

**DIN EN ISO 16484-6**

ICS 35.240.99; 91.140.01

**Entwurf**

Einsprüche bis 2014-04-14  
Vorgesehen als Ersatz für  
DIN EN ISO 16484-6:2009-11;  
Ersatz für  
E DIN EN ISO 16484-6:2013-09

**Systeme der Gebäudeautomation –  
Teil 6: Datenübertragungsprotokoll –  
Konformitätsprüfung (ISO/FDIS 16484-6:2013);  
Englische Fassung FprEN ISO 16484-6:2013, nur auf CD-ROM**

Building automation and control systems (BACS) –  
Part 6: Data communication conformance testing (ISO/FDIS 16484-6:2013);  
English version FprEN 16484-6:2013, only on CD-ROM

Systèmes d'automatisation et de gestion technique du bâtiment –  
Partie 6: Essais de conformité de la communication de données  
(ISO/FDIS 16484-6:2013);  
Version anglaise FprEN ISO 16484-6:2013, seulement en CD-ROM

**Anwendungswarnvermerk**

Dieser Norm-Entwurf mit Erscheinungsdatum 2014-02-14 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfes besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal des DIN unter [www.entwuerfe.din.de](http://www.entwuerfe.din.de) bzw. für Norm-Entwürfe der DKE auch im Norm-Entwurfs-Portal der DKE unter [www.entwuerfe.normenbibliothek.de](http://www.entwuerfe.normenbibliothek.de), sofern dort wiedergegeben;
- oder als Datei per E-Mail an [nhrs@din.de](mailto:nhrs@din.de) möglichst in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter [www.din.de/stellungnahme](http://www.din.de/stellungnahme) oder für Stellungnahmen zu Norm-Entwürfen der DKE unter [www.dke.de/stellungnahme](http://www.dke.de/stellungnahme) abgerufen werden;
- oder in Papierform an den Normenausschuss Heiz- und Raumluftechnik (NHRS) im DIN, 10772 Berlin (Hausanschrift: Burggrafenstr. 6, 10787 Berlin).

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevanten Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 576 Seiten

Normenausschuss Heiz- und Raumluftechnik (NHRS) im DIN  
DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

## Nationales Vorwort

Dieses Dokument (FprEN ISO 16484-6:2013) wurde vom Technischen Komitee ISO/TC 205 „Building environment design“ in Zusammenarbeit mit dem Technischen Komitee CEN/TC 247 „Gebäudeautomation und Gebäudemanagement“ (Sekretariat: SNV, Schweiz) unter Beteiligung deutscher Experten erarbeitet.

Das zuständige deutsche Gremium ist der Arbeitsausschuss NA 041-03-68 AA „Gebäudeautomation und -kommunikation (SpA CEN/TC 247/WG 4)“ im Normenausschuss Heiz- und Raumlufttechnik (NHRS).

Dieses Dokument (FprEN ISO 16484-6:2013) übernimmt den englischen Originaltext des ANSI/ASHRAE Standard 135.1 „BACnet — A Data Communication Protocol for Building Automation and Control Networks“, Ausgabe 2011. In derselben Weise sollen zwischenzeitlich veröffentlichte bzw. noch zu veröffentlichende Änderungen und Erweiterungen von ANSI/ASHRAE Standard 135.1 bei der Fortschreibung von EN ISO 16484-6 übernommen werden. Der Inhalt dieses Dokumentes befindet sich auf der beiliegenden CD-ROM.

Der NHRS war für das DIN Deutsches Institut für Normung e. V. an der Erstellung dieser Norm beteiligt. Da davon auszugehen ist, dass die Fachleute der Kommunikationstechnik sich bereits mit der englischen Fassung befasst haben, wurde von einer Übersetzung ins Deutsche abgesehen. Nach dem DIN-Präsidialbeschluss 13/1983 ist insbesondere auf dem Gebiet der Informatik die Veröffentlichung in Englisch zulässig, insbesondere wenn dadurch Fehler bei der Übertragung vermieden werden können.

Dieses Dokument (FprEN ISO 16484-6:2013) stellt ein genormtes Testverfahren zur Verfügung, um eine Implementierung (de: Umsetzung) des BACnet-Protokolls auf ihre Konformität zur Protokoll-Norm DIN EN ISO 16484-5 zu überprüfen. Dies geschieht für alle Funktionen, die vom Hersteller im „Protocol Implementation Conformance Statement“, dem so genannten PICS, für diese Implementierung angegeben worden sind.

## Änderungen

Gegenüber DIN EN ISO 16484-6:2009-11 wurden folgende Änderungen vorgenommen:

- a) Inhalt technisch überarbeitet.

