



BSI Standards Publication

Building automation and control systems (BACS)

Part 6: Data communication conformance testing

National foreword

This British Standard is the UK implementation of EN ISO 16484-6:2025. It is identical to ISO 16484-6:2024. It supersedes BS EN ISO 16484-6:2020, which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee RHE/2, Ventilation for buildings, heating and hot water services.

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

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 539 33151 6
ICS 35.240.67; 91.040.01; 97.120

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2025.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English Version

**Building automation and control systems (BACS) - Part 6:
Data communication conformance testing (ISO 16484-
6:2024)**

Systèmes d'automatisation et de gestion technique du
bâtiment (BACS) - Partie 6: Essais de conformité de la
communication de données (ISO 16484-6:2024)

Systeme der Gebäudeautomation - Teil 6:
Datenübertragungsprotokoll - Konformitätsprüfung
(ISO 16484-6:2024)

This European Standard was approved by CEN on 29 December 2024.

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

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

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



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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

European foreword

This document (EN ISO 16484-6:2025) has been prepared by Technical Committee ISO/TC 205 "Building environment design" in collaboration with Technical Committee CEN/TC 247 "Building Automation, Controls and Building Management" the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by May 2026, and conflicting national standards shall be withdrawn at the latest by May 2026.

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

This document supersedes EN ISO 16484-6:2020.

Any feedback and questions on this document should be directed to the users' national standards body/national committee. A complete listing of these bodies can be found on the CEN website.

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

Endorsement notice

The text of ISO 16484-6:2024 has been approved by CEN as EN ISO 16484-6:2025 without any modification.

CONTENTS

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

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

ISO 16484-6:2024(en)

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

FOREWORD

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

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

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

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

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

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

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

The main changes are as follows:

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

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

Building automation and control systems (BACS) —

Part 6:

Data communication conformance testing

1. PURPOSE

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

2. SCOPE

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

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

3. DEFINITIONS

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

3.1 Terms Adopted from International Standards

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

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

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

3.2 Abbreviations and Acronyms Used in the Standard

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

3.3 Common language used in tests

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

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

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

4. ELECTRONIC PICS FILE FORMAT

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

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

4.1 Character Encoding

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

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

Table 4-1. Character Set Codes

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

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

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

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

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

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

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

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

4.2 Structure of EPICS Files

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

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

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

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

BACnet Protocol Implementation Conformance Statement ↵

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

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

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

End of BACnet Protocol Implementation Conformance Statement ↵

4.3 Character Strings

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

4.4 Notational Rules for Parameter Values

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

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

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

4.4.1 Complex Parameter Values

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

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

4.4.2 Specifying Limits on Parameter Values

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

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

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

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

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

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

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

4.5 Sections of the EPICS File

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

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

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

An example EPICS file may be found in Annex A.

4.5.1 General Information Sections

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

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

4.5.2 Conformance Sections

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

4.5.2.1 BIBBs Supported

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

```
BIBBs Supported: ↵
{↵
  □↵
}↵
```

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

For example:

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

4.5.3 Application Services Supported

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

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

```

❑ Initiate↵
❑ Execute↵
❑ Initiate Execute↵
}

```

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

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

For example:

```

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

```

4.5.4 Object Types Supported

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

```

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

```

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

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

For example:

```

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

```

4.5.5 Data Link Layer Options

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

```

Data Link Layer Option: ↵
{↵

```

```

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

```

4.5.6 Character Sets

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

```

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

```

4.5.7 Special Functionality

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

```

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

```

4.5.8 Property Value Restrictions

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

```

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

```

```

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

```

4.5.9 Timers

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

Fail Times: ↵

```

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

```

4.5.10 Test Database

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

List of Objects in Test Device: ↵

```

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

```

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

```

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

```

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

```

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

```

```

    object-type: proprietary □
  }

```

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

```

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

```

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

```

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

```

5. EPICS CONSISTENCY TESTS

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

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

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

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

6. CONVENTIONS FOR SPECIFYING BACnet CONFORMANCE TESTS

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

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

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

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

6.1 TCSL Components

6.1.1 Common Symbols and Characters

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

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

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

6.1.2 Integers

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

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

6.1.3 Text Strings

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

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

6.1.4 Enumerations

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

Examples: TRUE, FALSE, RELIABLE, UNRELIABLE.

6.1.5 Property Identifiers

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

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

6.1.6 Service Parameters

The names of service parameters are enclosed in single quotes.

