
            Reject Reason =      INVALID_TAG)
4.  VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
```

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.

Test Steps:

```
1.  VERIFY (Object1), P1 = (the value defined for this property in the EPICS),
2.  TRANSMIT WriteProperty-Request,
        'Object Identifier' =    (Object1, any object with writable properties),
        'Property Identifier' =  (P1, any property with a restricted range of values),
        'Property Value' =       (any value, of the correct datatype, that is outside the supported range)
3.  IF (Protocol_Revision is present and Protocol_Revision ≥ 4) THEN
        RECEIVE (BACnet-Error PDU,
            Error Class =        PROPERTY,
            Error Code =         VALUE_OUT_OF_RANGE
    ELSE
```

```
        RECEIVE (BACnet-Error-PDU,
            Error Class =       PROPERTY,
            Error Code =        VALUE_OUT_OF_RANGE) |
        (BACnet-Reject-PDU,
            Reject Reason =  PARAMETER_OUT_OF_RANGE)
4.  VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
```

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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
6. VERIFY (Object2), P2 = (the value specified for P2 in step 3)

9.23.1.4 Writing Multiple Properties to Multiple Objects

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)

4. RECEIVE BACnet-BACnet-SimpleACK-PDU
5. VERIFY (Object1), P1 = (the value specified for P1 in step 5)
6. VERIFY (Object1), P2 = (the value specified for P2 in step 5)
7. VERIFY (Object2), P3 = (the value specified for P3 in step 5)
8. 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 =
(MaxAPDU – NonSegRqstHdrSize – ObjIdSize – TagsSize) / (PropIdSize + TagsSize + ValueSize) =
(MaxAPDU – 11) / (PropIdSize + 2 + ValueSize)

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

Test Steps:

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

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)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = WRITE_ACCESS_DENIED,
   objectIdentifier = Object1,
   propertyIdentifier = P2
4. VERIFY (Object1), P1 = (the value specified for P1 in step 3)
2. 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 that the supported size of the array)

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

9.23.2.6 Writing with a Property Value Having the Wrong Datatype

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype. This test shall only be performed if Protocol_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)
2. TRANSMIT WritePropertyMultiple-Request,
   'Object Identifier' = Object1,
   'Property Identifier' = P1,
   'Property Value' = (any value with an invalid datatype)
3. RECEIVE WritePropertyMultiple-Error,
   Error Class = PROPERTY,
   Error Code = INVALID_DATATYPE,
   objectIdentifier = Object1,
   propertyIdentifier = P1
4. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)

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-Simple-ACK-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)
6.  CHECK (Verify that the IUT has not transmitted any messages 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-Simple-ACK-PDU
9.  VERIFY (Device, X),
        (any required non-array property) = (the value for this property specified in the EPICS)
```

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-Simple-ACK-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)
6. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2)
7. TRANSMIT ReinitializeDevice-Request,
   Reinitialized State of Device' = WARMSTART,
   'Password' = (any appropriate password as described in the Test Concept)
8. RECEIVE BACnet-Simple-ACK-PDU
9. CHECK (Did the IUT perform a COLDSTART reboot?)
10. VERIFY (Device, X),
    (any required non-array property) = (the value for this property specified in the EPICS)

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-Simple-ACK-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 (T)
6. CHECK (Verify that the IUT did not transmit any messages between the acknowledgment in step 2 and expiration of timer T)
7. VERIFY (Device, X),
   (any required non-array property) = (the value for this property specified in the EPICS)

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 messages 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 messages 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-Simple-ACK-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-Simple-ACK-PDU
3. WAIT **Internal Processing Fail Time**
4. MAKE (do something that would normally cause the IUT to initiate a message)
5. WAIT (an arbitrary time > **Internal Processing Fail Time** selected by the tester)
6. CHECK (Verify that the IUT has not transmitted any messages since the acknowledgment in step 2.)
7. TRANSMIT ReadProperty-Request,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (any required non-array property of the Device object)

8. RECEIVE BACnet-ComplexACK-PDU,
   'Object Identifier' = (Device, X),
   'Property Identifier' = (the 'Property Identifier' specified in step 7),
   'Property Value' = (any valid value for the property)
9. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = ENABLE,
   'Password' = (any appropriate password as described in the Test Concept)
10. RECEIVE BACnet-Simple-ACK-PDU
11. MAKE (do something to cause the IUT to initiate a message)
12. WAIT **Internal Processing Fail Time**
13. CHECK (Verify that the IUT initiated a message)

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

Dependencies: ReinitializeDevice Service Execution Tests, 9.27.

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

Test Steps:

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

9.24.1.8 Finite Time Duration, Disable Initiation

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

Test Steps:

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

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

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),
   (any required non-array property) = (the value for this property specified in the EPICS)

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 this test case shall be omitted.

Test Steps:

1. TRANSMIT DeviceCommunicationControl-Request,
   'Enable/Disable' = DISABLE,
2. (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),
   (any required non-array property) = (the value for this property specified in the EPICS)

9.25ConfirmedPrivateTransfer 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 provided.

