

BSR/ASHRAE Addendum *t*  
to ANSI/ASHRAE Standard 135-2004

# Public Review Draft

ASHRAE® Standard

## Proposed Addendum *t* to Standard 135-2004, *BACnet*®—A *Data Communication Protocol for Building Automation and Control Networks*

First Public Review (September 2008)  
(Draft Shows Proposed Changes  
to Current Standard)

This draft has been recommended for public review by the responsible project committee. To submit a comment on this proposed addendum, go to the ASHRAE website at <http://www.ashrae.org/technology/page/331> and access the online comment database. The draft is subject to modification until it is approved for publication by the responsible project committee, the ASHRAE Standards Committee, and the Board of Directors. Then it will be submitted to the American National Standards Institute Board of Standards Review (BSR) for approval. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE web site) remains in effect. The current edition of any standard may be purchased from the ASHRAE Bookstore @ <http://www.ashrae.org> or by calling 404-636-8400 or 1-800-527-4723 (for orders in the U.S. or Canada).

This standard is under continuous maintenance. To propose a change to the current standard, use the change submittal form available on the ASHRAE web site @ <http://www.ashrae.org>.

The appearance of any technical data or editorial material in this public review document does not constitute endorsement, warranty, or guaranty by ASHRAE of any product, service, process, procedure, or design, and ASHRAE expressly disclaims such.

© **September 12, 2008**. This draft is covered under ASHRAE copyright. Permission to reproduce or redistribute all or any part of this document must be obtained from the ASHRAE Manager of Standards, 1791 Tullie Circle, NE, Atlanta, GA 30329-2305. Phone: 404-636-8400, Ext1125. Fax: 404-321-5478. E-mail: [standards.section@ashrae.org](mailto:standards.section@ashrae.org).

AMERICAN SOCIETY OF HEATING,  
REFRIGERATING AND AIR-CONDITIONING  
ENGINEERS, INC.  
1791 Tullie Circle, NE · Atlanta GA 30329-2305



**[This foreword and the “rationale” on the following page are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]**

## FOREWORD

The purpose of this addendum is to present a proposed change for public review. These modifications are the result of change proposals made pursuant to the ASHRAE continuous maintenance procedures and of deliberations within Standing Standard Project Committee 135. The proposed changes are summarized below.

135-2004t-1. XML Data Formats, p. 1.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2004 and Addenda is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment as this time. All other material in this addendum is provided for context only and is not open for public review comment except as it relates to the proposed changes.

**135-2004*t*-1. XML Data Formats**

**Rationale**  
 A new standard way of representing building data will give BACnet new capabilities for standardized communications between a wide range of applications. A definition for an XML syntax which can be used to represent building data in a consistent, flexible and extensible manner is defined by this addenda in the form of a new annex to the standard.

The Extensible Markup Language (XML) is a popular technology in the data processing and communications worlds due to its ability to model a wide range of data and its ability to be transformed and extended. With this new IT-friendly way of representing building data, BACnet will open up a range of possible new ways to share data. XML can be used for exchanging files between systems, integrating buildings with energy utilities, and expanding enterprise integration with richer Web services. Some of these new applications will be standardized in future addenda to the standard based on the syntax defined here.

**Addendum 135-2004*t*-1**

[This Table of Contents is not part of the standard. It is provided as an aid to the reviewer]

ANNEX X - XML DATA FORMATS (NORMATIVE) ..... 1

- X.1 Introduction..... 1
  - X.1.1 Design ..... 1
  - X.1.2 Syntax Examples..... 2
- X.2 Document Structure ..... 4
  - X.2.1 <CSML> ..... 4
    - X.2.1.1 'defaultLocale' ..... 4
    - X.2.1.2 <Definitions> ..... 5
- X.3 Expressing BACnet Datatypes in XML..... 5
  - X.3.1 Common Attributes..... 5
    - X.3.1.1 'name' ..... 6
    - X.3.1.2 'type' ..... 6
    - X.3.1.3 'extends' ..... 6
    - X.3.1.4 'overlays' ..... 6
    - X.3.1.5 'displayName' ..... 7
    - X.3.1.6 'description' ..... 7
    - X.3.1.7 'comment' ..... 8
    - X.3.1.8 'writable' ..... 8
    - X.3.1.9 'readable' ..... 9
    - X.3.1.10 'commandable' ..... 9
    - X.3.1.11 'associatedWith' ..... 9
    - X.3.1.12 'requiredWith' ..... 10
    - X.3.1.13 'requiredWithout' ..... 11
    - X.3.1.14 'notPresentWith' ..... 12
    - X.3.1.15 'writableWhen' ..... 12
    - X.3.1.16 'requiredWhen' ..... 13
    - X.3.1.17 'writeEffective' ..... 14
    - X.3.1.18 'optional' ..... 14
    - X.3.1.19 'absent' ..... 15
    - X.3.1.20 'variability' ..... 15
    - X.3.1.21 'volatility' ..... 16
    - X.3.1.22 'contextTag' ..... 16
    - X.3.1.23 'propertyIdentifier' ..... 16
  - X.3.2 Common Child Elements ..... 17
    - X.3.2.1 <DisplayName> ..... 17
    - X.3.2.2 <Description> ..... 17
    - X.3.2.3 <Documentation> ..... 17
    - X.3.2.4 <WritableWhen> ..... 18

X.3.2.5	<RequiredWhen> .....	18
X.3.2.6	<Extensions> .....	19
X.3.3	Named Values .....	19
X.3.3.1	'displayNameForWriting' .....	21
X.3.3.2	'notForWriting' .....	21
X.3.3.3	'notForReading' .....	21
X.3.3.4	<DisplayNameForWriting> .....	22
X.3.4	Primitive Values .....	22
X.3.4.1	'value' .....	22
X.3.4.2	'unspecifiedValue' .....	22
X.3.4.3	'charset' .....	23
X.3.4.4	'codepage' .....	23
X.3.4.5	'length' .....	23
X.3.5	Range Restrictions .....	24
X.3.5.1	'minimum' .....	25
X.3.5.2	'maximum' .....	25
X.3.5.3	'minimumForWriting' .....	25
X.3.5.4	'maximumForWriting' .....	25
X.3.5.5	'resolution' .....	26
X.3.6	Engineering Units .....	26
X.3.6.1	'units' .....	26
X.3.6.2	<Units> .....	26
X.3.7	Data Validity .....	27
X.3.7.1	'valueAge' .....	27
X.3.7.2	'error' .....	27
X.3.8	Length Restrictions .....	28
X.3.8.1	'minimumLength' .....	29
X.3.8.2	'maximumLength' .....	29
X.3.8.3	'minimumLengthForWriting' .....	29
X.3.8.4	'maximumLengthForWriting' .....	29
X.3.8.5	'minimumEncodedLength' .....	29
X.3.8.6	'maximumEncodedLength' .....	30
X.3.8.7	'minimumEncodedLengthForWriting' .....	30
X.3.8.8	'maximumEncodedLengthForWriting' .....	30
X.3.9	Collections .....	30
X.3.9.1	'minimumSize' .....	30
X.3.9.2	'maximumSize' .....	31
X.3.9.3	'memberType' .....	31
X.3.9.4	<MemberTypeDefinition> .....	31
X.3.10	Representing Primitive Data .....	32
X.3.10.1	<Null> .....	32
X.3.10.2	<Boolean> .....	32
X.3.10.3	<Unsigned> .....	32
X.3.10.4	<Integer> .....	32
X.3.10.5	<Real> .....	32
X.3.10.6	<Double> .....	32
X.3.10.7	<OctetString> .....	33
X.3.10.8	<String> .....	33
X.3.10.9	<BitString> .....	33
X.3.10.10	<Enumerated> .....	33
X.3.10.11	<Date> .....	33
X.3.10.12	<DatePattern> .....	33
X.3.10.13	<Time> .....	34
X.3.10.14	<TimePattern> .....	34
X.3.10.15	<ObjectIdentifier> .....	34
X.3.10.16	<WeekNDay> .....	35
X.3.11	Representing Constructed Data .....	35

- X.3.11.1 <Sequence>..... 35
- X.3.11.2 <Choice>..... 36
- X.3.11.3 <SequenceOf> ..... 36
- X.3.11.4 <Array>..... 36
- X.3.11.5 <List>..... 37
- X.3.12 Representing Data of Unknown Type..... 37
  - X.3.12.1 <Any> ..... 37
  - X.3.12.2 'allowedTypes'..... 37
- X.4 Expressing BACnet Objects and Properties in XML..... 37
  - X.4.1 <Object> ..... 38
- X.5 Definitions, Types, Instances, and Inheritance ..... 38
- X.6 Encoding and Access Rules ..... 44
- X.7 Extensibility ..... 44
  - X.7.1.1 XML extensions..... 44
  - X.7.1.2 Data Model Extensions ..... 44

## **Changes to ASHRAE Standard 135:** (reference: 135-2004)

[Add new **Annex X**, as follows]

### **ANNEX X - XML DATA FORMATS (NORMATIVE)**

**(This annex is part of this standard and is required for its use.)**

This annex defines formats for XML data exchanged between various BAS systems. This data may have a variety of purposes and may be conveyed through files or by other means.

#### **X.1 Introduction**

The Extensible Markup Language (XML) is a format for structured text that can be used to represent a variety of data in a machine-readable form. The XML syntax used in this standard conforms to the "Extensible Markup Language (XML) 1.0 (Fourth Edition)" and the XML datatypes used in this standard, indicated by the prefix "xs:", refer to the datatypes defined by "XML Schema Part 2: Datatypes Second Edition."

This syntax allows data structure definitions, such as those in Clause 21, and instances of those definitions to be represented in XML. The syntax is optimized for efficient representation of BACnet data, but is sufficiently flexible to be not limited to modeling BACnet data exclusively. Additionally, the syntax allows for human language descriptions, range restrictions, and usage information to be added to the basic data structure definitions.

##### **X.1.1 Design**

This XML syntax is designed to provide a common syntax and data model that can be used to represent both standard and proprietary data types along with any accompanying descriptive information. It is a general data definition and instance language rather than a syntax specific for the data defined in this standard. To allow the flexibility to present proprietary data along with standard data with a consistent syntax and data model, the syntax uses a datatype-centric form, such as `<String name="present-value" value="75">` and `<Boolean name="proprietary-value" value="true">` rather than a syntax specific to standard BACnet data names, such as `<PresentValue>75</PresentValue>`. This allows any kind of data, standard or proprietary, to be represented in the future without changing the XML schema or the code for the low level parsing of the XML into a native form for higher level processing to consume.

For the purposes of document brevity and human readability, this syntax represents data in XML attribute form rather than element body text form wherever possible. However, this representation choice does not limit extensibility of the data model because, like the attributes in the BACnet Web services data model described in Annex N, most attributes defined in this annex can be extended to have attributes of their own using an extension syntax defined in Clause X.7. This allows XML brevity for the common cases without limiting extensibility when needed.

With the exception of the `<Documentation>` element, which contains rich text XHTML-formatted documentation, this standard does not use mixed content in any element body. So simple consumers that ignore formatted documentation only need to process attribute text and simple element body text.

Validation of this syntax may be accomplished with XML data validators such as XML Schema. These validators can be used to validate the type and range of data in elements and attributes of the syntax itself. This syntax is also intended to represent a higher-level data model, such as BACnet objects and properties. Higher level data model validation, such as whether a property is allowed in a given object or whether its value is within its declared minimum and maximum limits, is beyond the capabilities of XML syntax validators like XML Schema and shall therefore be performed as needed by the application consuming this XML.

To simplify processing and avoid the definition of potentially complex scoping rules, all datatype definitions are given globally unique names. The management of the names to ensure global uniqueness is a local matter to the organization producing the XML and should at least consist of a prefix for the name that is unique to an organization and then have

the organization manage everything that follows that prefix. For brevity, if an organization has a BACnet Vendor ID, the prefix can consist of that ID as a decimal number followed by a dash character ('-'). In all cases, however, a reversed domain name, like "com.companyname.controlsdivision.", can be used as a prefix to ensure global uniqueness.

### X.1.2 Syntax Examples

Some examples using the Clause 21 datatypes will provide an introduction to the form and capabilities of the syntax. The full details of the XML elements and attributes are defined in Clauses X.3 and X.4, and a description of the data model and the type system is described in Clause X.5.

Enumerations in Clause 21 are defined as a mapping between an unsigned value and a textual identifier.

```
BACnetFileAccessMethod ::= ENUMERATED {  
    record-access      (0),  
    stream-access      (1)  
}
```

The definition of that same enumeration in XML would create the mapping with a series of named unsigned values.

```
<Definitions>  
  <Enumerated name="0-BACnetFileAccessMethod">  
    <NamedValues>  
      <Unsigned name="record-access" value="0" />  
      <Unsigned name="stream-access" value="1" />  
    </NamedValues>  
  </Enumerated>  
</Definitions>
```

An XML representation of a value of that enumeration would use the textual identifier rather than the number when the type is known.

```
<Enumerated type="0-BACnetFileAccessMethod " value="record-access" />
```

Some enumerations in BACnet are extensible, however, and in those cases, and in cases where the type is not known, a number is used in place of the textual identifier. This is discussed in more detail later.

Bit Strings in Clause 21 are similarly defined as a mapping between a bit position and a textual identifier.

```
BACnetEventTransitionBits ::= BIT STRING {  
    to-offnormal (0),  
    to-fault     (1),  
    to-normal    (2)  
}
```

The definition of that bit string in XML would define also the mapping with a series of named unsigned values, where the value specifies the bit position.

```
<Definitions>  
  <BitString name="0-BACnetEventTransitionBits">  
    <NamedValues>  
      <Unsigned name="to-offnormal" value="0" />  
      <Unsigned name="to-fault"     value="1" />  
      <Unsigned name="to-normal"    value="2" />  
    </NamedValues>  
  </BitString>
```

</Definitions>

An XML representation of a value of that bit string would be a list of the textual identifiers for the bits that are set.

```
<BitString type="0-BACnetEventTransitionBits" value="to-offnormal;to-normal" />
```

Similar to the extensible enumeration case, if a textual representation of a bit is not known, then a number indicating its bit-position is used instead.