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

6.1.7 Object Identifiers

The representation of object identifiers is defined as follows:

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

Where:

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

$$\langle \text{instance number} \rangle ::= \langle \text{unsigned} \rangle$$

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

6.2 TCSL Statements

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

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

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

$$\begin{aligned} \langle \text{simple statement} \rangle ::= & \langle \text{if statement} \rangle \mid \langle \text{repeat statement} \rangle \mid \langle \text{error statement} \rangle \\ & \mid \langle \text{check statement} \rangle \mid \langle \text{make statement} \rangle \\ & \mid \langle \text{transmit statement} \rangle \mid \langle \text{receive statement} \rangle \\ & \mid \langle \text{write statement} \rangle \mid \langle \text{verify statement} \rangle \mid \langle \text{before statement} \rangle \\ & \mid \langle \text{read statement} \rangle \end{aligned}$$

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.

$$\langle \text{if statement} \rangle ::= \text{IF '}' \langle \text{condition} \rangle \text{'}' THEN } \langle \text{statement} \rangle \text{ [ELSE } \langle \text{statement} \rangle \text{]}$$

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

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

6.2.2 REPEAT Statement

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

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

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

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

6.2.3 ERROR Statement

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

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

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

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

6.2.4 CHECK Statement

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

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

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

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

6.2.5 MAKE Statement

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

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

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

```
MAKE (Out_Of_Service TRUE)
```

6.2.6 TRANSMIT Statement

The TRANSMIT statement is used to transmit a packet.

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

Where:

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

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

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

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

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

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

Example 1:

TRANSMIT DESTINATION = GLOBAL BROADCAST, Who-Is

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

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

Example 2:

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

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

6.2.7 RECEIVE Statement

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

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

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

Example: TRANSMIT SubscribeCOV-Request,

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

RECEIVE ConfirmedCOVNotification-Request,

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

6.2.8 WAIT Statement

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

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

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

Example: WAIT **Internal Processing Fail Time**

6.2.9 WRITE Statement

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

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

This is a shortcut of the following statements:

1. TRANSMIT WriteProperty-Request,

'Object Identifier' = <object identifier>,
 'Property Identifier' = <property identifier>,
 'Property Value' = <property value>,
 'Priority Array Index' = <array index>,
 'Priority' = <write priority>

2. RECEIVE BACnet-SimpleACK-PDU

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

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

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

6.2.10 VERIFY Statement

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

The verify procedure consists of the following steps:

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

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

6.2.11 BEFORE Statement

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

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

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

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

6.2.12 WHILE Statement

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

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

Example:

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

6.2.13 READ Statement

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

The read procedure consists of the following steps:

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

Example: READ X = Priority_Array, ARRAY INDEX=10

6.2.14 Assignment Statement

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

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

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

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

6.3 Time Dependencies

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

6.3.1 Notification Fail Time

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

6.3.2 Internal Processing Fail Time

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

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

6.3.3 Minimum ON/OFF Fail Time

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

6.3.4 Schedule Evaluation Fail Time

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

6.3.5 External Command Fail Time

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

6.3.6 Program Object State Change Fail Time

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

6.3.7 Acknowledgment Fail Time

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

6.3.8 Default Time Delay in Test Descriptions

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

6.3.9 Unconfirmed Response Fail Time

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

6.3.10 Channel Write Fail Time

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

6.3.11 Auto-negotiation Fail Time

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

6.3.12 Activate Changes Fail Time

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

6.3.13 Foreign Device Registration Fail Time

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

6.3.14 Audit Notification Forwarder Fail Time

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

6.4 BACnet References

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

6.5 TD Requirements

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

6.6 Test Execution Considerations

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

6.6.1 Value Comparisons

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

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

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

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

6.6.2 Functional Expectations

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

6.6.3 Complex Datatypes

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

7. OBJECT SUPPORT TESTS

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

7.1 Read Support for Properties in the Test Database

7.1.1 Read Support Test Procedure

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

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

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

Test Steps:

```

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

```

```

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

```

7.1.2 Non-documented Property Test

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

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

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

Test Steps:

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

7.1.3 Verifying Property_List against the EPICS

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

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

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

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

Test Steps:

1. READ OL = Object_List

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

7.2 Write Support for Properties in the Test Database

7.2.1 Functional Range Requirements for Property Values

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

7.2.1.1 Enumerated and Boolean Values

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

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

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

7.2.1.3 Octetstrings and Characterstrings,

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

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

7.2.1.4 Bitstring

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

7.2.1.5 Date

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

7.2.1.6 Time

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

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

7.2.1.7 Constructed Datatypes

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

7.2.2 Write Support Test Procedure

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

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

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

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

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

Test Steps:

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

7.2.3 Read-only Property Test

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

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

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

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

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

Test Steps:

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

```

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

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

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

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

```

7.2.4 Date Pattern Properties Test

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

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

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

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

Test Steps:

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

7.2.5 Time Pattern Properties Test

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

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

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

Test Steps:

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

7.2.6 DateTime Pattern Properties Test

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

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

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

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

Test Steps:

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

7.2.7 Date Non-Pattern Properties Test

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

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

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

Test Steps:

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

7.2.8 Time Non-Pattern Properties Test

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

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

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

Test Steps:

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

```

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

```

7.2.9 DateTime Non-Pattern Properties Test

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

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

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

Test Steps:

```

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months,
             even months, last day of month, even days, odd days, hour unspecified, minute unspecified, second unspecified,
             hundredths unspecified) DO {
1.  TRANSMIT WriteProperty-Request
    'Object Identifier' = O1,
    'Property Identifier' = P1,
    'Property Value' = (DT1 updated with the special value SV)
2.  RECEIVE BACnet-Error-PDU
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE
}

```

7.2.10 BACnetDateRange Non-Pattern Properties Test

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

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

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

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

Test Steps:

```

REPEAT SV = (year unspecified, month unspecified, day of month unspecified, day of week unspecified, odd months,
             even months, last day of month, even days, odd days) DO {
1.  TRANSMIT WriteProperty-Request
    'Object Identifier' = O1,
    'Property Identifier' = P1,
    'Property Value' = (DR1 with startDate updated with the special value SV)
2.  RECEIVE BACnet-Error-PDU
    'Error Class' = PROPERTY,
    'Error Code' = VALUE_OUT_OF_RANGE
3.  TRANSMIT WriteProperty-Request

```

```

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

```

7.2.11 BACnetDateRange Open-Ended Pattern Properties Test

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

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

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

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

Test Steps:

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

7.3 Object Functionality Tests

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

7.3.1 Property Tests

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

7.3.1.1 Out_Of_Service, Status_Flags, and Reliability Tests

7.3.1.1.1 Out_Of_Service, Status_Flags, and Reliability Test

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

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

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

Test Steps:

1. READ SF1 = Status_Flags

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

7.3.1.1.2 Out_Of_Service for Commandable Value Objects Test

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

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

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

Test Steps:

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

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

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

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

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

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

Test Steps:

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

7.3.1.2 Relinquish Default Test

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

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

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

Test Steps:

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

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

7.3.1.3 Command Prioritization Test

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

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

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

Test Steps:

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

7.3.1.4 Minimum_Off_Time

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

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

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

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

Test Steps:

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

7.3.1.5 Minimum_On_Time

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

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

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

Test Steps:

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

7.3.1.6 Minimum On/Off Time Tests

7.3.1.6.1 Override of Minimum Time

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

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

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

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

Test Steps:

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

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum_Off_Time

End of Test

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

Test Steps:

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

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

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

End of Test

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

Test Steps:

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

- (Steps 11-14: Check that an INACTIVE value at P9 does not affect ARRAY INDEX = 6 or PV)
 15. WAIT (Minimum_On_Time + **Internal Processing Fail Time**)
 16. VERIFY Present_Value = INACTIVE

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

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

Test Steps:

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

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

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

Test Steps:

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

7.3.1.6.6 Minimum_Off_Time - Clock is not affected by additional write operations

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum Off Time

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

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

Test Steps:

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

7.3.1.6.7 Minimum_On_Time - Clock is not affected by additional write operations

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum On Time

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

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

Test Steps:

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

7.3.1.6.8 Ensuring Minimum_Off_Time starts at transition to INACTIVE

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

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

T1

T2

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

Notes to Tester: P9 and P7 may be equal.

Test Steps:

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

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

7.3.1.6.9 Ensuring Minimum_On_Time starts at transition to ACTIVE

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

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

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

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

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

Note: Bold font indicates the change invoked by write operation

Minimum_On_Time

T1

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

Notes to Tester: P9 and P7 may be equal.

Test Steps:

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

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

7.3.1.6.10 Ensuring Minimum Times Are Not Affected By Time Changes

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

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

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

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

Test Steps:

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

7.3.1.6.11 Minimum_Off_Time - Value Source Mechanism

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

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

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

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

Test Steps:

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

7.3.1.6.12 Minimum_On_Time - Value Source Mechanism

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

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

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

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

Test Steps:

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

7.3.1.7 COV Tests

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

7.3.1.7.1 COV_Resubscription_Interval Test

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

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

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

Test Steps:

1. IF (the IUT uses SubscribeCOV) THEN