Test Steps:

```
1.  TRANSMIT ReinitializeDevice-Request,
     'Reinitialized State of Device' =     COLDSTART
2.  (RECEIVE BACnet-Simple-ACK-PDU
     CHECK (Did the IUT perform a COLDSTART reboot?) ) |
    (RECEIVE BACnet-Error-PDU,
     Error Class =   SECURITY,
     Error Code =   PASSWORD_FAILURE)      |
    (RECEIVE BACnet-Error-PDU,
     Error Class =   SERIVCES,
     Error Code =   MISSING_REQUIRED_PARAMETER)
```

Notes to Tester: Two cases are possible. If the IUT requires the use of a password one of the specified errors shall be returned. If the IUT does not require the use of a password a simple acknowledgment shall be returned and the IUT shall reinitialize in the manner prescribed by the manufacturer. 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-Simple-ACK-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 provided.

Test Steps:

```
1.  TRANSMIT ReinitializeDevice-Request,
     'Reinitialized State of Device' =    WARMSTART
2.  (RECEIVE BACnet-Simple-ACK-PDU
     CHECK (Did the IUT perform a WARMSTART reboot?) |
    (RECEIVE BACnet-Error-PDU,
     Error Class =  SECURITY,
     Error Code =  PASSWORD_FAILURE)     |
    (RECEIVE BACnet-Error-PDU,
     Error Class =  SERIVCES,
     Error Code =  MISSING_REQUIRED_PARAMETER)
```

Notes to Tester: Two cases are possible. If the IUT requires the use of a password one of the specified errors shall be returned. If the IUT does not require the use of a password a simple acknowledgment shall be returned and the IUT shall reinitialize in the manner prescribed by the manufacturer. 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-Simple-ACK-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.

9.28 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 Initial_UTC_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: $0 \le L <$ 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.

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: $0 \le L <$ 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.

1. TRANSMIT
   DA = LOCAL BROADCAST,
   SA = TD,
   Who-Has-Request,
   'Device Instance Low Limit' = (any value > 0 such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
   'Device Instance High Limit' = (any value > Device Instance Low Limit such that the Device object instance number does not fall in the range between Device Instance Low Limit and Device Instance High Limit),
   'Object Name = (any object name specified in the EPICS)
2. WAIT **Internal Processing Fail Time**
3. CHECK (verify that the IUT does not respond)

9.32.1.7 Object ID Version with IUT Device Instance Equal to the High Limit of the Device Range

Purpose: To verify that the IUT correctly recognizes the high limit of the specified device range for Who-Has service requests that utilize the object identifier form.

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)

9.32.2 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 = TD,
   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. WAIT **Internal Processing Fail Time**
3. 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)

9.33.1.2 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. WAIT **Internal Processing Fail Time**
3. 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)

9.33.1.3 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. WAIT **Internal Processing Fail Time**
3. 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)

9.33.1.5 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. WAIT **Internal Processing Fail Time**
3. 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)

9.33.1.6 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: $0 \leq L <$ 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. WAIT **Internal Processing Fail Time**
3. 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)

9.33.2 Execution of Who-Is Service Requests Originating from a Remote Network

The purpose of this test group is to verify the correct execution of the Who-Is request service procedure for messages originating from a remote network. A comprehensive set of variations in Who-Is request parameters is not included in this test group because they are tested in 9.33.1. The tests in this group only represent variations in network layer addressing information.

9.33.2.1 General Inquiry, Global Broadcast from a Remote Network

Purpose: To verify the ability of the IUT to recognize the origin of a globally broadcast Who-Is service request and respond such that the device originating the request receives the response.

Test Steps:

1. TRANSMIT
   DESTINATION = GLOBAL BROADCAST,
   SNET = (any remote network number),
   SADR = (any MAC address valid for the specified network),
   Who-Is-Request
2. WAIT **Internal Processing Fail Time**
3. RECEIVE
   DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified by SNET in step 1),
   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. WAIT **Internal Processing Fail Time**

3. RECEIVE
   DESTINATION = GLOBAL BROADCAST | REMOTE BROADCAST (to the network specified by SNET in step 1),
   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 a broadcast I-Am service request even if the Who-Is service request was not transmitted with a broadcast address.

Test Steps:

1. TRANSMIT
   DESTINATION = IUT,
   SNET = (any remote network number),
   SADR = (any MAC address valid for the specified network),
   Who-Is-Request
2. WAIT **Internal Processing Fail Time**
3. 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)

9.34VT-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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-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)

9.36VT-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.

9.37RequestKey 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-Simple-ACK-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-Simple-ACK-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-Simple-ACK-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' = ($SK_{TD,IUT}$, BACnetAddress for TD)

Note: The 'List of Elements' shall be enciphered using PK_{IUT}.

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_{IUT}.

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-Simple-ACK-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, $SK_{TD,IUT}$, shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

```
1.  TRANSMIT Authenticate-Request,
        'Pseudo Random Number' =    (any valid pseudo random number)
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, $SK_{TD,IUT}$, shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