Constructed data definitions in Clause 21 define a set of named members and can also specify context tags, optionality, and comments about the data members.

```
BACnetPropertyReference ::= SEQUENCE {  
    propertyIdentifier    [0] BACnetPropertyIdentifier,  
    propertyArrayIndex   [1] Unsigned OPTIONAL --used only with array datatype  
                        -- if omitted with an array the entire array is referenced  
}
```

In XML, this sequence also specifies names, context tags, optionality, and can even capture comments:

```
<Definitions>  
  <Sequence name="0-BACnetPropertyReference">  
    <Enumerated name="propertyIdentifier" contextTag="0" type="0-BACnetPropertyIdentifier" />  
    <Unsigned name="propertyArrayIndex" contextTag="1" optional="true"  
      comment="Used only with array datatype. If omitted, the entire array is referenced." />  
  </Sequence>  
</Definitions>
```

An XML representation of a value of that sequence may provide values for each member that is present, using a representation appropriate to that member's type, and omit optional members that are not present.

```
<Sequence type="0-BACnetPropertyReference" >  
  <Enumerated name="propertyIdentifier" value="present-value" />  
</Sequence>
```

Choices in Clause 21 are defined as a choice of named members with specified types.

```
BACnetClientCOV ::= CHOICE {  
    real-increment    REAL,  
    default-increment NULL  
}
```

In XML, this choice is also defined as a choice of named members with specified types.

```
<Definitions>  
  <Choice name="0-BACnetClientCOV">  
    <Choices>  
      <Real name="real-increment" />  
      <Null name="default-increment" />  
    </Choices>  
  </Choice>  
</Definitions>
```

An XML representation of a value of that choice would indicate which member is chosen and give a value for it.

```
<Choice type="0-BACnetClientCOV">
```

```
<Real name="real-increment" value="75.0" />
</Choice>
```

Variable length collections of identical members are defined in Clause 21 using the SEQUENCE OF construct.

```
BACnetDailySchedule ::= SEQUENCE {
    day-schedule [0] SEQUENCE OF BACnetTimeValue
}
```

In XML, this collection would be represented by the SequenceOf element which takes a 'memberType' attribute.

```
<Definitions>
  <Sequence name="0-BACnetDailySchedule">
    <SequenceOf name="day-schedule" contextTag="0" memberType="0-BACnetTimeValue"/>
  </Sequence>
</Definitions>
```

An XML representation of a value of that SEQUENCE OF provides a collection of unnamed members of the appropriate type.

```
<Sequence type="0-BACnetDailySchedule ">
  <SequenceOf name="day-schedule">
    <Sequence>
      <Time name="time" value="08:00:00.00"/>
      <Unsigned name="value" value="1"/>
    </Sequence>
    <Sequence>
      <Time name="time" value="15:00:00.00"/>
      <Unsigned name="value" value="0"/>
    </Sequence>
  </SequenceOf>
</Sequence>
```

## X.2 Document Structure

The XML elements and attributes defined in this annex may be used for a variety of purposes. When used in a standalone XML document, there is a single top-level element, <CSML>.

### X.2.1 <CSML>

When used in an XML document, the XML syntax defined by this annex is enclosed in the single top-level element <CSML> that has and a single optional attribute, 'defaultLocale'.

The valid child elements of <CSML> are any number and combination of the <Definitions> element and the data elements <Any>, <Array>, <BitString>, <Boolean>, <Choice>, <Date>, <DatePattern>, <DateTime>, <Double>, <Enumerated>, <Integer>, <List>, <Null>, <Object>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>, all described elsewhere in this annex.

#### X.2.1.1 'defaultLocale'

This optional attribute of <CSML>, of type xs:language, specifies the locale (language plus optional country tag) that becomes the "default locale". All human language data in the document is for the default locale unless otherwise indicated.

If this attribute is not present, then the default locale for the document remains unspecified. In this case, the interpretation of any human language content is a local matter, and the use of the 'locale' attribute to specify alternate locales is not permitted elsewhere in the document.

### **X.2.1.2 <Definitions>**

This optional child element of <CSML> provides the "definition context" as used in this annex.

One of the fundamental functions of this XML syntax is to define new data structures. Child elements of <Definitions> provide named definitions that are globally available for use as type definitions for instances, or to be extended by other definitions.

The valid child elements of <CSML> are any number and combination of the data elements <Any>, <Array>, <BitString>, <Boolean>, <Choice>, <Date>, <DatePattern>, <DateTime>, <Double>, <Enumerated>, <Integer>, <List>, <Null>, <Object>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>, all described elsewhere in this annex.

For example, the following <Sequence> element creates a globally available definition for "0-BACnetAddress" by enclosing the <Sequence> element within the <Definitions> element.

```
<Definitions>
  <Sequence name="0-BACnetAddress">
    <Unsigned name="network-number"/>
    <OctetString name="mac-address"/>
  </Sequence>
</Definitions>
```

The following represents an instance of that <Sequence> that refers to its definition using the 'type' attribute.

```
<Sequence type="0-BACnetAddress">
  <Unsigned name="network-number" value="888" />
  <OctetString name="mac-address" value="AC101801BAC0" />
</Sequence>
```

## **X.3 Expressing BACnet Datatypes in XML**

BACnet data is expressed in XML using the data elements <Any>, <Array>, <BitString>, <Boolean>, <Choice>, <Date>, <DatePattern>, <DateTime>, <Double>, <Enumerated>, <Integer>, <List>, <Null>, <Object>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>.

While no two datatypes are the same, the various types do have many things in common. The characteristics that are common to all datatypes are described in Clauses X.3.1 and X.3.2; the characteristics that are common to groups of datatypes are described in Clauses X.3.3 through X.3.9; and finally, the characteristics of the individual datatypes are described in Clauses X.3.10 and X.3.11.

### **X.3.1 Common Attributes**

All BACnet data elements share a common set of optional attributes. In addition to the common attributes described here, each primitive data element may also define a specific set of required or optional attributes of its own. This is done in individual clauses that define those data elements.

### **X.3.1.1 'name'**

This optional attribute, of type `xs:string`, provides a name for the element. A name may or may not be required, based on the element's context. For example, a name is required for definitions, and for `<Sequence>`, `<Choice>`, and `<Object>` members, but not for `<Array>`, `<List>`, and `<SequenceOf>` members. When used in a definition context, the 'name' provides a globally unique name for the defined type which other elements may refer to by using the 'type', 'extends', or 'overlays' attributes.

For example, the 'name' attribute is used below to define the type name "0-BACnetDeviceObjectReference" and also to define the names of the two members.

```
<Definitions>
  <Sequence name="0-BACnetDeviceObjectReference">
    <ObjectIdentifier name="deviceIdentifier" contextTag="0" optional="true" />
    <ObjectIdentifier name="objectIdentifier" contextTag="1" />
  </Sequence>
</Definitions>
```

An XML representation of an instance of that type assigns values to the members by identifying the member by its name. Optional elements that do not have a value are simply omitted from the XML.

```
<Sequence type="0-BACnetPropertyReference" >
  <ObjectIdentifier name=" objectIdentifier " value="analog-input,0" />
</Sequence>
```

### **X.3.1.2 'type'**

This optional attribute, of type `xs:string`, indicates the type of the element when that element is an instance of a previously defined type.

This is only required in contexts that cannot otherwise determine the type unambiguously. For example, if an instance of a `<Sequence>` is explicitly given a 'type' attribute, then the types of its members are known and the 'type' attribute is not required on the members. If present in this case, however, it shall be exactly equal to the type specified for that member in its definition, unless the defined type is `<Any>`, in which case it is limited to the types allowed by the definition for the `<Any>`. Conversely, if the definition is `<Any>`, an instance shall always specify the type if an explicit type is known.

See Clause X.5 for further description of the use and rules for the 'type' attribute.

### **X.3.1.3 'extends'**

This optional attribute, of type `xs:string`, indicates the name of the existing defined type that is being extended by or within a new definition. If the new definition is not making any structural changes, then the 'type' attribute shall be used rather than the 'extends' attribute. See the description of the 'type' attribute for more information on this distinction.

The XML element type of the existing definition must match the new definition with the exception that, if the existing definition is `<Any>`, then the new definition can be of any type.

See Clause X.5 for further description of the use and rules for the 'extends' attribute.

### **X.3.1.4 'overlays'**

This optional attribute, of type `xs:string`, indicates the name of an existing type that is being augmented with extra metadata. It is used instead of the 'type' attribute to identify the existing type. It is used for elements in the `<Definitions>`

section but does not create a new definition and cannot make any structural changes; therefore, the 'name', and 'extends' attributes are not used either.

The XML element type of the existing definition must match the overlay element type.

A likely use for the “overlays” attribute would be to provide additional localization information to existing type definitions (e.g., translation information made available in a separate “language pack” file).

For example, the following provides Spanish display names for the members of the 0-BACnetDeviceObjectReference type, which was defined elsewhere.

```
<Definitions>
  <Sequence overlays="0-BACnetDeviceObjectReference">
    <ObjectIdentifier name="deviceIdentifier">
      <DisplayName locale="es">Identificador del Dispositivo</DisplayName>
    </ObjectIdentifier>
    <ObjectIdentifier name="objectIdentifier">
      <DisplayName locale="es">Identificador del Objeto</DisplayName>
    </ObjectIdentifier>
  </Sequence>
</Definitions>
```

#### **X.3.1.5 'displayName'**

This optional attribute, of type xs:string, provides a brief human-readable text to associate with the value of an element. This is intended to be a short descriptive identifier (approximately 30 characters or less) for this data element usable for human interface displays like dialog boxes and menus. The text consists of a single line of plain printable characters with no formatting markup. Because the XML representation may wrap and indent attribute values, all contiguous whitespace should be collapsed into a single space for display. The text provided in the “displayName” attribute is in the default locale. The <DisplayName> child element is used to provide display names in alternate locales.

The default value for this attribute is the value of the 'name' attribute, or "" (empty string) if no 'name' attribute is present.

For example, in the following, the default locale is set to "en", so the 'displayName' attribute provides the English display names, and <DisplayName> child elements are used to provide display names for other locales.

```
<CSML defaultLocale="en">
  <Definitions>
    <Sequence name="0-BACnetDeviceObjectReference" >
      <ObjectIdentifier name="deviceIdentifier" displayName="Device Identifier">
        <DisplayName locale="es">Identificador del Dispositivo</DisplayName>
      </ObjectIdentifier>
      <ObjectIdentifier name="objectIdentifier" displayName="Object Identifier">
        <DisplayName locale="es">Identificador del Objeto</DisplayName>
      </ObjectIdentifier>
    </Sequence>
  </Definitions>
</CSML>
```

#### **X.3.1.6 'description'**

This optional attribute, of type xs:string, provides a human readable description of an element. This is intended to be a reasonably complete description of the purpose or use of an element, but does not provide for any “rich text” formatting capabilities. It could be usable as “hover text”, “tool tip” or “pop-up help”. The text consists of plain printable characters

with no formatting markup or line breaks. Because the XML representation may wrap and indent attribute values, all contiguous whitespace should be collapsed into a single space for display. The text provided in the 'description' attribute is in the default locale. The <Description> child element is used to provide display names in alternate locales. Full “rich text” formatted documentation is provided by the <Documentation> child element.

The default value for this is "" (empty string).

For example, in the following, the default locale is set to "en", so the 'description' attribute provides the English descriptions, and <Description> child elements are used to provide descriptions for other locales.

```
<CSML defaultLocale="en">
  <Definitions>
    <Sequence name="0-BACnetDeviceObjectReference" >
      <ObjectIdentifier name="deviceIdentifier" description="The unique device identifier of the
        device containing the referenced object">
        <Description locale="es">El identificador de dispositivo único del dispositivo que contiene
          el objeto referido</Description>
      </ObjectIdentifier>
      <ObjectIdentifier name="objectIdentifier" description="The object identifier of the referenced
        object">
        <Description locale="es">El identificador del objeto del objeto referido</Description>
      </ObjectIdentifier>
    </Sequence>
  </Definitions>
</CSML>
```

### **X.3.1.7 'comment'**

This optional attribute, of type xs:string, provides a human-readable comment for an element. This is usually a technical note intended for readers of the XML itself, rather than users of the data, as displayName, Description, and Documentation are intended. Due to its limited audience, it is not localizable.

The default value for this is "" (empty string).

For example, in the following, an internal comment copied from Clause 21 is intended for readers of the XML, not user interfaces.

```
<Definitions>
  <Sequence name="0-BACnetPropertyReference">
    <Enumerated name="propertyIdentifier" contextTag="0" type="0-BACnetPropertyIdentifier" />
    <Unsigned name="propertyArrayIndex" contextTag="1" optional="true"
      comment="Used only with array datatype. If omitted, the entire array is referenced."/>
  </Sequence>
</Definitions>
```

### **X.3.1.8 'writable'**

This optional attribute, of type xs:boolean, specifies whether the data value is generally expected to be writable. Security concerns or temporary modes of operations may make the data value not writable at any given time, but this attribute represents the general case.

The default value for this attribute is "false".

The following example declares a property of the File Object to be writable.

```
<Definitions>
  < Object name="0-FileObject">
    ...
    <Boolean name="archive" writable="true" ... />
    ...
  </ Object >
</Definitions>
```

### **X.3.1.9 'readable'**

This optional attribute, of type `xs:boolean`, specifies whether the data value is generally expected to be readable using simple value reading services (e.g. `ReadProperty` or `getValue()`). Security concerns or temporary modes of operations may make the data value not readable at any given time, but this attribute represents the general case. An example where this would be "false" is the `Log_Buffer` property of the Trend Log object.

The default value for this attribute is "true".

This example shows that, while rare, some properties are not readable using the simple-value reading services.

```
<Definitions>
  < Object name="0-TrendLogObject">
    ...
    <List name="log-buffer" readable="false" ... />
    ...
  </ Object >
</Definitions>
```

### **X.3.1.10 'commandable'**

This optional attribute, of type `xs:boolean`, specifies whether the data value is commandable using BACnet's command prioritization mechanism described in Clause 19. While "commandable" often implies "writable", the two attributes nonetheless have independent values. It is possible for a definition to declare that `commandable="true"` and `writable="false"`, meaning that, by default, the property is not externally writable, but it is nonetheless commandable in nature.

The default value for this attribute is "false".

