

Building automation and control systems (BACS) —

Part 6: Data communication conformance testing

ICS 35.240.99; 91.040.01; 97.120

National foreword

This British Standard is the UK implementation of EN ISO 16484-6:2009. It supersedes BS EN ISO 16484-6:2005 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee RHE/16, Performance requirements for control systems.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2009.

© BSI 2009

ISBN 978 0 580 65648 4

Amendments/corrigenda issued since publication

Date	Comments

English Version

**Building automation and control systems (BACS) - Part 6: Data
communication conformance testing (ISO 16484-6:2009)**

Systèmes d'automatisation et de gestion technique du
bâtiment - Partie 6: Essais de conformité de la
communication de données (ISO 16484-6:2009)

Systeme der Gebäudeautomation - Teil 6:
Datenübertragungsprotokoll - Konformitätsprüfung (ISO
16484-6:2009)

This European Standard was approved by CEN on 21 February 2009.

CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN Management Centre or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the CEN Management Centre has the same status as the official versions.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.



EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: Avenue Marnix 17, B-1000 Brussels

Foreword

This document (EN ISO 16484-6:2009) has been prepared by Technical Committee ISO/TC 205 "Building environment design" in collaboration with Technical Committee CEN/TC 247 "Building Automation, Controls and Building Management", the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by September 2009, and conflicting national standards shall be withdrawn at the latest by September 2009.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

This document supersedes EN ISO 16484-6:2005.

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

Endorsement notice

The text of ISO 16484-6:2009 has been approved by CEN as a EN ISO 16484-6:2009 without any modification.

Contents

CLAUSE	PAGE
1 Scope	1
2 Relationship between this part of ISO 16484 and ANSI/ASHRE 135.1-2007	1
3 Terms, definitions and abbreviated terms	1
4 ELECTRONIC PICS FILE FORMAT	3
4.1 Character Encoding	3
4.2 Structure of EPICS Files	4
4.3 Character Strings	4
4.4 Notational Rules for Parameter Values	4
4.5 Sections of the EPICS File	6
5 EPICS CONSISTENCY TESTS	22
6 CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS	23
6.1 TCSL Components	24
6.2 TCSL Statements	25
6.3 Time Dependencies	29
6.4 BACnet References	29
7 OBJECT SUPPORT TESTS	30
7.1 Read Support for Properties in the Test Database	30
7.2 Write Support for Properties in the Test Database	30
7.3 Object Functionality Tests	31
8 APPLICATION SERVICE INITIATION TESTS	99
8.1 AcknowledgeAlarm Service Initiation Tests	99
8.2 ConfirmedCOVNotification Service Initiation Tests	100
8.3 UnconfirmedCOVNotification Service Initiation Tests	109
8.4 ConfirmedEventNotification Service Initiation Tests	111
8.5 UnconfirmedEventNotification Service Initiation Tests	141
8.6 GetAlarmSummary Service Initiation Tests	147
8.7 GetEnrollmentSummary Service Initiation Tests	147
8.8 GetEventInformation Service Initiation Tests	149
8.9 LifeSafetyOperation Service Initiation Tests	149
8.10 SubscribeCOV Service Initiation Tests	150
8.11 SubscribeCOVProperty Service Initiation Tests	151
8.12 AtomicReadFile Service Initiation Tests	152
8.13 AtomicWriteFile Service Initiation Tests	152
8.14 AddListElement Service Initiation Tests	153
8.15 RemoveListElement Service Initiation Tests	153
8.16 CreateObject Service Initiation Tests	154
8.17 DeleteObject Service Initiation Tests	155
8.18 ReadProperty Service Initiation Tests	155
8.19 ReadPropertyConditional Service Initiation Tests	156
8.20 ReadPropertyMultiple Service Initiation Tests	156
8.21 ReadRange Service Initiation Tests	157
8.22 WriteProperty Service Initiation Tests	159
8.23 WritePropertyMultiple Service Initiation Tests	159
8.24 DeviceCommunicationControl Service Initiation Tests	161
8.25 ConfirmedPrivateTransfer Service Initiation Test	162
8.26 UnconfirmedPrivateTransfer Service Initiation Test	163
8.27 ReinitializeDevice Service Initiation Tests	163
8.28 ConfirmedTextMessage Service Initiation Tests	164
8.29 UnconfirmedTextMessage Service Initiation Tests	165
8.30 TimeSynchronization Service Initiation Tests	166
8.31 UTCTimeSynchronization Service Initiation Tests	166
8.32 Who-Has Service Initiation Tests	166
8.33 I-Have Service Initiation Tests	167