```
1.  TRANSMIT    Authenticate-Request,
     'Pseudo Random Number' =   (any valid pseudo random number),
     'Expected Invoke ID' =     (any valid invoke ID)
Note: The service request portion of this PDU shall be enciphered using SK_TD,IUT.
2.  TRANSMIT ReadProperty-Request,
     Invoke ID =              (the 'Expected Invoke ID' used in step 1),
     'Object Identifier' =    (any object supported by the IUT),
     'Property Identifier' =        (any supported property of the specified object)
3.  BEFORE Internal Processing Delay Time
     RECEIVE Authenticate-Request-ACK,
         'Modified Random Number' =       (the modified pseudo random number)
Note: The service request portion of this PDU shall be enciphered using SK_TD,IUT.
4.  BEFORE Internal Processing Delay Time
     RECEIVE ReadProperty-ACK,
         'Object Identifier' =      (the object identifier used in step 2),
         'Property Identifier' =          (the property identifier used in step 2),
         'Property Value' =         (the value of the property as specified in the EPICS)
```

9.38.4 Message Initiation Authentication

This clause defines the tests necessary to demonstrate support for executing the Authenticate service for the purpose of message initiation authentication.

9.38.4.1 Message Initiation Authentication by a Key Server

Executing message initiation authentication as a key server is covered by test 9.37

9.38.4.2 Message Initiation Authentication Peer-to-Peer

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 $SK_{TD,IUT}$.

Configuration Requirements: The IUT shall be configured with a private 56-bit cryptographic key. A secure session between the TD and IUT with a session key, $SK_{TD,IUT}$, shall have previously been established. This may be accomplished by executing the test in 9.38.1.

Test Steps:

1. TRANSMIT Authenticate-Request,
   'Pseudo Random Number' = (any valid pseudo random number),
   'Start Enciphered Session' = TRUE
2. BEFORE **Internal Processing Fail Time**
   RECEIVE Authenticate-Request,
   'Pseudo Random Number' = (any valid pseudo random number),
   'Expected Invoke ID' = (the invoke ID used in step 1)
3. TRANSMIT Authenticate-ACK,
   'Modified Random Number' = (the modified random number from step 2)
4. BEFORE **Acknowledgment Fail Time**
   RECEIVE Authenticate-ACK,
   'Modified Random Number' = (the modified random number from step1)

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, 18.8.9

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

Test Steps:

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

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

9.39.2 Unsupported Unconfirmed Services Test

Dependencies: None

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

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

Test Steps:

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

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

10 NETWORK LAYER PROTOCOL TESTS

10.1 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' =      (the value of the specified property as defined in the EPICS)
```

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.

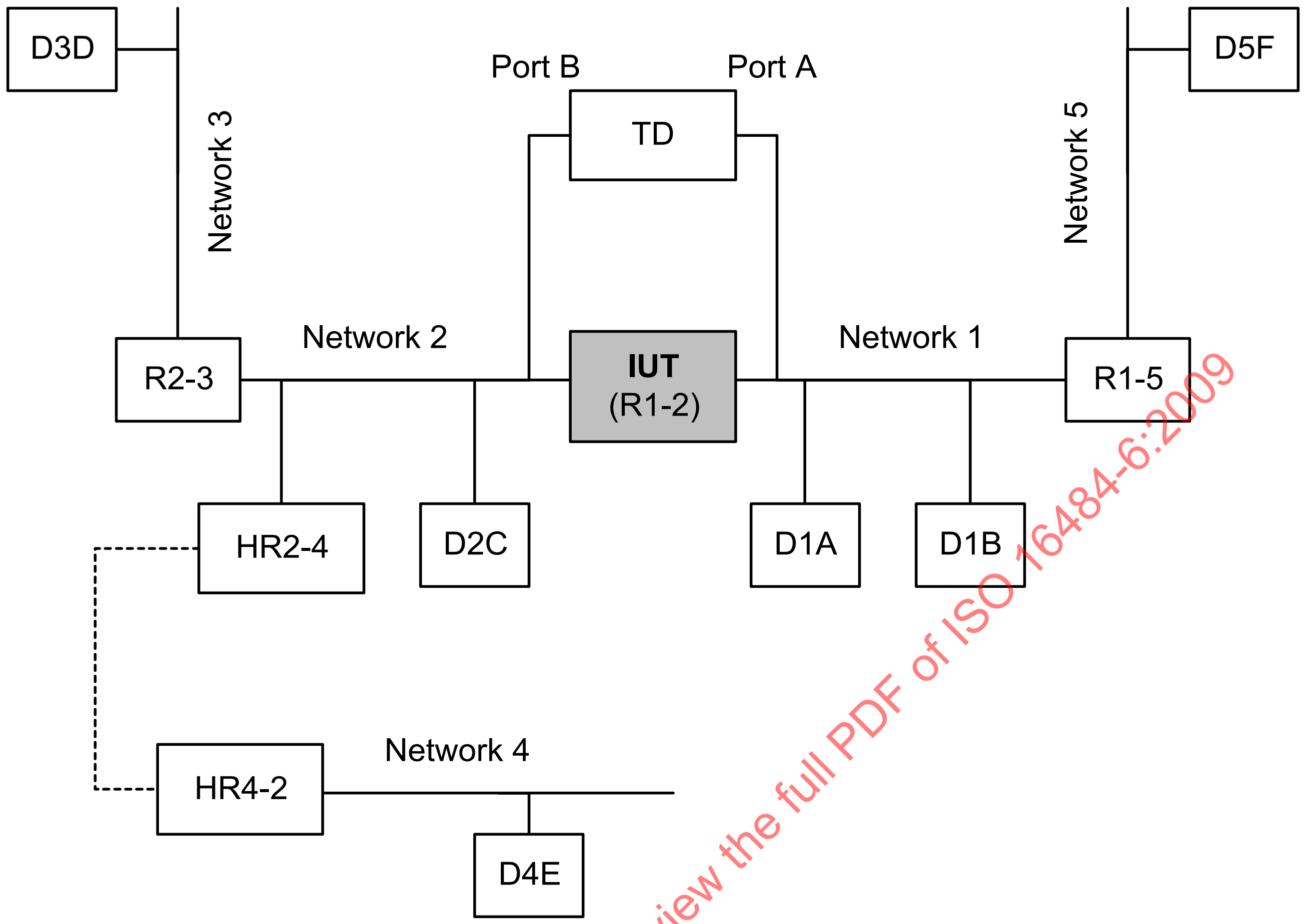

Figure 10-1. Logical internetwork configuration for router functionality tests

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
2.  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.