## Nationaler Anhang NA (informativ)

### Begriffe und Definitionen

Im Folgenden wird ISO 16484-6:2013, Abschnitt 3 ins Deutsche übersetzt. Die Benennung der folgenden Begriffe und Abkürzungen ist identisch mit der Benennung in der englischen Fassung.

### 3 Begriffe und Abkürzungen

Für die Anwendung dieses Dokuments gelten die folgenden Begriffe.

#### 3.1

##### **Lokales Netzwerk**

(en: local network)

Netzwerk, mit dem ein BACnet Gerät direkt verbunden ist

#### 3.2

##### **Remote Netzwerk**

(en: remote network)

Netzwerk, das von einem BACnet Gerät nur über einen oder mehrere Router zugänglich ist

#### 3.3

##### **Testdatenbank**

(en: test database)

Datenbank von BACnet-Funktionalitäten und Objekten erstellt durch das Auslesen des Inhaltes eines EPICS

### Abkürzungen

| Abkürzung (englisch) | Bedeutung (englisch)                                     | Bedeutung (deutsch)                                            |
|----------------------|----------------------------------------------------------|----------------------------------------------------------------|
| <b>BNF</b>           | Backus-Naur Form syntax                                  | Backus-Naur Form Syntax                                        |
| <b>EPICS</b>         | electronic protocol implementation conformance statement | elektronischen Protokoll-Implementierung Konformitätserklärung |
| <b>IUT</b>           | implementation under test                                | Umsetzung im Test                                              |
| <b>TCSL</b>          | testing and conformance scripting language               | Prüfungs- und Konformitätsskriptsprache                        |
| <b>TD</b>            | testing device                                           | Testgerät                                                      |
| <b>TPI</b>           | text protocol information                                | Text Protokoll Information                                     |

## Contents

## Page

|                                                                                |     |
|--------------------------------------------------------------------------------|-----|
| Foreword .....                                                                 | vii |
| 1 Scope .....                                                                  | 1   |
| 2 Relationship between this part of ISO 16484 and ANSI/ASHRAE 135.1-2011 ..... | 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 .....                                                | 24  |
| 6 CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS .....                    | 25  |
| 6.1 TCSL Components .....                                                      | 25  |
| 6.2 TCSL Statements .....                                                      | 26  |
| 6.3 Time Dependencies .....                                                    | 31  |
| 6.4 BACnet References .....                                                    | 32  |
| 6.5 TD Requirements .....                                                      | 32  |
| 7 OBJECT SUPPORT TESTS .....                                                   | 32  |
| 7.1 Read Support for Properties in the Test Database .....                     | 32  |
| 7.2 Write Support for Properties in the Test Database .....                    | 33  |
| 7.3 Object Functionality Tests .....                                           | 36  |
| 8 APPLICATION SERVICE INITIATION TESTS .....                                   | 132 |
| 8.1 AcknowledgeAlarm Service Initiation Tests .....                            | 132 |
| 8.2 ConfirmedCOVNotification Service Initiation Tests .....                    | 133 |
| 8.3 UnconfirmedCOVNotification Service Initiation Tests .....                  | 143 |
| 8.4 ConfirmedEventNotification Service Initiation Tests .....                  | 145 |
| 8.5 UnconfirmedEventNotification Service Initiation Tests .....                | 179 |
| 8.6 GetAlarmSummary Service Initiation Tests .....                             | 186 |
| 8.7 GetEnrollmentSummary Service Initiation Tests .....                        | 186 |
| 8.8 GetEventInformation Service Initiation Tests .....                         | 187 |
| 8.9 LifeSafetyOperation Service Initiation Tests .....                         | 188 |
| 8.10 SubscribeCOV Service Initiation Tests .....                               | 189 |
| 8.11 SubscribeCOVProperty Service Initiation Tests .....                       | 190 |
| 8.12 AtomicReadFile Service Initiation Tests .....                             | 190 |
| 8.13 AtomicWriteFile Service Initiation Tests .....                            | 191 |
| 8.14 AddListElement Service Initiation Tests .....                             | 192 |
| 8.15 RemoveListElement Service Initiation Tests .....                          | 192 |
| 8.16 CreateObject Service Initiation Tests .....                               | 193 |
| 8.17 DeleteObject Service Initiation Tests .....                               | 194 |
| 8.18 ReadProperty Service Initiation Tests .....                               | 194 |
| 8.19 ReadPropertyConditional Service Initiation Tests .....                    | 196 |
| 8.20 ReadPropertyMultiple Service Initiation Tests .....                       | 197 |
| 8.21 ReadRange Service Initiation Tests .....                                  | 199 |
| 8.22 WriteProperty Service Initiation Tests .....                              | 203 |
| 8.23 WritePropertyMultiple Service Initiation Tests .....                      | 205 |
| 8.24 DeviceCommunicationControl Service Initiation Tests .....                 | 207 |
| 8.25 ConfirmedPrivateTransfer Service Initiation Test .....                    | 208 |
| 8.26 UnconfirmedPrivateTransfer Service Initiation Test .....                  | 209 |
| 8.27 ReinitializeDevice Service Initiation Tests .....                         | 209 |