The following example declares that the Present Value of an Analog Output Object is writable and commandable.

```
<Definitions>
  < Object name="0-AnalogOutputObject">
    ...
    <Real name="present-value" writable="true" commandable="true" ... />
    ...
  </ Object >
</Definitions>
```

### **X.3.1.11 'associatedWith'**

This optional attribute, of type `xs:string`, indicates a peer element that this element is associated with. The value of this attribute is equal to the value of the 'name' attribute of the referenced peer element. This is primarily for human user interface purposes, to define hints for grouping related elements or to form a display hierarchy from an otherwise flat list of peers. Only one such relationship can be formed for a given element, so that it is not possible to define multiple associations that could result in a grouping conflict or the display of an element in more than one place.

This attribute appears on the dependent or subservient element(s) in a relationship, if such a relationship exists. For example, if there is a many-to-one relationship, then the 'associatedWith' attribute would be present on the "many" elements and would contain the name of the "one" element. If only two elements are involved, the one that is seen as secondary or dependent is given the 'associatedWith' attribute, which refers to the name of the primary element.

An example would be a commandable property in a BACnet object. The `Priority_Array` and `Relinquish_Default` properties would both have an 'associatedWith' attribute which would refer to the name of the `Present_Value` property.

The choice of a "primary" element may seem arbitrary in some groups of peers that have no clear hierarchy or dependency relationship. However, the choice of a primary element is nonetheless important because it may influence a user interface to put that selected element at the top of the list of associated peers. For example, all of the properties associated with intrinsic alarming are equal peers, but they may wish to be "associated with" the `Event_Enable` property as the "primary" since it exists for all algorithms.

Since XML data exchanged between systems is often dynamic and thus not certifiably correct ahead of time, consumers of this XML syntax should be designed defensively to deal with malformed or circular relationships.

The association created by 'associatedWith' is distinct from 'requiredWith'. Elements that are "associated" with each other are nonetheless still independently optional, whereas 'requiredWith' defines constraints to optionality.

The default value for this attribute is "" (empty string), which means that there is no association.

The following example associates the `Priority Array` property with the commandable `Present Value` property.

```
<Definitions>
  <Object name="0-AnalogOutputObject">
    ...
    <Array name="priority-array" associatedWith="present-value" ... />
    ...
  </Object>
</Definitions>
```

### **X.3.1.12 'requiredWith'**

This optional attribute, of type `xs:string`, indicates a list of peer optional elements that an optional element's presence is tied to. When any of the named peer elements is present, then the current element will be present as well. No implication is made about the reverse situation - if all of the peer elements are absent, the current element may be present or absent for other reasons. The value of this attribute is equal to a comma-separated concatenation of the values of the 'name' attributes of the referenced peer elements.

One of the purposes of this attribute is to allow clients to avoid attempts to read the "dependent" optional elements if the "primary" optional element is known to be absent.

An example would be the `Inactive_Text` property indicating that it is 'requiredWith' the `Active_Text` property.

Since XML data exchanged between systems is often dynamic and thus not certifiably correct ahead of time, consumers of this XML syntax should be designed defensively to deal with malformed or circular relationships.

The default value for this attribute is "" (empty string), which means that there is no dependency.

The following example connects the presence of two optional properties so that if either is present then they are both present.

```
<Definitions>
  <Object name="0-BinaryInputObject">
    ...
    <String name="inactive-text" optional="true" requiredWith="active-text" ... />
    <String name="active-text" optional="true" requiredWith="inactive-text" ... />
    ...
  </Object>
</Definitions>
```

The following example connects the presence of three optional properties so that if any is present, then they are all present.

```
<Definitions>
  <Object name="0-BinaryInputObject">
    ...
    <DateTime name="change-of-state-time" optional="true"
      requiredWith="change-of-state-count,time-of-state-count-reset" ... />
    <Unsigned name="change-of-state-count" optional="true"
      requiredWith="change-of-state-time,time-of-state-count-reset" ... />
    <DateTime name="time-of-state-count-reset" optional="true"
      requiredWith="change-of-state-time,change-of-state-count" ... />
    ...
  </Object>
</Definitions>
```

### **X.3.1.13 'requiredWithout'**

This optional attribute, of type xs:string, indicates a list of peer optional elements that an optional element's presence is tied to. When any of the named peer elements is absent, then the current element will be present. No implication is made about the reverse situation - if all of the peer elements are present, the current element may be present or absent for other reasons. The value of this attribute is equal to a comma-separated concatenation of the values of the 'name' attributes of the referenced peer elements.

One of the purposes of this attribute is to allow clients to know that, if an optional element is absent, then another is available, often as an alternative for a related purpose.

Since XML data exchanged between systems is often dynamic and thus not certifiably correct ahead of time, consumers of this XML syntax should be designed defensively to deal with malformed or circular relationships.

The default value for this attribute is "" (empty string), which means that there is no dependency.

The following example connects the presence of two optional properties so that if either is absent then the other must be present present.

```
<Definitions>
  <Object name="0-ScheduleObject">
    ...
    <String name="weekly-schedule" optional="true" requiredWithout="exception-schedule" ... />
    <String name="exception-schedule" optional="true" requiredWithout="weekly-schedule" ... />
  </Object>
</Definitions>
```

```

...
</Object>
</Definitions>

```

**X.3.1.14 'notPresentWith'**

This optional attribute, of type xs:string, indicates a list of peer optional elements that an optional element's presence is tied to in a negative way. When any of the named peer elements is present, then the current element will be absent. No implication is made about the reverse situation. If all of the peer elements are absent, the current element may be present or absent for other reasons. The value of this attribute is equal to a comma-separated concatenation of the values of the 'name' attributes of the referenced peer elements.

This attribute usually appears on the dependent element(s) in a relationship. For example, if there is a many-to-one relationship, then the 'notPresentWith' attribute would be present on the "many" elements and would contain the name of the "one" element. If only two elements are involved, then the one that is seen as dependent is given the 'notPresentWith' attribute, which refers to the name of the primary element. If neither is dependent, then the choice of primary is arbitrary, or they may each refer to each other.

One of the purposes of this attribute is to allow clients to know which sets of properties are mutually exclusive and to thus avoid attempts to read the "dependent" optional elements if the "primary" optional element is known to be present.

Since XML data exchanged between systems is often of dynamic in origin and thus not certifiably correct ahead of time, consumers of this XML syntax should be designed defensively to deal with malformed or circular relationships.

The default value for this attribute is "" (empty string), which means that there is no dependency.

The following example connects the presence of three optional properties where two are present as a pair but are mutually exclusive with a third.

```

<Definitions>
  <Object name="999-ExampleObject">
    ...
    <Real name="high-limit" optional="true" requiredWith="low-limit" notPresentWith="limits" ... />
    <Real name="low-limit" optional="true" requiredWith="high-limit" notPresentWith="limits" ... />
    <Sequence name="limits" optional="true" notPresentWith="high-limit,low-limit" ... />
    ...
  </Object>
</Definitions>

```

**X.3.1.15 'writableWhen'**

This optional attribute, of type xs:restriction of xs:string, indicates the conditions under which the data value may be writable. The choices for the value and their meanings are defined in the following table.

**Table X.x** Standard Rules for Writability Requirements

Attribute Value	Meaning
"out-of-service"	When Out_Of_Service is TRUE
"commandable"	When this property is commandable
"other"	Non-standard requirement. Descriptive text should be provided by <WritableWhen> elements

The default value for this attribute is "" (empty string) unless one or more <WritableWhen> child elements is specified, in which case, the default value is "other". Therefore, a <WritableWhen> child element may be present without requiring the presence of the 'writableWhen' attribute. However, if a value for the 'writableWhen' attribute is specified and is not equal to "other", then <WritableWhen> child elements are not inherited and no <WritableWhen> child elements shall be specified in the same context.

When this attribute is equal to "other", optional <WritableWhen> elements can be used to provide localized text to describe the nonstandard condition.

A common case in Clause 12 objects is the requirement that Present Value be writable when Out Of Service is true.

```
<Definitions>
  <Object name="0-AnalogInputObject">
    ...
    <Real name="present-value" writableWhen="out-of-service" ... />
    ...
  </Object>
</Definitions>
```

**X.3.1.16 'requiredWhen'**

This optional attribute, of type xs:restriction of xs:string, indicates the conditions under which optional elements must be present. The choices for the value and their meanings are defined in the following table.

**Table X.x** Standard Rules for Presence Requirements

Attribute Value	Meaning
"intrinsic-supported"	If the object supports intrinsic reporting
"cov-notify-supported"	If the object supports COV reporting
"cov-subscribe-supported"	If the device supports execution of either the SubscribeCOV or SubscribeCOVProperty service
"present-value-commandable"	If Present_Value is commandable
"segmentation-supported"	If Segmentation of any kind is supported
"virtual-terminal-supported"	If Virtual Terminal services are supported
"time-sync-execution"	If the device supports the execution of the TimeSynchronization service
"utc-time-sync-execution"	If the device supports the execution of the UTCTimeSynchronization service
"time-master"	If the device is a Time Master
"backup-restore-supported"	If the device supports the backup and restore procedures
"slave-proxy-supported"	If the device is capable of being a Slave-Proxy device
"slave-discovery-supported"	If the device is capable of being a Slave-Proxy device that implements automatic discovery of slaves
"other"	Non-standard requirement. Descriptive text should be provided by <RequiredWhen> elements

The default value for this attribute is "" (empty string) unless one or more <RequiredWhen> child elements is specified, in which case, the default value is "other". Therefore, a <RequiredWhen> child element may be present without requiring the presence of the 'requiredWhen' attribute. However, if a value for the 'requiredWhen' attribute is specified and is not equal to "other", then <RequiredWhen> child elements are not inherited and no <RequiredWhen> child elements shall be specified in the same context.

When this attribute is equal to "other", optional <RequiredWhen> elements can be used to provide localized text to describe the nonstandard condition.

Many properties in Clause 12 objects have their presence dependent on a standard condition.

```
<Definitions>
  <Object name="0-DeviceObject">
    ...
    <Unsigned name="max-segments-accepted" optional="true"
      requiredWhen="segmentation-supported" ... />
    ...
  </Object>
</Definitions>
```

#### **X.3.1.17 'writeEffective'**

This optional attribute, of type xs:enumeration of xs:string, is an indication of when a write to this value will be effective. The choices are: "immediately", "delayed", "on-program-restart", and "on-device-restart". The actual time delay associated with the "delayed" case is not specified, but it is nonetheless an indication that the effect of the write should not be expected to be immediate.

The default value for this attribute is "immediate".

This example shows that a setting controlling how much memory is allocated to audit logs is effective only after the next device restart.

```
<Definitions>
  <Object name="999-MemoryControlObject ">
    ...
    <Unsigned name="max-audit-log-space" units="percent" writeEffective="on-device-restart" ... />
    ...
  </Object>
</Definitions>
```

#### **X.3.1.18 'optional'**

This optional attribute, of type xs:boolean, used only in definitions, indicates that this element may not be present in an instance of this definition. This attribute can only be set to "true" when an element is initially defined. Subsequent definitions that inherit the element may set the value to "false" if the element will always be present in instances of that new definition, or they may set the 'absent' attribute to "true" to indicate that the element will never be present in an instance of that new definition.

An example case for this would be where the standard definition of a BACnet Analog Input declares the Description property to be optional by setting the 'optional' attribute to "true", but a specific vendor's extension to that type declares that every instance will have a Description property present by setting the 'optional' attribute to "false", or it declares that every instance will never have a Description property present by setting the 'absent' attribute to "true".

The default value of this attribute is "false".

See the description of the 'absent' attribute for an example of the interaction between the 'optional' and 'absent' attributes.

### X.3.1.19 'absent'

This optional attribute, of type `xs:boolean`, used only in definitions, indicates that an optional element will not be present in instances of that definition.

The default value of this attribute is "false".

An example of the use of this attribute would be where the standard definition for `BACnetDeviceObjectReference` has the 'deviceIdentifier' field marked as optional, but a specific vendor's device does not support references outside the device, so it can derive a new definition from the standard definition and set the 'absent' attribute on the 'deviceIdentifier' field to be "true" so that clients of that device would never try to write a deviceIdentifier to it.

In a standard definitions file:

```
<Definitions>
  <Sequence name="0-BACnetDeviceObjectReference">
    <ObjectIdentifier name="deviceIdentifier" optional="true" ... />
    <ObjectIdentifier name="objectIdentifier" optional="true" ... />
  </Sequence>
</Definitions>
```

In a vendor-specific file:

```
<Definitions>
  <Sequence name="999-LimitedDeviceObjectReference" extends="0-BACnetDeviceObjectReference">
    <ObjectIdentifier name="deviceIdentifier" absent="true" />
  </Sequence>
</Definitions>
```

### X.3.1.20 'variability'

This optional attribute, of type `xs:enumeration` of `xs:string`, indicates when and how the value of this element is expected to change over time. The choices are: "constant", "configuration-setting", "operation-setting", and "status". A value marked as "constant" is expected to not change, so clients can just read it once or use the value provided in XML. Values marked as "configuration-setting" are expected to be non-volatile settings that are made only during configuration or commissioning. Values marked as "operation-setting" are user settings like setpoints, alarm limit, etc. that are expected to change relatively infrequently, whether by operator or programmed control events. Values marked "status" are potentially continuously variable values representing the live status of calculated or measured quantities.

The default value of this attribute is undefined, meaning that the variability of the element is unknown.

In this example, the definition of an object indicates that the "max-audit-log-space" property is a value that is intended to be set when the device is commissioned, not as an on-going part of its operation, and therefore changes infrequently. It also indicates that the "audit-log-alarm-limit" is expected to be changed by an outside entity occasionally during the course of operation of the device, and that the "audit-log-space-used" is a status value that changes by itself at any time.

```
<Definitions>
  <Object name="999-MemoryControlObject ">
    ...
    <Unsigned name="max-audit-log-space" variability="configuration-setting" ... />
    <Unsigned name="audit-log-alarm-limit" variability="operation-setting" ... />
    <Unsigned name="audit-log-space-used" variability="status" ... />
  </Object>
</Definitions>
```

```
...
</Object>
</Definitions>
```

### **X.3.1.21 'volatility'**

This optional attribute, of type `xs:enumeration` or `xs:string`, indicates how values that are written are retained. The choices are "volatile", "nonvolatile", and "nonvolatile-limited-writes". The "volatile" case indicates that a written value may be forgotten over device resets and power failures. The "nonvolatile" case indicates that values are intended to survive device resets and power failures. And the "nonvolatile-limited-writes" is an extension to "nonvolatile" that indicates that the value is written to a form of memory that has a limited number of write cycles before wearing out, indicating to clients that this value should not be continuously changed.