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,
   SOURCE = 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,
   SOURCE = 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
5. 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,
        DESTINATION = 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,
        DESTINATION = D1A,
        SOURCE = IUT,
        SNET = 2,
        SADR = HR2-4,
        Hop Count = (any integer x: 0 < x < 255),
        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,
   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)
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,
   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)
5. 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 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,
    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 B,
    DESTINATION = 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.

```
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,
   DESTINATION = IUT,
   SOURCE = D1A,
   DNET = 9,
   DADR = (any valid MAC address),
   Hop Count = 255,
   BACnet-Confirmed-Request-PDU,
   'Service Choice' = ReadProperty-Request,
   'Object Identifier' = (any object identifier),
   'Property Identifier' = (any property of the specified object)
2. RECEIVE PORT B,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = IUT,
   Who-Is-Router-To-Network,
   Network Number = 9
3. RECEIVE PORT A,
   DESTINATION = D1A,
   SOURCE = IUT,
   Reject-Message-To-Network,
   Reject Reason = 1, (unknown destination network)
   DNET = 9

10.2.2.7.2 Unknown Network Layer Message Type

Purpose: To verify that the IUT will reject a network layer message with an unknown message type.

Test Steps:

1. TRANSMIT PORT A,
   DESTINATION = IUT,
   SOURCE = D1A,
   Message Type = (any value from X'0A' to X'7F)
2. RECEIVE PORT A,
   DESTINATION = D1A,
   SOURCE = IUT,
   Reject-Message-To-Network,
   Reject Reason = 3 (unknown network layer message type),
   DNET = 1

10.2.2.7.3 Unknown Network Layer Message Type For Someone Else

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

BACnet Reference Clause: 6.6.3.5

Test Steps:

1. TRANSMIT PORT A, DA = IUT, SA = D1A,
   DNET = 2,
   DADR = D2C,
   Hop Count = 255,
   Message Type = (any value in the range reserved for use by ASHRAE)
2. RECEIVE Port B, DA = D2C, SA = IUT,
   SNET = 1,
   SADR = D1A,
   Message Type = (value from step 1)

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

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,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D2C,
        SOURCE = IUT,
        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)
3.  TRANSMIT PORT B,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D1A,
        SOURCE = IUT,
        SNET = 2,
        SADR = D2C,
        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)
```

10.2.3.3 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,
        DESTINATION = 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,
    DESTINATION = R2-3,
    SOURCE = 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,
    DESTINATION = IUT,
    SOURCE = 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 = R2-3,
    SOURCE = 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,
    DESTINATION = IUT,
    SOURCE = 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,
   DESTINATION = D2C,
   SOURCE = IUT,
   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)