8.34	Who-Is Service Initiation Tests	167
8.35	I-Am Service Initiation Tests	168
8.36	VT-Open Service Initiation Tests	168
8.37	VT-Close Service Initiation Tests	169
8.38	VT-Data Service Initiation Tests	170
8.39	RequestKey Service Initiation Tests	172
8.40	Authenticate Service Initiation Tests	173
9	APPLICATION SERVICE EXECUTION TESTS	176
9.1	AcknowledgeAlarm Service Execution Tests	177
9.2	ConfirmedCOVNotification Service Execution Tests	189
9.3	UnconfirmedCOVNotification Service Execution Tests	193
9.4	ConfirmedEventNotification Service Execution Tests	193
9.5	UnconfirmedEventNotification Service Execution Tests	194
9.6	GetAlarmSummary Service Execution Tests	194
9.7	GetEnrollmentSummary Service Execution Tests	195
9.8	GetEventInformation Service Execution Tests	199
9.9	LifeSafetyOperation Service Execution Test	201
9.10	SubscribeCOV Service Execution Tests	202
9.11	SubscribeCOVProperty Service Execution Tests	207
9.12	AtomicReadFile Service Execution Tests	214
9.13	AtomicWriteFile Service Execution Tests	220
9.14	AddListElement Service Execution Tests	230
9.15	RemoveListElement Service Execution Tests	232
9.16	CreateObject Service Execution Tests	234
9.17	DeleteObject Service Execution Tests	238
9.18	ReadProperty Service Execution Tests	239
9.19	ReadPropertyConditional Service Execution Tests	241
9.20	ReadPropertyMultiple Service Execution Tests	242
9.21	ReadRange Service Execution Tests	249
9.22	WriteProperty Service Execution Tests	251
9.23	WritePropertyMultiple Service Execution Tests	256
9.24	DeviceCommunicationControl Service Execution Test	264
9.25	ConfirmedPrivateTransfer Service Execution Tests	268
9.26	UnconfirmedPrivateTransfer Service Execution Tests	269
9.27	ReinitializeDevice Service Execution Tests	269
9.28	ConfirmedTextMessage Service Execution Tests	271
9.29	UnconfirmedTextMessage Service Execution Tests	273
9.30	TimeSynchronization Service Execution Tests	273
9.31	UTCTimeSynchronization Service Execution Tests	274
9.32	Who-Has Service Execution Tests	275
9.33	Who-Is Service Execution Tests	280
9.34	VT-Open Service Execution Tests	283
9.35	VT-Close Service Execution Tests	284
9.36	VT-Data Service Execution Tests	285
9.37	RequestKey Service Execution Test	286
9.38	Authenticate Service Execution Tests	288
9.39	General Testing of Service Execution	292
10	NETWORK LAYER PROTOCOL TESTS	293
10.1	Processing Application Layer Messages Originating from Remote Networks	293
10.2	Router Functionality Tests	293
10.3	Half-Router Functionality Tests	317
10.4	B/IP PAD Tests	323
10.5	Initiating Network Layer Messages	325
11	LOGICAL LINK LAYER PROTOCOL TESTS	327
11.1	UI Command and Response	327
11.2	XID Command and Response	327
11.3	TEST Command and Response	328
12	DATA LINK LAYER PROTOCOLS TESTS	329
12.1	MS/TP State Machine Tests	329
12.2	PTP State Machine Tests	381

13	SPECIAL FUNCTIONALITY TESTS.....	417
13.1	Segmentation	417
13.2	Time Master	426
13.3	Character Sets	427
13.4	Malformed PDUs	427
13.5	Slave Proxy Tests.....	428
14	BACnet/IP Functionality Tests	431
14.1	Non-BBMD B/IP Device	431
14.2	Non-BBMD B/IP device Device with a Server Application.....	433
14.3	Broadcast Distribution Table Operations	433
14.4	Foreign Device Table Operations (Negative Tests)	436
14.5	BACnet Broadcast Management (No Foreign Device Table, No Applications).....	437
14.6	Foreign Device Management	438
14.7	Broadcast Management (BBMD, Foreign Devices, Local Application).....	440
15	Reporting Test Results	443
ANNEX A - Example EPICS (INFORMATIVE)		444

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 parenthesis, with the elements of the list separated by commas. If an element is itself a constructed value, then that element shall be enclosed in curly brackets;
- (d) array values are represented enclosed in curly brackets, with the elements of the array separated by commas. If an element is itself a constructed value, then that element shall be enclosed in curly brackets.

4.4.2 Specifying Limits on Parameter Values

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

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

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

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

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

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

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

4.5 Sections of the EPICS File

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

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

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

An example EPICS file may be found in Annex A.

4.5.1 General Information Sections

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

Vendor Name: "□"↵

Product Name: "□"↵

Product Model Number: "□"↵

Product Description: "□"↵

4.5.2 Conformance Sections

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

4.5.2.1 BIBBs Supported

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

BIBBs Supported: ↵

```
{↵  
  □↵  
}
```

The BIBBs may be any of:

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: {□, □, □}
  acknowledged-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> <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 WritePropertyMultiple services. The test is performed once using WriteProperty and once using WritePropertyMultiple. When writing to array properties, the whole array shall be written without using an array index, where possible.

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

Test Steps:

1. REPEAT X = (all objects in the IUT's database) DO {
 - REPEAT Y = (all writable properties in object X) DO {
 - REPEAT Z = (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)
 }
ELSE
 MAKE (Out_Of_Service FALSE)
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) than the priority that is currently controlling Present_Value.

Test Steps:

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

7.3.1.5 Minimum_On_Time

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

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

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

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

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

Test Steps:

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

7.3.1.6 Override of Minimum Time

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

BACnet Reference Clause: 19.2.

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

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

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

Test Steps:

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

Notes to Tester: If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1 through 3 and ACTIVE in steps 4 through 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' = ($T: (T_{ELAPSED} + 30) \leq T \leq (T_{ELAPSED} + 30 + \text{Internal Processing Fail Time})$)
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, TLOC)
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 TLOC)
```

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),

```