The default value of this attribute is undefined, meaning that the volatility of the element is unknown.

In this example, the definition of an object indicates that the "output-percent" property is a volatile commanded value that will likely not survive a device reset or power failure and should therefore be checked or refreshed periodically as needed. It also indicates that the "alarm-threshold" should not be continuously written to as a part of normal operation.

```
<Definitions>
  <Object name="999-FanControlObject ">
    ...
    <Unsigned name="output-percent" volatility="volatile" ... />
    <Unsigned name="alarm-threshold" volatility="nonvolatile-limited-writes" ... />
    ...
  </Object>
</Definitions>
```

### **X.3.1.22 'contextTag'**

This optional attribute, of type `xs: nonNegativeInteger`, indicates the context tag that should be used when encoding this element in ASN.1 according to the rules in Clause 20. If this attribute is absent, then the element is "application tagged" according to the rules in Clause 20.

If this attribute is absent, then the element is "application tagged" when encoding in ASN.1.

For example, because the `deviceIdentifier` field of the `BACnetDeviceObjectReference` construct is optional, the fields are context tagged.

```
<Definitions>
  <Sequence name="0-BACnetDeviceObjectReference">
    <ObjectIdentifier name="deviceIdentifier" contextTag="0" optional="true" />
    <ObjectIdentifier name="objectIdentifier" contextTag="1" />
  </Sequence>
</Definitions>
```

### **X.3.1.23 'propertyIdentifier'**

This optional attribute, of type `xs:nonNegativeInteger`, indicates the property identifier that is to be used when accessing this element's value as a BACnet property.

If this attribute is absent, then the element is not intended to be accessed as a BACnet property.

The following example declares that the Present Value of an Analog Output Object is accessible with property identifier 85.

```
<Definitions>
  <Object name="0-AnalogOutputObject">
    ...
    <Real name="present-value" propertyIdentifier="85" ... />
    ...
  </ Object >
</Definitions>
```

### **X.3.2 Common Child Elements**

All BACnet data elements share a common set of optional child elements. In addition to the common elements described here, each primitive data element may also define a specific set of required or optional child elements of its own. This is done in individual clauses that define those data elements.

#### **X.3.2.1 <DisplayName>**

This optional child element, of type `xs:string`, is used to provide alternate locale values for the 'displayName' attribute. This element has a required 'locale' attribute, of type `xs:language`, that identifies the locale for the string value. Display names in the default locale shall use the 'displayName' attribute. The `<DisplayName>` element is therefore only for locales different from the default. The text consists of plain printable characters with no formatting markup or line breaks.

See the description for the 'displayName' attribute for an example of the `<DisplayName>` element.

#### **X.3.2.2 <Description>**

This optional child element, of type `xs:string`, is used to provide alternate locale values for the 'description' attribute. This element has a required 'locale' attribute, of type `xs:language`, that identifies the locale for the string value. Descriptions in the default locale shall use the 'description' attribute. The `<Description>` element is therefore only for locales different from the default. The text consists of plain printable characters with no formatting markup or line breaks.

See the description for the 'description' attribute for an example of the `<Description>` element.

#### **X.3.2.3 <Documentation>**

This optional child element, of type "mixed content" (plain text and XHTML markup), is used to provide formatted "rich text" documentation on the purpose and use of an element. This element has an optional 'locale' attribute, of type `xs:language`, that identifies the locale for the text. Since there is no attribute form for the `<Documentation>` information, the 'locale' attribute is optional for this element; its absence indicates that the text is for the default locale. The "mixed context" type allows plain text combined with markup consisting of well-formed XML elements conforming to the XHTML namespace "<http://www.w3.org/1999/xhtml>".

The following example shows some formatted text in a `<Documentation>` element.

```
<Definitions>
  < Object name="999-ExampleObject">
    <Real name="a-good-property" ... >
```

```
<Documentation locale="en">This property documentation contains <b>bold</b> words
and is spread over several lines (all <i>white space</i> in XHTML is collapsed to a
single space)</Documentation>
</Real>
</ Object >
</Definitions>
```

#### **X.3.2.4 <WritableWhen>**

This optional child element, of type `xs:string`, is used to provide localized display text for the writability condition when the 'writableWhen' attribute has the value of "other". This element has an optional 'locale' attribute, of type `xs:language`, that identifies the locale for the text. If the 'locale' attribute is absent, then the text is for the default locale. While the 'writableWhen' attribute is an enumeration of fixed strings as defined by this standard, the `<WritableWhen>` element contains variable text consisting of plain printable characters with no formatting markup or line breaks.

For example, if the writability condition is not one of the standard conditions, then the 'writableWhen' attribute has the value of "other" and the `<WritableWhen>` elements provide the display text (the default locale in this example is "en").

```
<Unsigned name="trendMemoryAllocation" writableWhen="other" >
  <WritableWhen>The Device object's Device Status property is "download required"</WritableWhen>
</Unsigned>
```

If a `<WritableWhen>` element is present in a context without the 'writableWhen' attribute, the 'writableWhen' attribute is implicitly assigned the value "other".

For example, the following shows that it is not necessary to include `writableWhen="other"` in a context with `<WritableWhen>` elements.

```
<Unsigned name="aux-input" >
  <WritableWhen>The "Aux Disable" property is TRUE</WritableWhen>
</Unsigned>
```

#### **X.3.2.5 <RequiredWhen>**

This optional child element, of type `xs:string`, is used to provide localized display text for the presence requirements when the 'requiredWhen' attribute has the value of "other". This element has an optional 'locale' attribute, of type `xs:language`, that identifies the locale for the text. If the 'locale' attribute is absent, then the text is for the default locale. While the 'requiredWhen' attribute is an enumeration of fixed strings as defined by this standard, the `<RequiredWhen>` element contains variable text consisting of plain printable characters with no formatting markup or line breaks.

For example, if the presence requirement is not one of the standard conditions, then the 'requiredWhen' attribute has the value of "other" and the `<RequiredWhen>` elements provide the display text (the default locale in this example is "en").

```
<Unsigned name="auxbaud" optional="true" requiredWhen="other" >
  <RequiredWhen>The device is configured as a gateway</RequiredWhen>
</Unsigned>
```

If a `<RequiredWhen>` element is present in a context without the 'requiredWhen' attribute, then the 'requiredWhen' attribute is implicitly assigned the value "other".

For example, the following shows that it is not necessary to include `requiredWhen="other"` in a context with `<RequiredWhen>` elements.

<Unsigned name="aux-limit" >  
 <RequiredWhen>The object is configured to supports aux input<RequiredWhen>  
 </Unsigned>

**X.3.2.6 <Extensions>**

This optional child element is used to hold extra information that is not directly supported by the elements and attributes defined by this annex. Each extended piece of information, represented as child elements of the <Extensions> element, is identified by its 'name' attribute. Extended data is not restricted in type or depth.

There are no requirements for processing extensions. Consumers of the XML defined in this annex are allowed to consume extensions that are known to the consumer and to ignore the rest.

Extension mechanisms are described more fully in Clause X.7.

**X.3.3 Named Values**

In addition to Enumerations and BitStrings, most other primitive BACnet data elements can also have special values that are represented by textual identifiers rather than, or in addition to, their raw value form. Some of these values may be "special values" that are actually outside the normal restricted range of values.

In all cases, the mapping from the underlying value form to the human presentation form is done by an optional <NamedValues> child element, the children of which provide the individual mappings. The types of the child elements of <NamedValues> are appropriate to the mapping that is required and are described in the table below.

**Table X.x** Types and Meanings of the Child Elements of <NamedValues>

Data Element Type	Child Element Type	Meaning of Child Elements
<BitString>	<Unsigned>	The value of the <Unsigned> element provides the position of the bit in the encoded BitString, and the 'displayName' attribute can be used to provide a textual presentation for the individual bit. If a value is not provided, the next available value is automatically assigned, starting at 0, in the order of the child elements in the XML.
<Enumerated>	<Unsigned>	The value of the <Unsigned> element provides the numeric value for the encoded enumeration choice, and the 'displayName' attribute can be used to provide a textual presentation of the enumeration choice. If a value is not provided, the next available value is automatically assigned, starting at 0, in the order of the child elements in the XML.
<Boolean>	<Boolean>	Two <Boolean> elements, one with a value of "true" and the other with a value of "false", may be use to provide a 'displayName' attribute that can be used as an alternate textual presentation the underlying values of "true" and "false".
<Date> <DateTime> <Double> <Integer> <ObjectIdentifier> <OctetString> <Real>	(same as enclosing element)	The child elements provide the definition of special values. These values may be outside the range of valid values created by 'maximum' and 'minimum' and 'resolution' attributes.  The 'displayName' attributes of these special values may be used in place of the actual underlying value, if

<pre>&lt;String&gt; &lt;Time&gt; &lt;Unsigned&gt; &lt;WeekNDay&gt; &lt;WildDate&gt; &lt;WildTime&gt;</pre>		<p>desired and appropriate, or this information may simply be used to allow the special values to be considered valid even though they are otherwise outside the valid range.</p>
--	--	---

An example `<BitString>` shows the use of `<Unsigned>` child element to define textual names for the bits and assign the bit positions.

```
<BitString name="0-BACnetLimitEnable" length="2" ... >
  <NamedValues>
    <Unsigned name="lowLimitEnable" value="0" ... />
    <Unsigned name="highLimitEnable" value="1" ... />
  </NamedValues>
</BitString>
```

An example `<Enumerated>` shows the use of `<Unsigned>` child element to define textual names for the enumerated values states and assign the equivalent numeric value.

```
<Enumerated name="0-BACnetObjectType" minimum="128" maximum="1023" ... >
  <NamedValues>
    <Unsigned name="accumulator" value="23" ... />
    <Unsigned name="analog-input" value="1" ... />
    ...
    <Unsigned name="trend-log" value="20" ... />
  </NamedValues>
</Enumerated>
```

An example `<Boolean>` shows the use of `<Boolean>` child elements to assign alternate text for the boolean states "true" and "false".

```
<Boolean name="issueConfirmedNotifications" ... >
  <NamedValues>
    <Boolean name="confirmed" value="true" displayName="Confirmed" ... />
    <Boolean name="unconfirmed" value="false" displayName="Unconfirmed" ... />
  </NamedValues>
</Boolean>
```

The `<NamedValues>` element can only appear in a definition context because adding new named values constitutes a structural change to the data. When inheriting a `<NamedValues>` element from a definition, the newly specified child elements are logically added to the end of the list of existing child elements of the inherited `<NamedValues>`. The order of the child element is significant since it is used for auto numbering. See Clause X.5 for more on definitions and inheritance.

When primitive data elements are used as child elements of `<NamedValues>`, there are optional attributes, 'displayNameForWriting', 'notForWriting' and 'notForReading', and an optional child element, `<DisplayNameForWriting>`, that are available to them to provide extra information specifically for their use in the context of `<NamedValues>`. These attributes and child elements have no meaning outside of that context.

### **X.3.3.1 'displayNameForWriting'**

This optional attribute, of type `xs:string`, provides an alternate display name for use when the named value is used for writing, as opposed to when it is presented as a result of reading.

An example of this could be an Enumeration representing alarm states where the value zero would be presented as "No Alarm" when read, and "Reset" when written.

The default value of this attribute is the value of the 'displayName' attribute.

In this example, the "false" state has a different presentation when read than it does when written. This can be used to provide the "adjective for reading, verb for writing" pattern.

```
<Boolean name="tripwire">
  <NamedValues>
    <Boolean name="tripped" value="true" displayName="Tripped" ... />
    <Boolean name="armed" value="false" displayName="Armed" displayNameForWriting="Reset"/>
  </NamedValues>
</Boolean>
```

### **X.3.3.2 'notForWriting'**

This optional attribute, of type `xs:string`, is an indicator that a special value or a mapped enumeration value is not to be used for writing. It may appear when read, but an attempt to write it will likely be unsuccessful.

The default value of this attribute is "false".

An example of this could be an Enumeration representing alarm states where the value zero is the only value that can be written. In this case, every child of `<NamedValues>` other than the one for the value zero would be marked as 'notForWriting'.

Using the "tripwire" example from the description of the 'displayNameForWriting' attribute, if the "tripped" state is not allowed to be written, then that fact can be declared by using the 'notForWriting' attribute on the "true" state.

```
<Boolean name="tripwire">
  <NamedValues>
    <Boolean name="tripped" value="true" displayName="Tripped" notForWriting="true" />
    <Boolean name="armed" value="false" displayName="Armed" displayNameForWriting="Reset"/>
  </NamedValues>
</Boolean>
```

### **X.3.3.3 'notForReading'**

This optional attribute, of type `xs:string`, is an indicator that a special value is not to be used when displaying a value as a result of reading. It may appear as a special writable choice, but the corresponding underlying value should be presented when read.

The default value of this attribute is "false".

An example of this would be an Unsigned value representing the number of records collected, where zero is displayed numerically along with all other values, but the only value that is writable is a special value named "Clear" which also has the numeric value of zero but is marked 'notForReading' so that it is only used as a named choice for writing and is not used when the read value is zero.

```
<Unsigned name="recordCount" minimumForWriting="0" maximumForWriting="0">
  <NamedValues>
    <!-- this is marked notForReading, so 0 will show as "0" when read -->
    <Unsigned name="clear" value="0" displayNameForWriting="Clear" notForReading="true"/>
  </NamedValues>
</Unsigned>
```

#### **X.3.3.4 <DisplayNameForWriting>**

This optional child element, of type `xs:string`, is used to provide alternate locale values for the 'displayNameForWriting' attribute. This element has a required 'locale' attribute, of type `xs:language`, that identifies the locale for the string value. Display names for writing in the default locale shall use the 'displayNameForWriting' attribute. The <DisplayNameForWriting> element is therefore only for locales different from the default.