|      |                                                                              |     |
|------|------------------------------------------------------------------------------|-----|
| 8.28 | ConfirmedTextMessage Service Initiation Tests .....                          | 210 |
| 8.29 | UnconfirmedTextMessage Service Initiation Tests .....                        | 211 |
| 8.30 | TimeSynchronization Service Initiation Tests .....                           | 212 |
| 8.31 | UTCTimeSynchronization Service Initiation Tests .....                        | 212 |
| 8.32 | Who-Has Service Initiation Tests .....                                       | 212 |
| 8.33 | I-Have Service Initiation Tests .....                                        | 213 |
| 8.34 | Who-Is Service Initiation Tests .....                                        | 214 |
| 8.35 | I-Am Service Initiation Tests .....                                          | 214 |
| 8.36 | VT-Open Service Initiation Tests .....                                       | 214 |
| 8.37 | VT-Close Service Initiation Tests .....                                      | 216 |
| 8.38 | VT-Data Service Initiation Tests .....                                       | 217 |
| 8.39 | RequestKey Service Initiation Tests .....                                    | 219 |
| 8.40 | Authenticate Service Initiation Tests .....                                  | 219 |
| 9    | APPLICATION SERVICE EXECUTION TESTS .....                                    | 223 |
| 9.1  | AcknowledgeAlarm Service Execution Tests .....                               | 224 |
| 9.2  | ConfirmedCOVNotification Service Execution Tests .....                       | 245 |
| 9.3  | UnconfirmedCOVNotification Service Execution Tests .....                     | 248 |
| 9.4  | ConfirmedEventNotification Service Execution Tests .....                     | 249 |
| 9.5  | UnconfirmedEventNotification Service Execution Tests .....                   | 251 |
| 9.6  | GetAlarmSummary Service Execution Tests .....                                | 252 |
| 9.7  | GetEnrollmentSummary Service Execution Tests .....                           | 252 |
| 9.8  | GetEventInformation Service Execution Tests .....                            | 257 |
| 9.9  | LifeSafetyOperation Service Execution Test .....                             | 259 |
| 9.10 | SubscribeCOV Service Execution Tests .....                                   | 260 |
| 9.11 | SubscribeCOVProperty Service Execution Tests .....                           | 267 |
| 9.12 | AtomicReadFile Service Execution Tests .....                                 | 274 |
| 9.13 | AtomicWriteFile Service Execution Tests .....                                | 281 |
| 9.14 | AddListElement Service Execution Tests .....                                 | 293 |
| 9.15 | RemoveListElement Service Execution Tests .....                              | 295 |
| 9.16 | CreateObject Service Execution Tests .....                                   | 297 |
| 9.17 | DeleteObject Service Execution Tests .....                                   | 301 |
| 9.18 | ReadProperty Service Execution Tests .....                                   | 303 |
| 9.19 | ReadPropertyConditional Service Execution Tests .....                        | 305 |
| 9.20 | ReadPropertyMultiple Service Execution Tests .....                           | 306 |
| 9.21 | ReadRange Service Execution Tests .....                                      | 313 |
| 9.22 | WriteProperty Service Execution Tests .....                                  | 322 |
| 9.23 | WritePropertyMultiple Service Execution Tests .....                          | 327 |
| 9.24 | DeviceCommunicationControl Service Execution Test .....                      | 336 |
| 9.25 | ConfirmedPrivateTransfer Service Execution Tests .....                       | 342 |
| 9.26 | UnconfirmedPrivateTransfer Service Execution Tests .....                     | 343 |
| 9.27 | ReinitializeDevice Service Execution Tests .....                             | 343 |
| 9.28 | ConfirmedTextMessage Service Execution Tests .....                           | 346 |
| 9.29 | UnconfirmedTextMessage Service Execution Tests .....                         | 348 |
| 9.30 | TimeSynchronization Service Execution Tests .....                            | 348 |
| 9.31 | UTCTimeSynchronization Service Execution Tests .....                         | 349 |
| 9.32 | Who-Has Service Execution Tests .....                                        | 350 |
| 9.33 | Who-Is Service Execution Tests .....                                         | 355 |
| 9.34 | VT-Open Service Execution Tests .....                                        | 358 |
| 9.35 | VT-Close Service Execution Tests .....                                       | 359 |
| 9.36 | VT-Data Service Execution Tests .....                                        | 361 |
| 9.37 | RequestKey Service Execution Test .....                                      | 361 |
| 9.38 | Authenticate Service Execution Tests .....                                   | 363 |
| 9.39 | General Testing of Service Execution .....                                   | 367 |
| 10   | NETWORK LAYER PROTOCOL TESTS .....                                           | 368 |
| 10.1 | Processing Application Layer Messages Originating from Remote Networks ..... | 368 |