10.2.3.6 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,
   DESTINATION = IUT,
   SOURCE = 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,
   DESTINATION = 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,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = R2-3,
        SOURCE = 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,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = D1A,
   DNET = GLOBAL BROADCAST,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = Who-Is
2. RECEIVE PORT B,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = 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,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = 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,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = 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,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = D2C,
   DNET = 1,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = Who-Is
2. RECEIVE PORT A,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = IUT,
   SNET = 2,
   SADR = D2C,
   Hop Count = (any integer x: 0 < x < 255),
   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,
   DESTINATION = LOCAL BROADCAST,
   SOURCE = D2C,
   DNET = 5,
   DLEN = 0,
   Hop Count = 255,
   BACnet-Unconfirmed-Request-PDU,
   'Service Choice' = Who-Is
2. RECEIVE PORT A,
   DESTINATION = R1-5,
   SOURCE = 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,
   DESTINATION = IUT, SOURCE = 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,
    DESTINATION = LOCAL BROADCAST,
    SOURCE = IUT,
    SNET = 3,
    SADR = D3D,
    Hop Count = (any integer x,: $0 < x < 254$),
    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,
    DESTINATION = IUT,
    SOURCE = 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,
    DESTINATION = R2-3,
    SOURCE = 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,
    DESTINATION = LOCAL BROADCAST,
    SOURCE = D2C,
    DNET = 3,
    DLEN = (any valid MAC address length),
    DADR = (any valid MAC address the agrees with DLEN),
    Hop Count = 255,
    ReadProperty-Request,
        'Object Identifier' = (any BACnet object),
        'Property Identifier' = (any property of the specified object)
2. CHECK (verify that the IUT does not forwarded 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.

```
1.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D2C,
        SOURCE = IUT,
        Priority = B'00',
        SNET = 1,
        SDR = 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)
3.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D2C,
        SOURCE = IUT,
        Priority = B'01',
        SNET = 1,
        SDR = 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 3),
        'Property Identifier' = (the property identifier used in step 3)
5.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D2C,
        SOURCE = IUT,
        Priority = B'10',
        SNET = 1,
        SDR = 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 5),
        'Property Identifier' = (the property identifier used in step 5)
7.  TRANSMIT PORT A,
        DESTINATION = IUT,
        SOURCE = 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,
        DESTINATION = D2C,
        SOURCE = IUT,
        Priority = B'11',
        SNET = 1,
        SDR = 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 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.

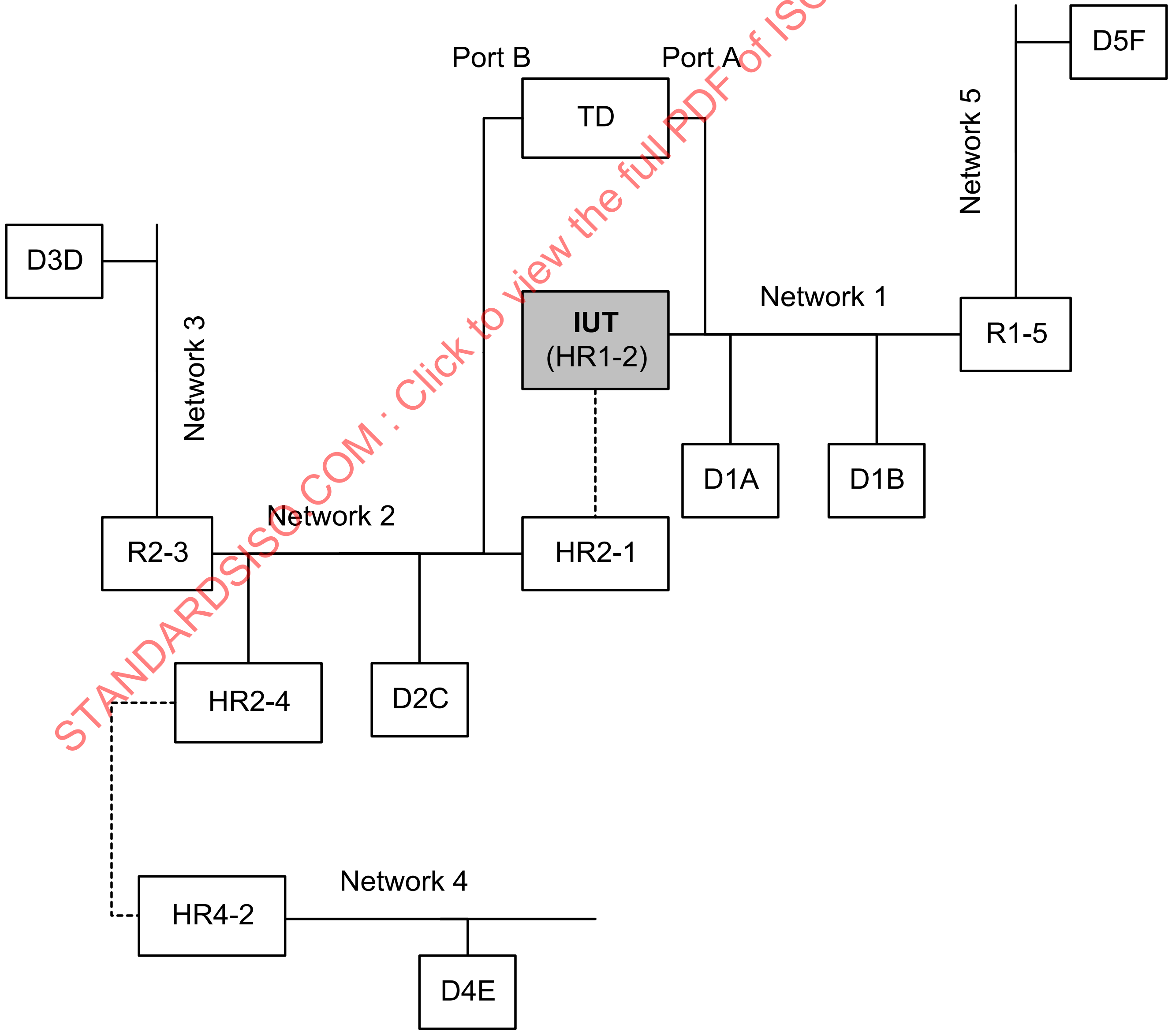

Figure 10-2. Logical internetwork configuration for half-router functionality tests

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'.

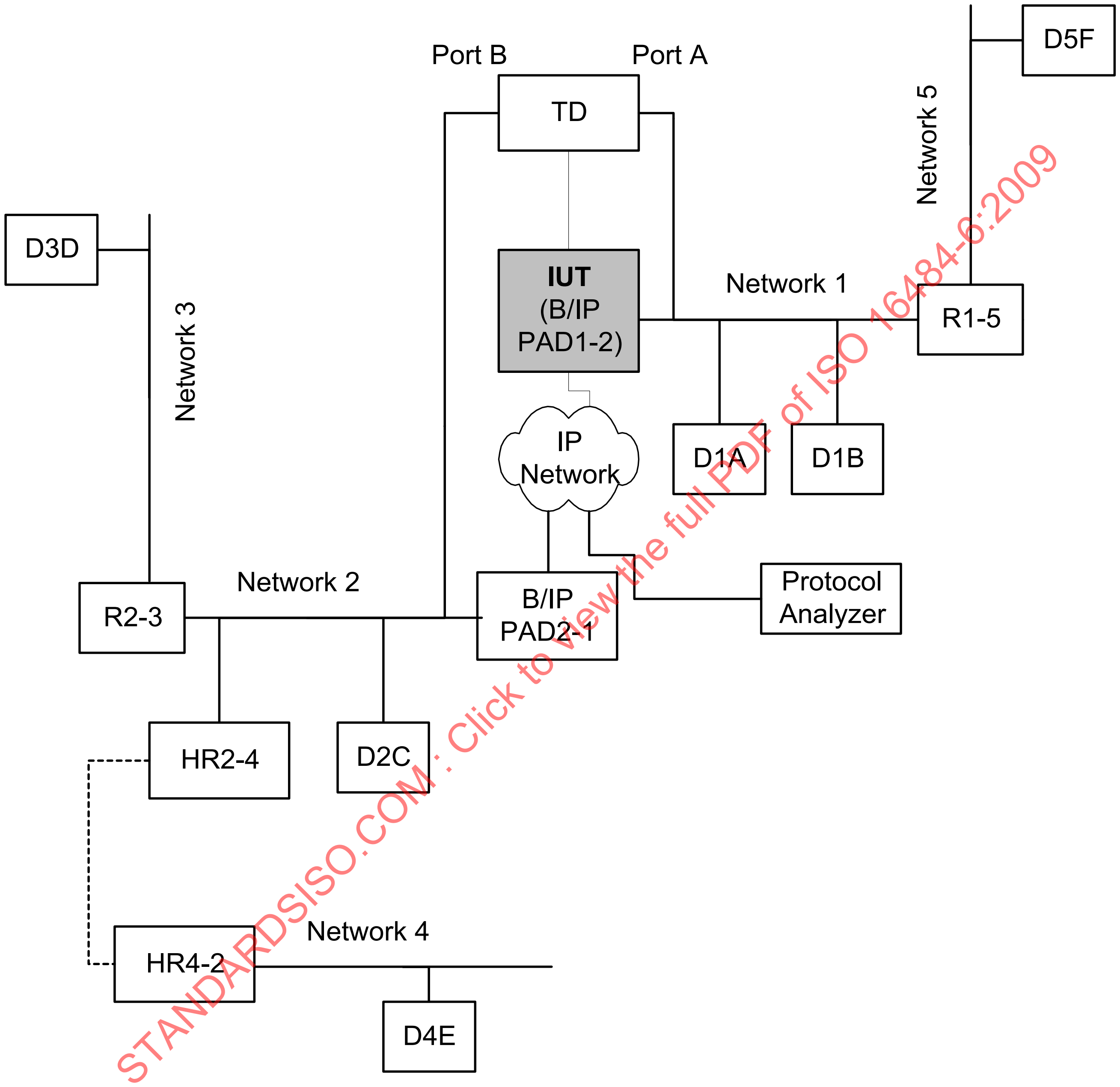

Figure 10-3. Logical internetwork configuration for B/IP router functionality tests

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)

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,
            }
        }
    }
```