### **X.3.4 Primitive Values**

The primitive data elements, other than <Null>, each have a way to represent their value in XML. Most use only the 'value' attribute of an appropriate type, but the <String> and <OctetString> also have extended forms of value for large or multi-locale values.

The primitive data elements <BitString>, <Boolean>, <Date>, <DatePattern>, <DateTime>, <Double>, <Enumerated>, <Integer>, <ObjectIdentifier>, <Real>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay> all can specify their values in attribute form as described in this clause. In addition to the attribute form of value, the data elements <OctetString> and <String> can also specify their values in element form using the <Value> child element.

#### **X.3.4.1 'value'**

This optional attribute, of the type specified in Table X.x, provides the value for the data element.

For the <String> element, the 'value' attribute represents the data value in the default locale. Values in other locales, or values containing character data unsuitable for XML attributes, are represented using the optional child element <Value>. Since the 'value' attribute and a <Value> child element specifying the default locale are two ways to specify the same value, they are mutually exclusive in the same XML context, and when one is present it overrides the other that may have been inherited.

For the <OctetString> element, the 'value' attribute, of type `xs:hexBinary`, is in hexadecimal format, which is easier to process both for humans and machines, but is not as succinct as `xs:base64Binary` for large amounts of data. If a short amount of data is to be conveyed, the attribute form should be used. However, if a large amount of data is to be conveyed, the optional <Value> child element, of type `xs:base64Binary`, should be used. The threshold to select between the two methods is a local matter. Since the 'value' attribute and the <Value> child element are two ways to specify the same value, they are mutually exclusive in the same XML context, and when one is present it overrides the other that may have been inherited.

#### **X.3.4.2 'unspecifiedValue'**

This optional attribute, of type `xs:boolean`, indicates that a value for a <Date>, <DateTime>, <Time> or <ObjectIdentifier> is unspecified. This condition is encoded in binary as all octets equal to 255.

For example, in this pair of date properties, only the start date is specified.

```
<Date name="start-date" value="2008-06-15" />  
<Date name="end-date" unspecifiedValue="true" />
```

This attribute applies only to the <Date>, <DateTime>, and <Time> data elements. Its default value is "true" but becomes "false" when a 'value' attribute is provided.

The 'value' attribute, the <Value> element, and the 'unspecifiedValue' attribute are all mutually exclusive and shall not be present in the same context. The presence of any one of them in an instance overrides any one that was inherited from a definition.

#### **X.3.4.3 'charset'**

This optional attribute, of type `xs:restriction of xs:string`, describes the character set that was used to encode BACnet character string data. The choices are "ansi", "dbcs", "jis", "unicode4", "unicode2", "iso8859", and "utf8". This attribute applies only to the string value in the default locale.

This attribute only applies to <String> elements. If not present, then the character set is unknown or undefined.

The 'charset' attribute is indivisibly part of the value of the element and is not specified or inherited separately from the 'value' attribute or the <Value> element. If a 'value' attribute or <Value> element is specified in an instance without a 'charset' attribute, then the character set reverts to unknown or undefined. The 'charset' attribute shall not be specified without also specifying the 'value' attribute or the <Value> element in the same context.

#### **X.3.4.4 'codepage'**

This optional attribute, of type `xs:nonNegativeInteger`, describes the code page that was used to encode BACnet character string data. This attribute has meaning only when the 'charset' attribute has the value "dbcs". This attribute applies only to the string value in the default locale.

This attribute only applies to <String> elements and shall be present if and only if the 'charset' attribute is present and has the value "dbcs".

The 'codepage' attribute is indivisibly part of the value of the element and is not specified or inherited separately from the 'value' attribute or the <Value> element. If a 'value' attribute or <Value> element is specified in an instance without a 'codepage' attribute, or the 'charset' attribute is present and does not have the values of 'dbcs', then the 'codepage' attribute reverts to undefined. The 'codepage' attribute shall not be specified without also specifying the 'value' attribute or the <Value> element in the same context.

#### **X.3.4.5 'length'**

This optional attribute, of type `xs:nonNegativeInteger`, specifies the length of Bit String data, in bits. This is the length of the actual data bits and does not include any extra encoding overhead.

The default value of this attribute is undefined, meaning the length of the Bit String is variable or not known. An unknown length is acceptable for definitions when a value for the <BitString> is not provided. However, the length of a <BitString> value must be known to properly process the value. Therefore, if a 'length' attribute is not specified on the definition of a <BitString>, then it shall be present on any instance that contains a value.

This attribute only applies to <BitString> elements.

##### **X.3.4.5.1 <Value>**

This optional child element provides the value for the <String> and <OctetString> data elements. It is not usable by any other data element.

For the <String> element, the optional <Value> child element, of type xs:string, contains the value for a particular locale. If the optional 'locale' attribute, of type xs:language, is specified, then the value is for that locale. If the 'locale' attribute is missing, then the value is for the default locale.

The 'value' attribute, when used, represents the value in the default locale. the data value in the default locale Short values in the default locale should use the attribute form, while longer values should use the element form. The threshold to select between the two is a local matter.

Values in other locales, or values containing character data unsuitable for XML attributes, are represented using the optional child element <Value>.

For the <OctetString>, the 'value' attribute, of type xs:hexBinary, is in hexadecimal format which is easier for human creation and consumption but is not as efficient as xs:base64Binary for large amounts of data. If a short amount of data is to be conveyed, the attribute form is simpler to process. However, if a large amount of data is to be conveyed, the optional <Value> child element, of type xs:base64Binary, can be used to reduce the size of the XML.

### X.3.5 Range Restrictions

Primitive data that expresses a continuous range of values can have that range restricted by optional attributes. These attributes can be used to specify the high and low ends of the range and the minimum increment of the values.

The attributes that are used for restricting the range of primitive data elements are specified in the following clauses. In the case of the <ObjectIdentifier> element, the range restrictions apply to the instance portion of the value only.

The type and applicability of the range restriction attributes are summarized in Table X.x.

**Table X.x** Range Restriction Attributes

Data Element	Attribute Name	Attribute Type
<Date>	minimum maximum minimumForWriting maximumForWriting	xs:date
<DateTime>	minimum maximum minimumForWriting maximumForWriting	xs:dateTime
<Double>	minimum maximum minimumForWriting maximumForWriting resolution	xs:double
<Enumerated>	minimum maximum minimumForWriting maximumForWriting	xs:nonNegativeInteger
<Integer>	minimum maximum minimumForWriting maximumForWriting resolution	xs:integer

<ObjectIdentifier>	minimum maximum minimumForWriting maximumForWriting	xs:nonNegativeInteger
<Real>	minimum maximum minimumForWriting maximumForWriting resolution	xs:float
<Time>	minimum maximum minimumForWriting maximumForWriting	xs:time
<Unsigned>	minimum maximum minimumForWriting maximumForWriting resolution	xs:nonNegativeInteger

### X.3.5.1 'minimum'

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the continuous range of values.

The default value of this attribute is undefined, meaning that the value is unlimited or that the limit is unknown.

An example of this attribute is given in the description of the 'maximumForWriting' attribute.

### X.3.5.2 'maximum'

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound on the continuous range of values.

The default value of this attribute is undefined, meaning that the value is unlimited or that the limit is unknown.

An example of this attribute is given in the description of the 'maximumForWriting' attribute.

### X.3.5.3 'minimumForWriting'

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the continuous range of values when the value is written.

The default value of this attribute is the value of the 'minimum' attribute.

An example of this attribute is given in the description of the 'maximumForWriting' attribute.

### X.3.5.4 'maximumForWriting'

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound on the continuous range of values when the value is written.

The default value of this attribute is the value of the 'maximum' attribute.

An example of this would be a value that has separate read and write ranges. This <Unsigned> can read values up to 150%, but can't be written with a value greater than 100%.

```
<Unsigned name="motor-speed" minimum="0" maximum="150" units="percent"
    minimumForWriting="0" maximumForWriting="100" />
```

### **X.3.5.5 'resolution'**

This optional attribute, of the type specified in Table X.x, provides the minimum increment that occurs between values. If this attribute is specified, then the value will be in increments of this attribute, starting at the value of the 'minimum' attribute if it is specified, or starting at zero if the 'minimum' attribute is not specified.

The default value of this attribute is undefined, meaning that the resolution of the value is set by the capabilities of the underlying XML data type of the 'value' attribute.

In this example, a normally continuous <Real> declares that it only represents values in increments of 10, starting at -35. So the valid values would be -35, -25, -15, -5, 5, 15, 25, and 35.

```
<Real name="position" minimum="-35.0" maximum="35.0" resolution="10.0" />
```

## **X.3.6 Engineering Units**

Primitive data elements that express a continuous range of values often have known engineering units associated with those values. The attributes and child element defined here only apply to the numeric data types <Double>, <Integer>, <Real>, and <Unsigned>.

### **X.3.6.1 'units'**

This optional attribute, of type xs:restriction of xs:string, describes the engineering units for the numeric value, if known. The choices are the ASN enumeration names of the BACnetEngineeringUnits production in Clause 21 (i.e. "meters-per-second-per-second", "square-meters", ... "watts-per-square-meter-degree-kelvin").

When this attribute is equal to "other", optional <Units> elements can be used to provide localized text to describe the nonstandard units.

The default value of this attribute is "other"; therefore, a <Units> child element may be present without requiring the presence of the 'units' attribute. However, if a value for the 'units' attributes is specified and is not equal to "other", then <Units> child elements are not inherited and no <Units> child elements shall be specified in the same context.

See the description of the <Units> for an example usage.

### **X.3.6.2 <Units>**

This optional child element, of type xs:string, is used to provide localized display text for the units when the 'units' attribute has the value of "other". This element has an optional 'locale' attribute, of type xs:language, that identifies the locale for the text. If the 'locale' attribute is absent, then the text is for the default locale. While the 'units' attribute is an enumeration of fixed strings as defined by this standard, the <Units> element is a free-form plain text whose contents is a local matter.

For example, if the engineering units is not one of the standard units, then the 'units' attribute has the value of "other" and the <Units> elements provide the display text (the default locale in this example is "en").

```
<Real name="snailspeed" units="other" >
  <Units>Inches/Week</Units>
  <Units locale="de">Zoll/Woche</Units>
</Real>
```

If a <Units> element is present in a context without the 'units' attribute, then the 'units' attribute is implicitly assigned the value "other".

For example, the following shows that it is not necessary to include units="other" in a context with <Units> elements.

```
<Real name="warpspeed" >
  <Units>lightyears/year</Units>
</Real>
```

### **X.3.7 Data Validity**

Data elements that express live data from measurements or calculations may become unreliable or unavailable for some reason and this syntax supports additional qualifiers to indicate the validity of the data values.

These data validity qualifiers are allowed on any primitive data element and apply to the value of that element. They are also allowed on any constructed data element and apply to the values of all the child elements that constitute the value of the construction, unless overridden by a child element's individual data validity qualifiers.

Values may have varying degrees of validity. Data that is known to be in error is represented with the 'error' attribute, while the 'valueAge' attribute can be used to let the client gauge the staleness of a value that was retrieved or calculated successfully at some point in the past.

Display text for nonstandard error conditions is provided with optional child elements.

#### **X.3.7.1 'valueAge'**

This optional attribute, of type `xs:nonNegativeInteger`, indicates the number of seconds since the last successful update of the value of an element. Note that, like the value of a dynamic quantity itself, the value of this attribute is only accurate at the moment the XML is generated.

Absence of this attribute indicates that the age of the value is unknown. This attribute is not inherited from a definition, so its presence in a definition is meaningless.

#### **X.3.7.2 'error'**

This optional attribute, of type `xs:nonNegativeInteger`, indicates an error that affects the validity of the value of an element. If the 'error' attribute is present, then the value of the element should not be trusted to be valid. The error numbers are defined Clause N.13. When this attribute is equal to 0, meaning "unspecified error", optional <Error> elements can be used to provide localized text to describe the error condition.

When no known error condition exists, this attribute shall be absent. This attribute is not inherited from a definition, so its presence in a definition is meaningless.

See the description of the <Error> element for an example usage.

**X.3.7.2.1 <Error>**

This optional child element, of type `xs:string`, is used to provide localized display text for the error condition when the 'error' attribute is present and has the value zero. This element has an optional 'locale' attribute, of type `xs:language`, that identifies the locale for the text. If the 'locale' attribute is absent, then the text is for the default locale. The <Error> element is a single line plain text string whose contents is a local matter.

For example, if an error is not caused by one of the standard conditions, then the 'error' attribute has the value of zero and the <Error> elements can be used to provide the display text (the default locale in this example is "en").

```
<Real name="zone-temp" error="0" >
  <Error>The device is not feeling well today</Error>
  <Error locale="de">Es ist nicht heute gesund</Error>
</Real>
```

When no known error condition exists, the <Error> elements and the 'error' attribute shall be absent. The 'error' attribute and the <Error> elements are not inherited from a definition. However, the 'error' attribute and/or <Error> elements may be specified without specifying a value to indicate that a value inherited from a definition is no longer valid.

If an <Error> element is present in a context without the 'error' attribute, the 'error' attribute is implicitly assigned the value "0".

For example, the following shows that it is not necessary to include `error="0"` in a context with <Error> elements.

```
<Real name="aux-temp" >
  <Error>No aux device attached</Error>
</Real>
```

**X.3.8 Length Restrictions**

Primitive data elements that have variable length, <String>, and <OctetString>, can have their length restricted by optional attributes.

The attributes that are used for restricting the length of primitive data elements are specified in the following clauses.

The type and applicability of the range restriction attributes are summarized in Table X.x.

**Table X.x** Length Restriction Attributes

Data Element	Attribute Name	Attribute Type
<BitString>,	minimumLength maximumLength minimumLengthForWriting maximumLengthForWriting	xs:nonNegativeInteger
<OctetString>,	minimumLength maximumLength minimumLengthForWriting maximumLengthForWriting	xs:nonNegativeInteger
<String>	minimumLength maximumLength minimumLengthForWriting maximumLengthForWriting minimumEncodedLength maximumEncodedLength	xs:nonNegativeInteger

	minimumEncodedLengthForWriting maximumEncodedLengthForWriting	
--	--	--

**X.3.8.1 'minimumLength'**

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the length of the value. For the <String> element, this indicates the length, in characters, as represented in XML. For the <OctetString> element, this represents the length in octets of the underlying binary data, not its character representation in XML. For the <BitString> element, this represents the length in bits of the underlying binary data, not including any binary encoding overhead, and not its character representation in XML.