|      |                                                                              |     |
|------|------------------------------------------------------------------------------|-----|
| 10.2 | Router Functionality Tests.....                                              | 369 |
| 10.3 | Half-Router Functionality Tests .....                                        | 393 |
| 10.4 | B/IP PAD Tests .....                                                         | 400 |
| 10.5 | Initiating Network Layer Messages.....                                       | 403 |
| 10.6 | Non-Router Functionality Tests.....                                          | 404 |
| 10.7 | Route Binding Tests .....                                                    | 406 |
| 11   | LOGICAL LINK LAYER PROTOCOL TESTS .....                                      | 411 |
| 11.1 | UI Command and Response .....                                                | 411 |
| 11.2 | XID Command and Response.....                                                | 412 |
| 11.3 | TEST Command and Response .....                                              | 412 |
| 11.4 | MS/TP State Machine Tests .....                                              | 413 |
| 11.5 | PTP State Machine Tests .....                                                | 471 |
| 12   | SPECIAL FUNCTIONALITY TESTS .....                                            | 509 |
| 12.1 | Segmentation .....                                                           | 509 |
| 12.2 | Time Master.....                                                             | 518 |
| 12.3 | Character Sets.....                                                          | 519 |
| 12.4 | Malformed PDUs.....                                                          | 519 |
| 12.5 | Slave Proxy Tests .....                                                      | 520 |
| 12.6 | Automatic Network Mapping .....                                              | 523 |
| 12.7 | Automatic Device Mapping.....                                                | 523 |
| 13   | BACnet/IP Functionality Tests.....                                           | 524 |
| 13.1 | Non-BBMD B/IP Device.....                                                    | 524 |
| 13.2 | BBMD B/IP Device with a Server Application.....                              | 526 |
| 13.3 | Broadcast Distribution Table Operations.....                                 | 530 |
| 13.4 | Foreign Device Table Operations (Negative Tests).....                        | 534 |
| 13.5 | BACnet Broadcast Management (No Foreign Device Table, No Applications) ..... | 535 |
| 13.6 | Foreign Device Management.....                                               | 537 |
| 13.7 | Broadcast Management (BBMD, Foreign Devices, Local Application) .....        | 541 |
| 13.8 | Registering as a Foreign Device .....                                        | 549 |
| 13.9 | Initiating BVLL Service Requests Conveying an NPDU .....                     | 550 |
| 14   | Reporting Test Results.....                                                  | 551 |
|      | ANNEX A – EXAMPLE EPICS (INFORMATIVE).....                                   | 552 |

## Foreword

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

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

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

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

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: [Foreword - Supplementary information](#)

The committee responsible for this document is ISO/TC 205, *Building environment design*.

This third edition cancels and replaces the second edition (ISO 16484-6:2009), of which it forms the subject of a minor revision.

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

- *Part 1: Project specification and implementation*
- *Part 2: Hardware*
- *Part 3: Functions*
- *Part 5: Data communication protocol*
- *Part 6: Data communication conformance testing*

Applications and project implementation are to form the subjects of future Parts 4 and 7.

blank page

# 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/ASHRAE 135.1-2011

This part of ISO 16484 comprises, from Clause 4 onwards, the US standard ANSI/ASHRAE 135.1-2011, *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 those BIBBs described in Annex K. The format in the EPICS shall be to use the short acronym described in the title for each BIBB. For example: A device which supports 'Data Sharing - ReadProperty - B' should include 'DS-RP-B' in this section.

For example:

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

```
DM-DDB-A↵
}↵
```

#### 4.5.3 Application Services Supported

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

```
BACnet Standard Application Services Supported: ↵
{↵
  ☐ Initiate↵
  ☐ Execute↵
  ☐ Initiate Execute↵
}
```

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

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

For example:

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

#### 4.5.4 Object Types Supported

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

```
Standard Object Types Supported: ↵
{↵
  ☐↵
  ☐ Createable↵
  ☐ Deleteable↵
  ☐ Createable Deleteable↵
}↵
```

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

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

For example:

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

#### 4.5.5 Data Link Layer Options

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

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

#### 4.5.6 Character Sets

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

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

#### 4.5.7 Special Functionality

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

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

#### 4.5.8 Property Value Restrictions

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

```
Default Property Value Restrictions: ↵
{↵
  unsigned-integer: <minimum: □; maximum: □>↵
  signed-integer: <minimum: □; maximum: □>↵
  real: <minimum: □; maximum: □; resolution: □>↵
  double: <minimum: □; maximum: □; resolution: □>↵
  date: <minimum: □; maximum: □>↵
  octet-string: <maximum length string: □>↵
  character-string: <maximum length string: □>↵
  list: <maximum length list: □>↵
  variable-length-array: <maximum length array: □>↵
}↵
```

#### 4.5.9 Timers

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

```
Fail Times: ↵
{↵
  Notification Fail Time: □↵
  Internal Processing Fail Time: □↵
  Minimum ON/OFF Time: □↵
  Schedule Evaluation Fail Time: □↵
  External Command Fail Time: □↵
  Program Object State Change Fail Time: □↵
  Acknowledgement Fail Time: □↵
  Slave Proxy Confirm Interval: □↵
  Unconfirmed Response Fail Time: □↵
}↵
```

#### 4.5.10 Test Database

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

```
List of Objects in Test Device: ↵
{↵
```

```
object1↵  
object2↵  
...  
objectN↵  
}↵
```

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

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

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

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

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

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

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

```
{  
  object-identifier: (analog-input, 6)  
  object-name: "□"  
  object-type: analog-input  
  present-value: 12.3 C      -- 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: {□,□,□}
  acknowledged-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: {□,□,□}  
  acknowledged-transitions: {□,□,□}  
  notify-type: □  
  event-time-stamps: {□,□,□}  
  silenced: □  
  operation-expected: □  
  maintenance-required: □  
  setting: □  
  direct-reading□  
  units: □  
  member-of: (□,□...)  
  profile-name: "□"  
}
```

#### 4.5.10.16 Life Safety Zone

```
{  
  object-identifier: (life-safety-zone, □)
```

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

#### 4.5.10.17 Loop

```
{  
  object-identifier: (loop, □)  
  object-name: "□"  
  object-type: loop  
  present-value: ?  
  description: "□"  
  status-flags: {□,□,□,□}  
  event-state: □  
  reliability: □  
  out-of-service: □  
  update-interval: □  
  output-units: □  
  manipulated-variable-reference: {□}  
  controlled-variable-reference: {□}  
  controlled-variable-value □  
  controlled-variable-units □  
  setpoint-reference: {□}  
  setpoint: □  
  action: □  
  proportional-constant: □  
  proportional-constant-units: □  
  integral-constant: □  
  integral-constant-units: □  
  derivative-constant: □  
  derivative-constant-units: □  
  bias: □  
}
```

```

maximum-output: □
minimum-output: □
priority-for-writing: □
cov-increment: □
time-delay: □
notification-class: □
error-limit: □
event-enable: {□,□,□}
acked-transitions: {□,□,□}
notify-type: □
event-time-stamps: {□,□,□}
profile-name: "□"
}

```

#### 4.5.10.18 Multi-state Input

```

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

```

#### 4.5.10.19 Multi-state Output

```

{
  object-identifier: (multi-state-output, □)
  object-name: "□"
  object-type: multi-state-output
  present-value: ?
  description: "□"
  device-type: "□"
  status-flags: {□,□,□,□}
  event-state: □
  reliability: □
  out-of-service: □
  number-of-states: □
  state-text: {"□", "□"...}
  priority-array: {□,□,?,?,□,?,?,□,?,?,?,?,?,?}
  relinquish-default: □
}

```

```
time-delay: □  
notification-class: □  
feedback-value: □  
event-enable: {□,□,□}  
acked-transitions: {□,□,□}  
notify-type: □  
event-time-stamps: {□,□,□}  
profile-name: "□"  
}
```

#### 4.5.10.20 Multi-state Value

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

#### 4.5.10.21 Notification Class

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

#### 4.5.10.22 Program

```
{  
  object-identifier: (program, □)
```

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

#### 4.5.10.23 Pulse Converter

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

#### 4.5.10.24 Schedule

```
{  
  object-identifier: (schedule, ☐)  
  object-name: ""  
  object-type: schedule  
  present-value: ☐  
}
```

```
description: "□"  
effective-period: {□,□}  
weekly-schedule: {□,□...}  
exception-schedule: {□,□...}  
list-of-object-property-references: (□,□...)  
priority-for-writing □  
profile-name: "□"  
}
```