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 $\mathbf{T}_{\mathbf{reply\_delay}}$, there must be no inter-frame gaps greater than the interval specified by $\mathbf{T}_{\mathbf{frame\_gap}}$, and after receiving a Token or Poll For Master frame the IUT must begin transmitting the next frame within the time specified by $\mathbf{T}_{\mathbf{usage\_delay}}$.

12.1.1.1 Test Setup

The IUT shall be set to MAC address 2 with $\mathbf{N}_{\mathbf{max\_master}}$ set to 127 and $\mathbf{N}_{\mathbf{max\_info\_frames}}$ 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: In this test, IUT initialization is defined to be complete when the IUT emits a Reply To Poll For Master frame.

```
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

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 $\mathbf{T}_{\mathbf{frame\_abort}}$ 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≤ x ≤ 50),
            'Data' = (x number of octets)
6.  BEFORE (T_reply_delay) {
        IF (anything received) THEN
            ERROR "MS/TP Framing error undetected."
        }
7.  TRANSMIT
        Test_Request,
            'Length' = (x where 1≤ x ≤ 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: During the transmission of step 5, one octet in the Data field shall be transmitted with a logical zero in the stop bit position.

```
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≤ x ≤ 50),
            'Data' = (x number of octets)
6.  BEFORE (T_reply_delay) {
        IF (anything received) THEN
            ERROR "MS/TP Framing error undetected."
        }
7.  TRANSMIT
        Test_Request,
            'Length' = (x where 1≤ x ≤ 50),
            'Data' = (x number of octets)
8.  RECEIVE
        Test_Response
```

12.1.1.3.1.4 Bad Header CRC

Purpose: To verify that the Receive Frame State Machine detects and rejects frames with an incorrect Header CRC. The following Receive Frame State Machine transition is verified by this test:

HEADER CRC: BadCRC

Test Steps:

```
1.  MAKE (IUT turned off or otherwise reset)
2.  MAKE (IUT turned on or otherwise started)
3.  TRANSMIT
        Test_Request
4.  RECEIVE
        Test_Response
5.  TRANSMIT
        Test_Request
            'Header CRC' = (any incorrect value)
6.  BEFORE (T_reply_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 ($\mathbf{T}_{\mathbf{reply\_delay}}$) {
        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: During the transmission of step 5, one octet in the Header shall be transmitted with a logical zero in the stop bit position.

```
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 ($\mathbf{T}_{\mathbf{reply\_delay}}$) {
        IF (anything received) THEN
            ERROR "MS/TP Frame Header framing error undetected."
        }
7.  TRANSMIT
        Test_Request
8.  RECEIVE
        Test_Response
```