The default value of this attribute is undefined, meaning that the length of the value is unlimited or that the limit is unknown.

**X.3.8.2 'maxLength'**

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound on the length of the value. For the <String> element, this indicates the length, in characters, as represented in XML. For the <OctetString> element, this represents the length in octets of the underlying binary data, not its character representation in XML. For the <BitString> element, this represents the length in bits of the underlying binary data, not including any binary encoding overhead, and not its character representation in XML.

The default value of this attribute is undefined, meaning that the length of the value is unlimited or that the limit is unknown.

**X.3.8.3 'minimumLengthForWriting'**

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the length of the value when written. For the <String> element, this indicates the length, in characters, as represented in XML. For the <OctetString> element, this represents the length in octets of the underlying binary data, not its character representation in XML. For the <BitString> element, this represents the length in bits of the underlying binary data, not including any binary encoding overhead, and not its character representation in XML.

The default value of this attribute is the value of the 'minimumLength' attribute.

**X.3.8.4 'maxLengthForWriting'**

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound on the length of the value when written. For the <String> element, this indicates the length in characters, as represented in XML. For the <OctetString> element, this represents the length in octets of the underlying binary data, not its character representation in XML. For the <BitString> element, this represents the length in bits of the underlying binary data, not including any binary encoding overhead, and not its character representation in XML.

The default value of this attribute is the value of the 'maxLength' attribute.

**X.3.8.5 'minimumEncodedLength'**

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the length of the encoded value, in octets, when it is encoded for BACnet as described in Clause 20. This attribute is applicable only to the <String> element.

The default value of this attribute is undefined, meaning that the length of the value is unlimited or that the limit is unknown.

#### **X.3.8.6 'maximumEncodedLength'**

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound on the length of the encoded value, in octets, when it is encoded for BACnet as described in Clause 20. This attribute is applicable only to the <String> element.

The default value of this attribute is undefined, meaning that the length of the value is unlimited or that the limit is unknown.

#### **X.3.8.7 'minimumEncodedLengthForWriting'**

This optional attribute, of the type specified in Table X.x, provides the inclusive lower bound on the length of the encoded value, in octets, when it is encoded for writing with BACnet as described in Clause 20. This attribute is applicable only to the <String> element.

The default value of this attribute is the value of the 'minimumEncodedLength' attribute.

#### **X.3.8.8 'maximumEncodedLengthForWriting'**

This optional attribute, of the type specified in Table X.x, provides the inclusive upper bound for writing on the length of the encoded value, in octets, when it is encoded for writing with BACnet as described in Clause 20. This attribute is applicable only to the <String> element.

The default value of this attribute is the value of the 'maximumEncodedLength' attribute.

### **X.3.9 Collections**

Some constructed values are variable sized collections of elements of the same type. The three types of collections defined in this annex are <Array>, <List>, and <SequenceOf>. Structurally in XML, these three are identical. The difference between them is in their use for modeling BACnet data, where their names correspond to the types of access methods available through BACnet services.

All three types of collection have optional attributes that are specific to collections. These attributes can be applied to the <Array>, <List>, and <SequenceOf> elements.

If the type of the members is not a built in type or a previously defined type, an anonymous type can be declared using the <MemberTypeDefinition> child element instead of the 'memberType' attribute. The 'memberType' attribute cannot be used simultaneously with the <MemberTypeDefinition> child element.

#### **X.3.9.1 'minimumSize'**

This optional attribute, of type `xs:nonNegativeInteger`, indicates the minimum size that the collection is likely to be able to reach. Variable sized collections typically have zero as a minimum size, but fixed size collections do not. Fixed sized collections shall specify both 'minimumSize' and 'maximumSize' as the same value.

The default value of this attribute is "0".

### **X.3.9.2 'maximumSize'**

This optional attribute, of type `xs:nonNegativeInteger`, indicates the maximum size that the collection is likely to be able to reach. Fixed sized collections shall specify both 'minimumSize' and 'maximumSize' as the same value.

The default value of this attribute is undefined, implying an unlimited or unknown maximum size.

### **X.3.9.3 'memberType'**

This optional attribute, of type `xs:string`, indicates the name of the existing defined type that is to be used as the type of the members of the collection. This can be either the name of a type defined elsewhere in XML, or the name of one of the built-in types: "Any", "Array", "BitString", "Boolean", "Choice", "Date", "DatePattern", "DateTime", "Double", "Enumerated", "Integer", "List", "Null", "Object", "ObjectIdentifier", "OctetString", "Real", "Sequence", "SequenceOf", "Time", "TimePattern", "Unsigned", or "WeekNDay".

All members of the collection must be of the same type. If a collection of different types is desired, then a 'memberType' of "Any" can be used.

If the type of the members is not a built-in type or a previously defined type, an anonymous type can be declared using the `<MemberTypeDefinition>` child element instead of the 'memberType' attribute. The 'memberType' attribute cannot be used simultaneously with the `<MemberTypeDefinition>` child element.

The default value of this attribute is "Any", unless the `<MemberTypeDefinition>` element is present, in which case the value of this attribute is undefined.

An inherited 'memberType' attribute cannot be changed, and a subsequent `<MemberTypeDefinition>` element cannot override it. Therefore, once defined, the member type of a collection cannot change.

An example of the 'memberType' attribute is given in the description of the `<MemberType>` element.

### **X.3.9.4 <MemberTypeDefinition>**

This optional child element is used to provide an anonymous in-line definition for the type of the members of a collection. The `<MemberTypeDefinition>` element has a required single child element that defines the type for the members. The child element may use the 'extends' attribute to refer to another type that it is extending. The 'name' of the child element is ignored and the 'type' and 'overlay' attributes are not applicable.

This example shows the three kinds of member type definitions for three different `<SequenceOf>` elements. The `<SequenceOf>` named "listOfEventSummaries" defines an anonymous type for its members using the `<MemberTypeDefinition>` element, the `<SequenceOf>` named "eventTimeStamps" uses the 'memberType' attribute to refer to a previously defined type, and the `<SequenceOf>` named "eventPriorities" uses the 'memberType' attribute to refer to the built-in primitive type "Unsigned".

```
<Sequence name="0-GetEventInformation-ACK">
  <SequenceOf name="listOfEventSummaries" ...>
    <MemberTypeDefinition>
      <Sequence>
        ...
      </MemberTypeDefinition>
    </SequenceOf>
  <SequenceOf name="eventTimeStamps" memberType="0-BACnetTimeStamp" ... />
  ...
  <SequenceOf name="eventPriorities" memberType="Unsigned" ... />
</Sequence>
```

```
        </Sequence>
      </MemberTypeDefinition>
    </SequenceOf>
    <Boolean name="moreEvents".../>
  </Sequence>
```

An inherited `<MemberTypeDefinition>` element cannot be changed, and a subsequent 'memberType' attribute cannot override it. Therefore, once defined, the member type of a collection cannot change.

### **X.3.10 Representing Primitive Data**

Primitive data is represented by a single XML element and its associated metadata. The data elements available for modeling primitive BACnet data are: `<Any>`, `<BitString>`, `<Boolean>`, `<Date>`, `<Double>`, `<Enumerated>`, `<Integer>`, `<Null>`, `<ObjectIdentifier>`, `<OctetString>`, `<Real>`, `<Time>`, `<Unsigned>`, `<WeekNDay>`, `<WildDate>`, and `<WildTime>`. These are individual described more fully in the following clauses.

#### **X.3.10.1 <Null>**

The BACnet Null data is encoded with the XML element `<Null>`. Other than the common attributes and child elements described in Clauses X.3.1 and X.3.2, there are no other attributes or child elements for this element.

#### **X.3.10.2 <Boolean>**

BACnet Boolean data is encoded with the element `<Boolean>`. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the `<Boolean>` element can also have the value specifier described in Clause X.3.4, and the named values described in Clause X.3.3.

#### **X.3.10.3 <Unsigned>**

BACnet Unsigned Integer data is encoded with the element `<Unsigned>`. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the `<Unsigned>` element can also have the value specifier described in Clause X.3.4, the range restrictions described in Clause X.3.5, the named values described in Clause X.3.3, and the units specifier described in X.3.6.

#### **X.3.10.4 <Integer>**

BACnet Signed Integer data is encoded with the element `<Integer>`. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the `<Integer>` element can also have the value specifier described in Clause X.3.4, the range restrictions described in Clause X.3.5, the named values described in Clause X.3.3, and the units specifier described in X.3.6.

#### **X.3.10.5 <Real>**

BACnet Real data is encoded with the element `<Real>`. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the `<Real>` element can also have the value specifier described in Clause X.3.4, the range restrictions described in Clause X.3.5, the named values described in Clause X.3.3, and the units specifier described in X.3.6.

#### **X.3.10.6 <Double>**

BACnet Double data is encoded with the element `<Double>`. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the `<Double>` element can also have the value specifier described in Clause X.3.4, the range restrictions described in Clause X.3.5, the named values described in Clause X.3.3, and the units specifier described in X.3.6.

### **X.3.10.7 <OctetString>**

BACnet Octet String primitive data is encoded with the element <OctetString>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <OctetString> element can also have the value specifier described in Clause X.3.4, the length restrictions described in clause X.3.8, and the named values described in Clause X.3.3.

### **X.3.10.8 <String>**

BACnet Character String primitive data is encoded with the element <String>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <String> element can also have the value specifiers described in Clause X.3.4, the length restrictions described in clause X.3.8, and the named values described in Clause X.3.3.

### **X.3.10.9 <BitString>**

BACnet BitString primitive data is encoded with the XML element <BitString>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <BitString> element can also have the value specifiers described in Clause X.3.4, the length restrictions described in clause X.3.8, and the named values described in Clause X.3.3.

### **X.3.10.10 <Enumerated>**

BACnet Enumerated primitive data is encoded with the element <Enumerated>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Enumerated> element can also have the value specifier described in Clause X.3.4, the range restrictions described in Clause X.3.5, and the named values described in Clause X.3.3.

An extensible Enumerated data range may be defined with the range restrictions attributes. If neither 'minimum' nor 'maximum' are present, then the enumeration is not extensible and only the values specified by the named values are possible.

The value of an <Enumerated> element is an xs:string. This string is either a decimal formatted number, in the same form as xs:nonNegativeInteger, or a string that matches exactly the 'name' attribute of a child element of <NamedValues>.

For nonextensible Enumerations, if the number format is used, it shall match the value of one of the child elements of <NamedValue>. For extensible Enumerations, the numeric value is not restricted to match a child element of <NamedValues>, but its value may be restricted by the 'minimum' and 'maximum' attributes, if present.

### **X.3.10.11 <Date>**

BACnet Date data that represents either a single specific date or a wholly "unspecified" date is encoded with the element <Date>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Date> element can also have the value specifiers described in Clause X.3.4, the range restrictions described in Clause X.3.5, and the named values described in Clause X.3.3.

### **X.3.10.12 <DatePattern>**

BACnet Date data that is allowed to contain individually "unspecified" fields is encoded with the element <DatePattern>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <DatePattern> element can also have the value specifier described in Clause X.3.4 and the named values described in Clause X.3.3.

The value of an <DatePattern> element is an xs:string. The format of the string value is "YYYY-MM-DD" or "YYYY-MM-DD W", where:

YYYY is either a four-digit year or a single asterisk ("\*") character to indicate "unspecified",

MM is either a two-digit month or a single asterisk ("\*") character to indicate "unspecified",

DD is either a two-digit day of the month or a single asterisk ("\*") character to indicate "unspecified",

W is either the one-digit day of the week (1=Monday) or a single asterisk ("\*") character to indicate "unspecified".

The numeric fields shall have leading zeros to achieve the number of digits specified. The YYYY, MM and DD fields are separated by a single dash ("-") character and the optional W field is separated from the DD field by a single space character. If the W field is not present, then neither is the space separator.

The W field is required to be present if any of the YYYY, MM, or DD fields is "unspecified". It is allowed to be absent only if the YYYY, MM, and DD specify a single date and the W field can thus be calculated unambiguously. When a field is "unspecified", it is encoded for BACnet binary communications as the value 255.

#### **X.3.10.13 <Time>**

BACnet Time data that represents either a single specific time or a wholly "unspecified" time is encoded with the element <Time>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Time> element can also have the value specifiers described in Clause X.3.4, the range restrictions described in Clause X.3.5, and the named values described in Clause X.3.3.

#### **X.3.10.14 <TimePattern>**

BACnet Time data that is allowed to contain individually "unspecified" fields is encoded with the element <TimePattern>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <TimePattern> element can also have the value specifier described in Clause X.3.4 and the named values described in Clause X.3.3.

The value of an <TimePattern> element is an xs:string. The format of the string value is "HH-MM-SS.NN", where:

HH is either a two-digit hour or a single asterisk ("\*") character to indicate "unspecified",

MM is either a two-digit minute or a single asterisk ("\*") character to indicate "unspecified",

SS is either a two-digit second or a single asterisk ("\*") character to indicate "unspecified",

NN is either the two-digit hundredths or a single asterisk ("\*") character to indicate "unspecified".

The numeric fields shall have leading zeros to achieve the number of digits specified. The MM, HH, and SS fields are separated by a single colon (":") character and the NN field is separated from the SS field by a single period (".") character. When a field is "unspecified", it is encoded for BACnet binary communications as the value 255.

#### **X.3.10.15 <ObjectIdentifier>**

BACnet Object Identifier primitive data is encoded with the element <ObjectIdentifier>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <ObjectIdentifier> element can also have the value specifiers described in Clause X.3.4, the range restrictions described in Clause X.3.5, and the named values described in Clause X.3.3.

The value of an <ObjectIdentifier> elements is an xs:string. The format of this string is "TTT,NNN", where

TTT is either a type identifier or a single asterisk ("\*") character to indicate "unspecified", The type identifier is either a decimal number with no leading zeroes, or a standard type name exactly equal to the names specified in the definition for BACnetObjectTypesSupported in Clause 21.