#### 4.5.10.25 Trend Log

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

#### 4.5.10.26 Access Door

```
{  
  object-identifier: (access-door, □)  
  object-name: "□"  
  object-type: access-door  
  present-value : □  
  description: "□"  
  status-flags: □  
  event-state: □  
  reliability: □  
  out-of-service: □  
  priority-array: □  
  relinquish-default: □  
  door-status: □  
  lock-status: □  
}
```

```
secured-status: ☐  
door-members: {☐, ☐...}  
door-pulse-time: ☐  
door-extended-pulse-time: ☐  
door-unlock-delay-time: ☐  
door-open-too-long-time: ☐  
door-alarm-state: ☐  
masked-alarm-values: {☐, ☐ ...}  
maintenance-required: ☐  
time-delay: ☐  
notification-class: ☐  
alarm-values: {☐, ☐...}  
fault-values: {☐, ☐...}  
event-enable: {☐, ☐, ☐}  
acked-transitions: {☐, ☐, ☐}  
notify-type: ☐  
event-time-stamps: {☐, ☐, ☐}  
profile-name: "☐"  
}
```

#### 4.5.10.27 Load Control

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

#### 4.5.10.28 Structured View

```
{  
  object-identifier: (structured-view, ☐)  
  object-name: "☐"  
}
```

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

## 5 EPICS CONSISTENCY TESTS

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

- (a) All object types required by the specified BIBBs shall be indicated as supported in the Standard Object Types Supported section of the EPICS.
- (b) A minimum of one instance of each object type required by the specified BIBBs shall be included in the test database.
- (c) The Object\_Types\_Supported property of the Device object in the test database shall indicate support for each object type required by the supported BIBBs.
- (d) All application services required by the supported BIBBs shall be indicated as supported in the BACnet Standard Application Services Supported section of the EPICS with Initiate and Execute indicated as required by the supported BIBBs.
- (e) The Application\_Services\_Supported property of the Device object in the test database shall indicate support for each application service for which the supported BIBBs requires support for execution of the service.
- (f) The object types listed in the Standard Object Types Supported section of the EPICS shall have a one-to-one correspondence with object types listed in the Object\_Types\_Supported property of the Device object contained in the test database.
- (g) For each object type listed in the Standard Object Types Supported section of the EPICS there shall be at least one object of that type in the test database.
- (h) There shall be a one-to-one correspondence between the objects listed in the Object\_List property of the Device object and the objects included in the test database. The Object\_List property and the test database shall both include all proprietary objects. Properties of proprietary objects that are not required by BACnet Clause 23.4.3 need not be included in the test database.
- (i) For each object included in the test database, all required properties for that object as defined in Clause 12 of BACnet shall be present. In addition, if any of the properties supported for an object require the conditional presence of other properties, their presence shall be verified.
- (j) For each property that is required to be writable, that property shall be marked as writable in the EPICS.
- (k) The length of the Protocol\_Services\_Supported bitstring shall have the number of bits defined for BACnetProtocolServicesSupported for the IUT's declared protocol revision.
- (l) The length of the Protocol\_Object\_Types\_Supported bitstring shall have the number of bits defined for BACnetObjectTypesSupported for the IUT's declared protocol revision.

## 6 CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

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

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

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

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

### 6.1 TCSL Components

#### 6.1.1 Common Symbols and Characters

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

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

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

#### 6.1.2 Integers

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

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

#### 6.1.3 Text Strings

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

Examples: Object\_Name = "CW\_STEMP", Description = "AC1 Supply Temperature".

#### 6.1.4 Enumerations

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

Examples: TRUE, FALSE, RELIABLE, UNRELIABLE.

### 6.1.5 Property Identifiers

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

Examples: Present\_Value, Reliability, Object\_Identifier, Object\_Type, and Vendor\_Name.

### 6.1.6 Service Parameters

The names of service parameters are enclosed in single quotes.

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

### 6.1.7 Object Identifiers

The representation of object identifiers is defined as follows:

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

Where:

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

<instance number> ::= <unsigned>

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

## 6.2 TCSL Statements

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

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

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

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

<simple statement> ::= <if statement> | <repeat statement> | <error statement>

| <check statement> | <make statement>

| <transmit statement> | <receive statement>

| <write statement> | <verify statement> | <before statement>

| <read statement>

### 6.2.1 IF Statement

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

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

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

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

### 6.2.2 REPEAT Statement

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

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

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

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

### 6.2.3 ERROR Statement

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

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

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

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

### 6.2.4 CHECK Statement

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

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

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

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

### 6.2.5 MAKE Statement

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

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

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

```
MAKE (Out_Of_Service TRUE)
```

### 6.2.6 TRANSMIT Statement

The TRANSMIT statement is used to transmit a packet.