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: 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.

```
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 ($\mathbf{T}_{\mathbf{reply\_delay}}$) {
        IF (anything received) THEN
            ERROR "MS/TP Frame Header timeout undetected."
        }
7.  TRANSMIT
        Test_Request
8.  RECEIVE
        Test_Response
```

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: 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.

```
1.  MAKE (IUT turned off or otherwise reset)
2.  MAKE (IUT turned on or otherwise started)
3.  TRANSMIT
        Test_Request
4.  RECEIVE
        Test_Response
```

```
5.  TRANSMIT
        Test_Request
6.  BEFORE (T_reply_delay) {
        IF (anything received) THEN
            ERROR "MS/TP Preamble receive error undetected."
        }
7.  TRANSMIT
        Test_Request
8.  RECEIVE
    Test_Response
```

12.1.1.3.1.10 Eat An Octet

Purpose: To verify that the Receive Frame State Machine correctly rejects an initial preamble octet that does not have the value X'55'. The following Receive Frame State Machine transition is verified by this test:

IDLE:	EatAnOctet

Test Steps:

```
1.  MAKE (IUT turned off or otherwise reset)
2.  MAKE (IUT turned on or otherwise started)
3.  TRANSMIT
        Test_Request
4.  RECEIVE
        Test_Response
5.  TRANSMIT
        Test Request,
            'Preamble' = (any value other than X'55), X'FF'
6.  BEFORE (T_reply_delay) {
        IF (anything received) THEN
            ERROR "MS/TP Frame incorrect Preamble undetected."
        }
7.  TRANSMIT
        Test_Request
8.  RECEIVE
        Test_Response
```

12.1.1.3.1.11 Frame Too Long

Purpose: To verify the Receive Frame state machine error check for a frame too large for the IUT. This tests the following Receive Frame State machine transition:

HEADER CRC:	FrameTooLong

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 ($T_{reply\_delay}$) {
        IF (anything received) THEN
            ERROR "MS/TP Frame Header framing error undetected."
        }
7.  TRANSMIT
        Test_Request,
            'Length' = 0
8.  RECEIVE
        Test_Reply
```

12.1.1.3.2 Tests with Response

This clause defines the test cases where data link errors are corrected or which cause a response to be issued.

12.1.1.3.2.1 Repeated Preamble1

Purpose: To verify the Receive Frame state machine check for a repeated first preamble octet. The following Receive Frame State Machine transition is verified by this test:

PREAMBLE: RepeatedPreamble1

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≤ x ≤ 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 ($\mathbf{T}_{\mathbf{no\_token}}$ – 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
10. WAIT $T_{usage\_timeout}$
11. RECEIVE
        DA = IUT+2,
        Poll For Master
12. WAIT $T_{usage\_timeout}$
13. 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 $\mathbf{T}_{\mathbf{usage\_timeout}}$
        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 $\mathbf{T}_{\mathbf{usage\_timeout}}$
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 $\mathbf{T}_{\mathbf{usage\_timeout}}$
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 $\mathbf{T}_{\mathbf{usage\_timeout}}$
        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 $T_{usage\_timeout}$
11. RECEIVE
        DA = 6,
        Token
12. 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
        }
13. TRANSMIT
        DA = 127,
        SA = 6,
        Poll For Master
14. WAIT $T_{usage\_timeout}$
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 $\mathbf{T}_{\mathbf{usage\_timeout}}$
            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
            }
        TRANSMIT
            DA = 127,
            SA = 6,
            Poll For Master
        WAIT $\mathbf{T}_{\mathbf{usage\_timeout}}$
        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)
        }
    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 ($T_{reply\_delay}$) {
        IF (anything received) THEN
            ERROR "Response issued by IUT to BACnet Data Not Expecting Reply."
        }
6.  TRANSMIT
        Poll For Master
7.  RECEIVE
        Reply To Poll For Master
```

12.1.1.6.2 Received Invalid Frame

Purpose: To verify that the Master Node Finite State Machine in the IDLE state properly handles an invalid frame. Transitions verified:

IDLE:	ReceivedInvalidFrame
IDLE:	Received PFM

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 ($T_{reply\_delay}$) {
        IF (frame received) THEN
            ERROR "Invalid Frame accepted by IUT."
        }
6.  TRANSMIT
        Poll For Master
7.  RECEIVE
        Reply To Poll For Master
```

12.1.1.6.3 Unwanted Frame Tests

The tests described by this clause verify that the IUT in its IDLE state rejects frames not addressed to it, or frames that were illegally broadcast. Clause 12.1.1.6.3.1 verifies a transition that should never occur since the Receive Frame State Machine rejects messages for other devices in its NotForUs transition. Transitions verified:

IDLE:	ReceivedUnwantedFrame
IDLE:	Received PFM

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 ($\mathbf{T}_{reply\_delay}$) {
        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 ($\mathbf{T}_{reply\_delay}$) {
        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 ($\mathbf{T}_{\mathbf{reply\_delay}}$) {
        IF (frame received) THEN
            ERROR "Broadcast Test Request frame accepted by IUT."
        }
6.  TRANSMIT
        Poll For Master
7.  RECEIVE
        Reply To Poll For Master
```

12.1.1.7 Sole Master Tests

These tests verify the ability of the IUT to properly recognize itself to be the only master on the MS/TP LAN, and to identify when another master enters.