NNN is either an instance number or a single asterisk ("\*") character to indicate "unspecified". The instance number is a decimal number with no leading zeroes.

When the type is "unspecified", it shall be encoded for BACnet binary communications as the value 1023. When the instance number is "unspecified", it shall be encoded for BACnet binary communications as the value 4194303.

### **X.3.10.16 <WeekNDay>**

BACnetWeekNDay primitive data is encoded with the element <WeekNDay>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <WeekNDay> element can also have the value specifier described in Clause X.3.4 and the named values described in Clause X.3.3.

The value of an <WeekNDay> elements is an xs:string. The format of the string value is "M,W,D", where:

M is either a decimal month identifier or an asterisk ("\*") character to indicate "unspecified",

W is either a decimal week identifier or an asterisk ("\*") character to indicate "unspecified",

D is either a decimal day-of-week identifier or an asterisk ("\*") character to indicate "unspecified".

The numeric fields do not have leading zeros. The M, W, and D fields are separated by a comma (",") character. The range and meaning of the numeric values for M, W and D is described in the BACnetWeekNDay production in Clause 21. When a field is "unspecified", it is encoded for BACnet binary communications as the value 255.

## **X.3.11 Representing Constructed Data**

Constructed data is represented by an XML element that contains one or more child elements that provide the value for the construct.

### **X.3.11.1 <Sequence>**

BACnet SEQUENCE constructed data is encoded with the element <Sequence>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Sequence> element can also have optional child elements representing the members of the sequence.

For modeling BACnet data, the allowed child elements of <Sequence> are: <Any>, <BitString>, <Boolean>, <Choice>, <Date>, <DateTime>, <DatePattern>, <Double>, <Enumerated>, <Integer>, <Null>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>. For modeling abstract data, <Sequence> additionally allows the child elements <Array>, <List>, and <Object>.

The 'name' attributes of the child element are significant and are used to match the child elements in an instance with a corresponding child element in a type definition. The names must be unique among sibling elements.

Named child elements provided in an instance must exist in the type definition and must be of the same element type, with the exception that the <Any> element in a definition can be replaced by any appropriate data element in an instance.

The order of definition of sequence members in XML is significant, as it is in Clause 21. New named elements added as part of a new definition using the 'extends' attribute are added to the end of the existing elements in the sequence. The order of sequence members in an instance is not significant, because the members are matched by their corresponding 'name' attribute and not by position.

### **X.3.11.2 <Choice>**

BACnet CHOICE constructed data is encoded with the element <Choice>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Choice> element can also have an optional <Choices> child element that defines the available choices and a single child data element holding the currently chosen data.

The single named child data element provides the value for the <Choice> and must exist in the <Choices> element and must be of the same element type, with the exception that the <Any> element in a definition can be replaced by any data element in an instance.

A single named child data element provided in a definition of a <Choice> can be used to provide a default value for the type. A child data element in an instance of that type replaces the default child data element from the definition since there can only be one chosen element at a time.

#### **X.3.11.2.1 <Choices>**

The list of possible choices for a <Choice> type is provided by the <Choices> element, which is an optional child element of <Choice>. All of the child elements of <Choices> shall have non-empty 'name' attributes with values unique among their sibling elements.

For modeling BACnet data, the allowed child elements of <Choices> are the data elements: <Any>, <BitString>, <Boolean>, <Choice>, <Date>, <DateTime>, <DatePattern>, <Double>, <Enumerated>, <Integer>, <Null>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>. For modeling abstract data, <Choices> additionally allows the child data elements <Array>, <List>, and <Object>.

The order of the definition of choice members in <Choices> is not significant.

The <Choices> element can only appear in a definition context, since adding new choices constitutes a structural change to the data. When inheriting a <Choices> element from a definition, the newly specified child elements are logically added to the list of existing child elements of the inherited <Choices> but in no prescribed order.

### **X.3.11.3 <SequenceOf>**

BACnet SEQUENCE OF constructed data is encoded with the element <SequenceOf>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <SequenceOf> element can also have the additional capabilities of Collections elements described in Clause X.3.9.

The child elements in a <SequenceOf> do not have 'name' attributes, and the order of the elements in an instance is significant.

Child elements provided in a type definition of a <SequenceOf> can be used to provide a default value for the type. However, any child elements in an instance of that type completely replace the default child elements since instance values of Collections are not merged with their definition.

### **X.3.11.4 <Array>**

The BACnetARRAY construct is encoded with the element <Array>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Array> element can also have the additional capabilities of Collections elements described in Clause X.3.9.

The child elements in a <Array> are not required to have 'name' attributes, and the order of the elements in an instance is significant. When 'name' attributes are not provided, the child elements specify the values for the array, in order, starting with array index 1.

Although the child elements of <Array> are not required to have 'name' attributes, when provided, the 'name' attribute indicates the indexed position in the array for which the child element is providing a value. This provides for a compact representation in XML where the majority of the array members are equal to their default values. Array positions that are not provided a value with an appropriately named child element retain their default value from their definition. If a 'name' attribute is provided, it shall be formatted as an xs:nonNegativeNumber, indicating the index position in the array. The first position in an array is index 1. If the 'name' attribute is omitted, then the child element is assigned to the next higher index in the array, starting with index 1.

Child elements provided in a type definition of a <Array> can be used to provide a default value for the type. However, any child elements in an instance of that type completely replace the default child elements since instance values of Collections are not merged with their definition.

#### **X.3.11.5 <List>**

The "List of" construct is encoded with the element <List>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <List> element can also have the additional capabilities of Collections elements described in Clause X.3.9.

The child elements in a <List> do not have 'name' attributes, and the order of the elements in an instance is not significant.

Child elements provided in a type definition of a <List> can be used to provide a default value for the type. However, any child elements in an instance of that type completely replace the default child elements since instance values of Collections are not merged with their definition.

### **X.3.12 Representing Data of Unknown Type**

Data whose type is not known or not restricted by a definition is represented by BACnet in ASN.1 as ABSTRACT-SYNTAX.&Type, and is represented in XML using the <Any> element.

#### **X.3.12.1 <Any>**

The BACnet ABSTRACT-SYNTAX.&Type place-holder is represented with the XML element <Any>. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Any> element can also have an attribute, 'allowedTypes', that defines which actual types are allowed to replace the <Any>. The <Any> element is only allowed in a definition context. Instances shall replace the <Any> with an actual primitive or constructed data type, subject to the 'allowedTypes' restrictions.

#### **X.3.12.2 'allowedTypes'**

This optional attribute, of type xs:string, indicates a list of types that are allowed to be substituted for an <Any> element. This attribute is only allowed on an <Any> element. The value of this attribute is equal to a comma-separated concatenation of the strings suitable for use as a 'type' attribute value.

The default value for this attribute is "" (empty string), which means that there are no restrictions on what datatypes can be substituted for the <Any>.

## **X.4 Expressing BACnet Objects and Properties in XML**

BACnet objects are represented in XML by the <Object> element. The properties of a BACnet object are expressed as child elements of the <Object> element and use the 'propertyIdentifier' attribute to specify the property identifier to use when accessing the property using BACnet binary services.

#### **X.4.1 <Object>**

BACnet Objects are represented by the <Object> element. In addition to the common attributes and child elements described in Clauses X.3.1 and X.3.2, the <Object> element can also have child elements representing the properties of the object.

The allowed child elements of <Object> are: <Any>, <Array>, <BitString>, <Boolean>, <Choice>, <Date>, <DateTime>, <DatePattern>, <Double>, <Enumerated>, <Integer>, <List>, <Null>, <ObjectIdentifier>, <OctetString>, <Real>, <Sequence>, <SequenceOf>, <String>, <Time>, <TimePattern>, <Unsigned>, and <WeekNDay>.

The 'name' attribute of the child elements is significant and is used to match a child element in an instance with a corresponding child element in a type definition. The names must be unique among sibling elements.

Named child elements provided in an instance must exist in the type definition and must be of the same element type, with the exception that the <Any> element in a definition can be replaced by any appropriate data element in an instance.

The order of the definition of child elements in <Object> is not significant. New named elements added as part of a new definition using the 'extends' attribute are added to the list of child elements in no prescribed order.

#### **X.5 Definitions, Types, Instances, and Inheritance**

This annex specifies not only an XML syntax, but also an underlying data model that is expressed by the XML. This data model has a type and instance system similar to many programming languages.

An element's "type" defines the set of attributes and child elements that it is allowed to have. Every element has a type, whether explicitly or implicitly specified.

A "definition" is a named element that can be referred to by another element by using the 'type', 'extends', or 'overlays' attribute. To alert the processor that a named type definition is being created, elements that are to be used as definitions are declared within a "definition context", which means that the element is a direct child element of a <Definitions> element. There are no limits on the number of definition contexts in an XML document. However, to simplify processing, there is a requirement that definitions be declared before they are used.

The terms "type" and "definition" are mostly synonymous and are often used interchangeably in this annex, but the term "definition" always refers to a referenceable element (a "typedef" in some languages), whereas the term "type" can also refer to anonymous types created inline within another definition and to the built-in types like "String".

An "instance" is an element that refers to a definition element using the 'type' attribute. If no 'type' attribute is provided, the element's definition is implicitly identified by the element's XML tag name. For example, <String name="foo"> is equivalent to <String name="foo" type="String">. So, every element has a definition even when the 'type' attribute is not explicitly given.

The syntax defined in this annex is used for both definitions and instances. The two contexts are mostly identical: definitions can have values, which can be considered their "default values", and instances can change previously defined metadata like 'maximum'. However, there are some restrictions that are noted in this clause and elsewhere. For examples, an instance cannot add new child elements to <NamedValues> or add new members to a <Sequence>. Those actions can only take place in a definition context.

Elements "inherit" from their definition by logically copying every attribute and child element of the definition into themselves and then adding or overlaying the attributes and child elements that are specified for the element itself. Most attributes and child elements are inherited without modification, but there are a few exceptions, such as the interactions between 'value' and <Value>. These exceptions are described in the individual clauses that define the attributes or child elements involved.

Even though some attributes, like 'requiredWith', are interpreted as a concatenated list of strings, newly specified attribute value are not merged with inherited values. Attribute values are replaced in their entirety when a new value is specified. Element-based lists, like <NamedValues> and <Choices>, however, are merged with their definitions, with new child elements being logically added to the list of existing child elements.

Because of this logical copying behavior, the term "inherit" is used in this annex to mean not only the process of adopting the existing members of a <Sequence> or <Object> when making an extension, as would be the common use of the term in Object Oriented languages, but also the process of receiving all the attributes and child elements logically from an element's definition. In this data model, even elements that represent "primitive" data, like <Unsigned> are actually composed of multiple parts, like 'maximum' and 'value', each of which would be modeled in Object Oriented languages as individual properties or member variables of an "Unsigned" class or type and which may have default values that were defined when they were declared or that were overridden by subsequent definitions or constructors. Those properties or member variables are all logically part of any instance of that type and retain ("inherit" in this annex) their default values unless overridden by the instance. This XML syntax and data model is designed to support that expected behavior.

For example, given this definition for a Real element named "0-Percent":

```
<Definitions>
  <Real name="0-Percent" value="50" minimum="0" maximum="100" units="percent" />
</Definitions>
```

Consider the following two other definitions.

```
<Definitions>
  <Real name="0-LimitedPercent1" type="0-Percent" minimum="10" maximum="90"/>
</Definitions>
```

```
<Definitions>
  <Real name="0-LimitedPercent2" value="50" minimum="10" maximum="90" units="percent"/>
</Definitions>
```

The types defined by "0-LimitedPercent1" and "0-LimitedPercent2" are logically equivalent because 0-LimitedPercent1 inherited the 'value' and 'units' attributes from its definition, "0-Percent", and overrode the 'minimum' and 'maximum' attributes to new values, while the "0-LimitedPercent2" specifies all attributes itself without the use of a previous definition.

The above example also shows that a definition can specify any attributes that are allowed by its type, including 'value'. So consider some instances of "0-Percent":

```
<Real type="0-Percent" />

<Real type="0-Percent" value="50" />

<Real type="0-Percent" value="25" />

<Real type="0-Percent" value="25" maximum="50" />
```

The first two instances are identical because the all of the attributes of the definition "0-Percent" are inherited by all instances, making the value="50" in the second instance redundant. However, the third instance changes the value and so the 'value' attribute is required. The fourth instance shows not only that values can be changed in instances but that any non-structural metadata can be changed as well. The meaning of "nonstructural" is defined in the description of the 'type' attribute.

Therefore, those four instances of "0-Percent" are logically equivalent to these four elements, respectively.

```
<Real value="50" minimum="0" maximum="100" units="percent" />
<Real value="50" minimum="0" maximum="100" units="percent" />
<Real value="25" minimum="0" maximum="100" units="percent" />
<Real value="25" minimum="0" maximum="50" units="percent" />
```

The copying behavior of the definition is cascaded as needed until elements with inherent definitions are reached. If, while copying a set of child elements, one of the child elements itself has a 'type' or 'extends' attribute, before that child element's own attributes and child elements are considered, the contents of its definition are logically copied into it.

For example, given these three definitions:

```
<Definitions>
  <Unsigned name="0-UnlimitedPercent" units="percent" />
  <Unsigned name="0-NormalPercent" type="0-UnlimitedPercent" maximum="100"/>
  <Unsigned name="0-LimitedPercent" type="0-NormalPercent" minimum="10" maximum="90"/>
</Definitions>
```

These two instances are logically equivalent:

```
<Unsigned type="0-LimitedPercent" value="75"/>
<Unsigned value="75" minimum="10" maximum="90" units="percent"/>
```

So far, these examples have been making new definitions by making nonstructural changes to existing definitions. In these cases, the 'type' attribute was used within the definition context, rather than 'extends'. This is because an action like specifying maximum="100" in the definition for "0-NormalPercent" above was not changing the data model. The 'maximum' attribute is always part of the data model for the <Unsigned> element, so this is a nonstructural change.

When structural changes are needed, the 'extends' attribute is used instead. This alerts the processor that changes to the data model are allowed and, typically, that new child elements of a <Sequence>, <Object>, <Choices>, or <NamedValues> element are being added.