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

Where:

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

```

<port> ::= PORT <port identifier>
<port identifier> ::= 'A' | 'B' ... 'Z'
<addressing> ::= ( <dst> | <src> | <dst> ',' <src> )
<src> ::= SOURCE '=' <src parm value>
<src parm 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 135-2008, or as otherwise specified.

Example: WAIT **Internal Processing Fail Time**

### 6.2.9 WRITE Statement

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

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

This is a shortcut of the following statements:

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

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

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

Example: WRITE (Analog Output, 1), Present\_Value = 6.5, PRIORITY = 8

#### 6.2.10 VERIFY Statement

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

The verify procedure consists of the following steps:

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

Example: WRITE (Analog Output, 1), Present\_Value = 6.5, PRIORITY = 8  
VERIFY (Analog Output, 1), Present\_Value = 6.5

#### 6.2.11 BEFORE Statement

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

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

The <timer value> shall be one of the timers specified in 6.3 of this standard, in ANSI/ASHRAE 135-2008, or as otherwise specified. If the action indicated by <statement> has not yet occurred when the timer expires the test fails. Otherwise this test step passes and the test continues. For example:

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  
}
```

#### 6.2.13 READ Statement

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

The read procedure consists of the following steps:

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

Example: READ X = Priority\_Array, ARRAY\_INDEX=10

### 6.3 Time Dependencies

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

#### 6.3.1 Notification Fail Time

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

#### 6.3.2 Internal Processing Fail Time

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

#### 6.3.3 Minimum ON/OFF Fail Time

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

#### 6.3.4 Schedule Evaluation Fail Time

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

#### 6.3.5 External Command Fail Time

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

#### 6.3.6 Program Object State Change Fail Time

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

#### 6.3.7 Acknowledgment Fail Time

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

### 6.3.8 Default Time Delay in Test Descriptions

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

### 6.3.9 Unconfirmed Response Fail Time

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

## 6.4 BACnet References

All references to BACnet clauses in this standard refer to ANSI/ASHRAE 135-2008, except as otherwise noted ("BACnet-2001" refers to ANSI/ASHRAE 135-2001).

## 6.5 TD Requirements

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

## 7 OBJECT SUPPORT TESTS

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

### 7.1 Read Support for Properties in the Test Database

#### 7.1.1 Read Support Test Procedure

Dependencies: ReadProperty Service Execution Tests, 9.18.

Purpose: 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, if supported. 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.

#### 7.1.2 Non-documented Property Test

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

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

Test Steps:

```

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

```

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

## 7.2 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  
        }  
    }  
}

### 7.2.3 Read-only Property Test

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

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

Test Steps:

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

```

        'Property Identifier' =      Y,
        List of Elements' =      (any element value meeting the range requirements of 7.2.1 excluding
                                those in Z)
        RECEIVE BACnet-Error-PDU,
            Error Class =          PROPERTY,
            Error Code =          WRITE_ACCESS_DENIED
    ELSE
        READ LEN = X, Array_Index = 0
        IF (LEN > 0) THEN
            READ Z = X, Array_Index = 1
            TRANSMIT WriteProperty-Request,
                'Object Identifier' =      X,
                'Property Identifier' =      Y,
                'Property Value' =          Z
                'Array Index' =            1
            RECEIVE BACnet-Error-PDU,
                Error Class =          PROPERTY,
                Error Code =          WRITE_ACCESS_DENIED

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

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

```

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

Notes to Tester: When modifying a property, it is expected that an error class of PROPERTY with an error code of WRITE\_ACCESS\_DENIED will be returned, but the IUT may instead return an error\_class of PROPERTY with an

error\_code of VALUE\_OUT\_OF\_RANGE, or an error\_class of RESOURCES with an error\_code of NO\_SPACE\_TO\_WRITE\_PROPERTY.

### 7.3 Object Functionality Tests

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

#### 7.3.1 Property Tests

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

##### 7.3.1.1 Out\_Of\_Service, Status\_Flags, and Reliability Tests

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

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

Purpose: This test case verifies that Present\_Value is writable when Out\_Of\_Service is TRUE. It also verifies the interrelationship between the Out\_Of\_Service, Status\_Flags, and Reliability properties. If the PICS indicates that the Out\_Of\_Service property of the object under test is not writable, and if the value of the property cannot be changed by other means, then this test shall be omitted. This test applies to Accumulator, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, Life Safety Point, Life Safety Zone, Multi-state Input, Multi-state Output, Multi-state Value, Loop and Pulse Converter objects.

Test Concept: The IUT will select one instance of each appropriate object type and test it as described. If the Reliability property is not supported then step 4 shall be omitted.