12.1.1.7.1 Drop Token

Purpose: To verify that the IUT recognizes that the token has been lost and generates another. This tests the transitions:

IDLE:	LostToken
NO_TOKEN:	GenerateToken

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 ($\mathbf{T}_{\mathbf{no\_token}}$ + (2 * $\mathbf{T}_{\mathbf{slot}}$)) {
        IF (frame received) THEN
            ERROR "Lost Token detected too early by IUT."
        }
5.  BEFORE ($\mathbf{T}_{\mathbf{slot}}$) 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 ($\mathbf{T}_{\mathbf{usage\_timeout}}$) {
        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 ($\mathbf{T}_{\mathbf{usage\_timeout}}$) {
            IF (frame received) THEN
                ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
    RECEIVE
        DA = X,
        Poll For Master
    }
```

12.1.1.7.4 Declare Sole Master (a)

Purpose: To verify that the IUT has declared itself the sole master but is still conducting a poll to find another master. This verifies the transitions:

DONE_WITH_TOKEN:	Solemaster (a)
DONE_WITH_TOKEN:	SendMaintenancePFM

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 (T_usage_timeout) {
            IF (frame received) THEN
                ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
        RECEIVE
            DA = X,
            Poll For Master
    }
6.  BEFORE (T_usage_timeout) {
        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
    }
7.  RECEIVE
        DA = IUT+1,
        Poll For Master
```

12.1.1.7.5 New Master Enters

Purpose: To verify that the IUT recognizes the presence of another master.

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_usage_timeout) {
            IF (frame received) THEN
                ERROR "IUT didn't wait long enough for Reply To Poll For Master."
            }
        RECEIVE
            DA = X,
            Poll For Master
        }
6.  BEFORE (T_usage_timeout) {
        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
7.  RECEIVE
        DA = IUT+1,
        Poll For Master
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_usage_timeout) {
            IF (frame received) THEN
                ERROR "IUT didn't wait long enough for Reply To Poll For Master."
            }
        RECEIVE
            DA = X,
            Poll For Master
        }
6.  BEFORE ($\mathbf{T}_{usage\_timeout}$) {
        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
7.  RECEIVE
        DA = IUT+1,
        Poll For Master
8.  TRANSMIT
        SA = IUT+1,
        Reply To Poll For Master
9.  RECEIVE
        DA = IUT+1
        Token
10. BEFORE ($\mathbf{T}_{usage\_timeout}$) {
        IF (frame received) THEN
            ERROR "IUT didn't wait long enough for Reply To Poll For Master."
        }
11. RECEIVE
        DA = IUT+2,
        Poll For Master
```

12.1.1.7.7 DeclareSoleMaster (b)

Purpose: To verify that the IUT undergoes the transitions necessary for it to declare itself the sole master when it receives an invalid frame in response to the transmitted Poll For Master Frame. The transitions tested are:

POLL_FOR_MASTER:	SendNextPFM
POLL_FOR_MASTER:	DeclareSoleMaster (b)

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 $T_{reply\_timeout}$
9.  TRANSMIT
        SA = 0,
        DA = IUT,
        Poll For Master
10. RECEIVE
        DA = 0,
        SA = IUT,
        Reply To Poll For Master
```

12.1.1.8.2 Broadcast

Purpose: To detect a second token when a broadcast message appears out of turn.

Test Steps:

```
1.  MAKE (IUT turned off or otherwise reset)
2.  MAKE (IUT turned on or otherwise started)
3.  TRANSMIT
        Test_Request
4.  RECEIVE
        Test_Response
5.  MAKE (IUT to generate a confirmed service request)
6.  WHILE (BACnet Data Expecting Reply frame not received) DO {
        }
7.  TRANSMIT
        DA = LOCAL BROADCAST,
        BACnet Data Not Expecting Reply
8.  WAIT $T_{reply\_timeout}$
9.  TRANSMIT
        SA = 0,
        DA = IUT,
        Poll For Master
10. RECEIVE
        DA = 0,
        SA = IUT,
        Reply To Poll For Master
```

12.1.1.8.3 Token

Purpose: To detect a second token when it gets passed to the IUT already holding a token.

Test Steps:

```
1.  MAKE (IUT turned off or otherwise reset)
2.  MAKE (IUT turned on or otherwise started)
3.  TRANSMIT
        Test_Request
4.  RECEIVE
        Test_Response
5.  MAKE (IUT to generate a confirmed service request)
6.  WHILE (BACnet Data Expecting Reply frame not received) DO {
        }
7.  TRANSMIT
        Token
8.  WAIT $\mathbf{T}_{\mathbf{reply\_timeout}}$
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 $\mathbf{T}_{\mathbf{reply\_timeout}}$
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: In these steps, if the Who-Is service is not supported, substitute the Who-Has and I-Have services.

```
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
```

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: The ReadProperty-ACK of step 6 shall be an appropriate and correct response to the request of step 5.

```
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."
        }
```

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)
```