For example, consider this definition for the type named "0-base".

```
<Definitions>
  <Sequence name="0-base">
    <Real name="foo"/>
  </Sequence>
</Definitions>
```

The following extension creates a new definition for a type named "0-derived", which is based on "0-base".

```
<Definitions>
  <Sequence name="0-derived" extends="0-base">
    <Real name="bar"/>
  </Sequence>
</Definitions>
```

An instance of "0-derived" thus contains the members defined in "0-base" as well as those added in "0-derived".

```
<Sequence type="0-derived">
  <Real name="foo" value="1.0"/>
  <Real name="bar" value="2.0"/>
</Sequence>
```

The 'extends' attribute is only used in the definition context, but is not limited to the outermost element that is defining the new type. When used on an inner element, a new anonymous type is created as an extension of the referenced type. Thus, anonymous types are either fully defined in-line without the use of the 'extends' attribute, or are an extension of an existing type by using the 'extends' attribute.

The following example shows all four methods of assigning a type for a new member. The 'simple-member' uses a built-in type with no need for the 'type' or 'extends' attributes. The "typed-member" refers to the "0-derived" type using the 'type' attribute. The "full-anonymous-type-member" fully defines its anonymous type in-line, also without the use of the 'type' or 'extends' attributes. The "extension-anonymous-type-member" extends an existing type using the 'extends' attribute.

```
<Definitions>
  <Sequence name="0-example-1">
    <Real name="simple-member"/>
    <Sequence name="typed-member" type="0-derived"/>
    <Sequence name="full-anonymous-type-member">
      <Real name="foo" />
      <Real name="bar" />
    </Sequence>
    <Sequence name="extension-anonymous-type-member" extends="0-base">
      <Real name="bar" />
    </Sequence>
  </Sequence>
</Definitions>
```

The result is that the last three members defined above all have child members named "foo" and "bar". The first three methods are all used in Clause 21 data structures; the fourth is a capability only of this XML syntax, since no Clause 21 data structure is an extension of another.

An instance of that sequence, providing a value for each member, would look like this.

```
<Sequence type="0-example-1">
  <Real name="simple-member" value="55"/>
  <Sequence name="typed-member"
    <Real name="foo" value="1"/>
    <Real name="bar" value="2" />
  </Sequence>
  <Sequence name="full-anonymous-type-member">
    <Real name="foo" value="1"/>
    <Real name="bar" value="2" />
  </Sequence>
  <Sequence name="extension-anonymous-type-member">
    <Real name="foo" value="1"/>
    <Real name="bar" value="2" />
  </Sequence>
</Sequence>
```

The 'type' attribute on an element is required only where it cannot be determined from the context in which the element appears. In the above example, the 'type' attribute for the three structured members of "0-example-1" was not given, because child elements are always matched by name with their corresponding child element in the definition of their parent element. This matching continues up the ancestor chain until the context cannot be determined, at which point a 'type', 'extends', or 'overlays' attribute shall be present to provide a context for all the descendant elements. The 'type' attribute is not disallowed in contexts where it can be inferred, but it is not required, and should be left off where brevity of XML is desirable.

In this example, the 'type' attribute is required to define the type for the member "propertyIdentifier".

```
<Definitions>
  <Sequence name="0-BACnetPropertyReference">
    <Enumerated name="propertyIdentifier" contextTag="0" type="0-BACnetPropertyIdentifier"/>
    <Unsigned name="propertyArrayIndex" contextTag="1" optional="true" />
  </Sequence>
</Definitions>
```

In an XML representation of a value of that member, the use of the 'type' attribute on the outer element sets the context for interpretation of the inner element; therefore, the 'type' attribute is not needed on the "propertyIdentifier" member because it is known from the definition of 0-BACnetPropertyReference.

```
<Sequence type="0-BACnetPropertyReference" >
  <Enumerated name="propertyIdentifier" value="present-value" />
</Sequence>
```

The use of the 'type' attribute indicates that an element is an instance of a previously defined type definition and that its attributes and child elements do not cause any structural changes. Conversely, the use of the 'extends' attribute indicates that structural changes are allowed and expected. Consequently, the 'type' attribute can be used in any context, but the 'extends' attribute can only be used in a definition context where a new type is being created.

A "structural change" is defined as one that adds a new member to a <Sequence> or <Object>, adds a new choice to a <Choice>, changes the member type of an <Array>, <List> or <SequenceOf>, adds any new named values, or changes the 'optional', 'absent', or 'contextTag' attributes. Changes other than this are considered nonstructural. Typically, non-structural changes are limited to assigning a value for the data, but in some cases, other nonstructural metadata may be changed as well. See Clause X.x for more on contexts, and see Clause X.x for more on types and prototypes.

For an example of a nonstructural change, consider the definition:

```
<Definitions>
  <Sequence name="0-base">
    <Real name="foo"/>
  </Sequence>
</Definitions>
```

A new definition can be made from that without making structural changes to "999-base" by using the 'type' attribute in a definition context. Below, the existing member is only given new metadata; in this case, new limits.

```
<Definitions>
  <Sequence name="0-limited-base" type="0-base">
    <Real name="foo" minimum="0.0" maximum="100.0" />
  </Sequence>
</Definitions>
```

If, however, a structure change is needed, the 'extends' attribute is used instead of the 'type' attribute. Below, the derived type adds a new member.

```
<Definitions>
  <Sequence name="0-limited-base" extends="0-base">
    <Real name="bar" />
  </Sequence>
</Definitions>
```

Inheriting <NamedValues> is only allowed in a definition context and involves overlaying existing child elements and adding new ones to the end.

For example, this enumeration definition creates an enumeration where red=0, green=1, and blue=6.

```
<Definitions>
  <Enumerated name="0-base-enum">
    <NamedValues>
      <Unsigned name="red" />
      <Unsigned name="green" />
      <Unsigned name="blue" value="6"/>
    </NamedValues>
  </Enumerated>
</Definitions>
```

An extension to that enumeration adds `displayName` attributes to the existing red, green, and blue named values and adds new named values for purple and yellow. The order of existing named values is deliberately skewed in this example to illustrate that it does not matter since they have already been assigned values, but the order of the newly added purple and yellow is significant.

```
<Definitions>
  <Enumerated name="0-extended-enum" extends="0-base-enum">
    <NamedValues>
      <Unsigned name="green" displayName="Green"/>
      <Unsigned name="purple" displayName="Purple"/>
      <Unsigned name="blue" displayName="Blue"/>
      <Unsigned name="yellow" displayName="Yellow"/>
      <Unsigned name="red" displayName="Red"/>
    </NamedValues>
  </Enumerated>
</Definitions>
```

The above extension logically has an ordered list of named values.

```
<NamedValues>
  <Unsigned name="red" displayName="Red" />
  <Unsigned name="green" displayName="Green" />
  <Unsigned name="blue" displayName="Blue" value="6"/>
  <Unsigned name="purple" displayName="Purple" />
  <Unsigned name="yellow" displayName="Yellow" />
</NamedValues>
```

This ordering means that the automatically assigned values for purple and yellow will be 7 and 8, respectively. Since this was not obvious from the definition of "0-extended-enum", enumerations should explicitly assign values when those enumerations are mapped to external data or where the numerical values are otherwise significant outside of XML.

Inheriting `<Choices>` is only allowed in a definition context and involves overlaying existing child elements and adding new ones to the list of choices (order is not significant).

For example, this choice definition creates two choices and defines the default value of the choice itself to be the "joe" choice.

```
<Definitions>
  <Choice name="0-base-choice">
    <Choices>
      <Unsigned name="fred" displayName="Fred"/>
      <Real name="joe" displayName="Joe"/>
    </Choices>
  </Choice>
</Definitions>
```

```
<Real name="joe"/>
</Choice>
</Definitions>
```

An extension to that choice changes a `displayName` of the existing choices and adds a new "bob" choice. Additionally, it changes the default value of the choice itself to be "bob" rather than the default value for "0-base-choice", which was "joe". The order of existing choices is deliberately skewed in this example to illustrate that it does not matter, and the order of the resulting list of choices is not significant either.

```
<Definitions>
  <Choice name="0-extended-choice" extends="0-base-choice">
    <Choices>
      <Real name="joe" displayName="Joseph"/>
      <Double name="bob" displayName="Robert"/>
      <Unsigned name="fred" displayName="Frederick"/>
    </Choices>
    <Real name="bob"/>
  </Choice>
</Definitions>
```

## X.6 Encoding and Access Rules

The binary encoding of the data elements defined in this annex is implied by the element's name and matches expected encoding for the like-named structures and primitives defined in Clauses 20 and 21. The 'contextTag' attribute provides the context tag to use when required and its absence implies that the appropriate application tag shall be used instead.

The accessibility of the data using BACnet services is also implied by the element's name and the 'propertyIdentifier' attribute. When used as child elements of an <Object>, the <List> and <Array> elements imply the appropriate behavior for BACnet "List of" and "BACnetARRAY of" properties when accessed using BACnet binary services. For all child elements of an <Object>, the 'propertyIdentifier' attribute provides the property number that can be used to access the data with BACnet binary services.

## X.7 Extensibility

Both the XML syntax and the data it represents can be extended.

### X.7.1.1 XML extensions

Documents conforming to this standard can be extended through the use of XML attributes and elements from other XML namespaces. XML attributes from other namespaces are allowed on any standard element, and elements from other namespaces are allowed under any standard element that already has child elements defined for it in this standard. With the exception of the <Documentation> element, this standard does not use mixed content, so any element other than <Documentation> that uses body text may not be extended with elements from other namespaces.

### X.7.1.2 Data Model Extensions

Extensions to the data represented by this standard XML syntax is accomplished with the <Extensions> element defined in Clause X.3.2.6. Normally, these extensions represent data that is beyond what is accessible through standard BACnet binary services but which may be of interest to the consumer of the XML, or they may represent extended data that is accessible through BACnet Web services or by other means.

Standard elements and attributes can be both extended with proprietary attributes. The names of proprietary attributes shall begin with a period character (".") to prevent conflict with standard attribute names. While not required, it is recommended that proprietary attributes also use a vendor specific prefix, following the required period character, to prevent conflicts among proprietary attributes.

Standard attributes, like displayName, can be extended with standard attributes that are appropriate to its datatype. While this clause provides the syntax and method for extending the standard attributes, it makes no requirement that consumers of this XML understand or process any of these extensions. When extending standard attributes, the names used for the extensions use the naming convention of the corresponding attributes in the Annex N data model and are shown in the following table. The table also indicates an "effective type" which defines the standard element type that shall be used as the child of <Extensions> when extending the standard attribute.

**Table X.x** Standard Attribute Extensibility

Attribute Name	Extension Name	Effective Type
type	n/a <sup>1</sup>	n/a <sup>1</sup>
extends	n/a <sup>1</sup>	n/a <sup>1</sup>
overlays	n/a <sup>1</sup>	n/a <sup>1</sup>
displayName	"DisplayName"	<String>
displayNameForWriting	"DisplayNameForWriting"	<String>
description	"Description"	<String>
writable	"Writable"	<Boolean>
readable	"Readable"	<Boolean>
commandable	"Commandable"	<Boolean>
associatedWith	"AssociatedWith"	<String>
requiredWith	"RequiredWith"	<String>
requiredWithout	"RequiredWithout"	<String>
notPresentWith	"NotPresentWith"	<String>
writableWhen	"WritableWhen"	<String>
requiredWhen	"RequiredWhen"	<String>
writeEffective	"WriteEffective"	<Enumerated>
optional	"Optional"	<Boolean>
absent	"Absent"	<Boolean>
variability	"Variability"	<Enumerated>
volatility	"Volatility"	<Enumerated>
contextTag	"ContextTag"	<Unsigned>
propertyIdentifier	"PropertyIdentifier"	<Unsigned>
notForWriting	"NotForWriting"	<Boolean>
notForReading	"NotForReading"	<Boolean>
minimum	"Minimum"	varies <sup>2</sup>
maximum	"Maximum"	varies <sup>2</sup>
minimumForWriting	"MinimumForWriting"	varies <sup>2</sup>
maximumForWriting	"MaximumForWriting"	varies <sup>2</sup>
resolution	"Resolution"	varies <sup>2</sup>
minimumLength	"MinimumLength"	<Unsigned>
maximumLength	"MaximumLength"	<Unsigned>
minimumLengthForWriting	"MinimumLengthForWriting"	<Unsigned>
maximumLengthForWriting	"MaximumLengthForWriting"	<Unsigned>
minimumEncodedLength	"MinimumEncodedLength"	<Unsigned>
maximumEncodedLength	"MaximumEncodedLength"	<Unsigned>
minimumEncodedLengthForWriting	"MinimumEncodedLengthForWriting"	<Unsigned>
maximumEncodedLengthForWriting	"MaximumEncodedLengthForWriting"	<Unsigned>
minimumSize	"MinimumSize"	<Unsigned>
maximumSize	"MaximumSize"	<Unsigned>
memberType	"MemberType"	<String>

units	"Units"	<Enumerated>
value	n/a <sup>1</sup>	n/a <sup>1</sup>
charset	n/a <sup>1</sup>	n/a <sup>1</sup>
codepage	n/a <sup>1</sup>	n/a <sup>1</sup>
length	n/a <sup>1</sup>	n/a <sup>1</sup>
error	n/a <sup>1</sup>	n/a <sup>1</sup>
locale	n/a <sup>1</sup>	n/a <sup>1</sup>

<sup>1</sup>The attributes marked as n/a are not extensible because they are not "metadata". They are either used to define the data type or they are an indivisible part of the data value.

<sup>2</sup>The effective type of the range restriction attributes is based on the enclosing element. The effective type is <Unsigned> for the enclosing types <Enumerated> and <ObjectIdentifier>, and is equal to the enclosing type for all others.

The following example shows the standard attribute, 'maximumLength', being extended with the standard attributes 'writable' and 'writeEffective', and the standard element <String> being extended with a proprietary attribute ".999-WritePrivilegeLevel".

```

<Definitions>
  < Object name="999-ExampleObject">
    <String name="write-me" writable="true" maximumLength="50" >
      <Extensions>
        <Unsigned name="MaximumLength" writable="true" writeEffective="on-device-restart" />
        <String name=".999-WritePrivilegeLevel" value="6" />
      </Extensions>
    </Real>
  </ Object >
</Definitions>

```