Test Steps:

1. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = TRUE  
ELSE  
    MAKE (Out\_Of\_Service TRUE)
2. VERIFY Out\_Of\_Service = TRUE
3. VERIFY Status\_Flags = (?, FALSE, ?, TRUE)
4. REPEAT X = (all values meeting the functional range requirements of 7.2.1) DO {  
    WRITE Present\_Value = X  
    VERIFY Present\_Value = X  
}
5. IF (Reliability is present and writable) THEN  
    REPEAT X = (all values of the Reliability enumeration appropriate to the object type except  
        NO\_FAULT\_DETECTED) DO {  
        WRITE Reliability = X  
        VERIFY Reliability = X  
        VERIFY Status\_Flags = (?, TRUE, ?, TRUE)  
        WRITE Reliability = NO\_FAULT\_DETECTED  
        VERIFY Reliability = NO\_FAULT\_DETECTED  
        VERIFY Status\_Flags = (?, FALSE, ?, TRUE)  
    }
6. IF (Out\_Of\_Service is writable) THEN  
    WRITE Out\_Of\_Service = FALSE

ELSE

MAKE (Out\_Of\_Service FALSE)

7. VERIFY Out\_Of\_Service = FALSE

8. VERIFY Status\_Flags = (?, ?, ?, FALSE)

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

### 7.3.1.2 Relinquish Default Test

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

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

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

Test Concept: A pre-requisite to this test is that an object has been provided for which all prioritized commands have been relinquished and any minimum on/off time has been accounted for. The Present\_Value is compared to the value of Relinquish\_Default to ensure that they are the same. If possible, the value of Relinquish\_Default is changed to verify that Present\_Value tracks the changes.

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

Test Steps:

1. VERIFY Priority\_Array = (NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL)
2. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Present\_Value
3. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Present\_Value  
'Property Value' = (any valid value, X)
4. VERIFY Relinquish\_Default = X
5. IF (Relinquish\_Default is writable) THEN  
WRITE Relinquish\_Default = (any valid value, Y, other than the one returned in step 3)  
VERIFY Present\_Value = Y

### 7.3.1.3 Command Prioritization Test

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

BACnet Reference Clause: 19.2.

Purpose: To verify that the command prioritization algorithm is properly implemented. This test applies to 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: 12.7.19, 12.7.20, 12.8.17, 12.8.18, 19.2.

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

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

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

Test Steps:

1. WRITE Present\_Value = ACTIVE, PRIORITY = 7
2. VERIFY Present\_Value = ACTIVE
3. VERIFY Priority\_Array = ACTIVE, ARRAY\_INDEX = 6
4. BEFORE Minimum\_On\_Time  
WRITE Present\_Value = INACTIVE, PRIORITY = (any value numerically lower than 6 (higher priority))
5. VERIFY Present\_Value = INACTIVE
6. VERIFY Priority\_Array = INACTIVE, PRIORITY = 6

Notes to Tester: If minimum on time is not supported but minimum off time is supported, this test should be conducted by using INACTIVE in steps 1 through 3 and ACTIVE in steps 4 through 6, and by using Minimum\_Off\_Time in step 4.

### **7.3.1.7 COV Tests**

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

### **7.3.1.8 Binary Object Change of State Tests**

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

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

Purpose: To verify that the properties of binary objects that collectively track state changes (changes in Present\_Value) function as required. If the Change\_Of\_State\_Count, Change\_Of\_State\_Time, and Time\_Of\_State\_Count\_Reset properties are not supported this test shall be omitted. This test applies to Binary Input, Binary Output, and Binary Value objects.

Test Concept: The Present\_Value of the binary object under test is changed. The Change\_Of\_State\_Count property is checked to verify that it has been incremented and the Change\_Of\_State\_Time property is checked to verify that it has been updated. The Change\_Of\_State\_Count is reset and Time\_Of\_State\_Count\_Reset is checked to verify that it has been updated appropriately.

Configuration Requirements: The object being tested shall be configured such that the Present\_Value and Change\_Of\_State\_Count properties are writable or another means of changing these properties shall be provided.

Test Steps:

1. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Present\_Value
2. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Present\_Value,  
'Property Value' = ACTIVE | INACTIVE
3. TRANSMIT ReadProperty-Request,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Change\_Of\_State\_Count
4. RECEIVE ReadProperty-ACK,  
'Object Identifier' = (the object being tested),  
'Property Identifier' = Change\_Of\_State\_Count,  
'Property Value' = (any valid value, N)
5. IF (Present\_Value is writable) THEN  
IF (the value returned in step 2 was ACTIVE) THEN

```

        WRITE Present_Value = INACTIVE
        VERIFY Present_Value = INACTIVE
    ELSE
        WRITE Present_Value = ACTIVE
        VERIFY Present_Value = ACTIVE
    ELSE
        MAKE (Present_Value change to the opposite state)
6.  TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Date
7.  RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Date,
    'Property Value' = (the current local date, D)
8.  TRANSMIT ReadProperty-Request,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Time
9.  RECEIVE ReadProperty-ACK,
    'Object Identifier' = (the IUT's Device object),
    'Property Identifier' = Local_Time,
    'Property Value' = (the current local time, TLOC)
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 TLOC)
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, TLOC)
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 TLOC)

```

### 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