BSR/ASHRAE Addendum ap to
ANSI/ASHRAE Standard 135-2010

Advisory Public Review Draft

Proposed Addendum ap to Standard 135-2010, BACnet® - A Data Communication Protocol for Building Automation and Control Networks

First Advisory Public Review (March 2012)
(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 standard, go to the ASHRAE website at www.ashrae.org/standards-research--technology/public-review-drafts and access the online comment database. The draft is subject to modification until it is approved for publication by the Board of Directors and ANSI. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE website) remains in effect. The current edition of any standard may be purchased from the ASHRAE Online Store at www.ashrae.org/bookstore or by calling 404-636-8400 or 1-800-727-4723 (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 website, 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.

ASHRAE, 1791 Tullie Circle, NE, Atlanta GA 30329-2305

[This foreword and the "rationales" on the following pages 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 advisory 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.

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2010 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 at 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.

135-2010*am*-1 Define Application Interfaces

Rationale

Clause 12 defined a number of useful common abstractions for the building blocks of modern control systems. However, this leaves an unaddressed concept of "application" that sits above this building block level. A high level application, like "Air Handler" may be composed of many of the Clause 12 objects, and may also itself be composed of other applications like variable frequency drives for its fans.

The Structured View object was created to address this need for a higher level composition of objects. However, there was no mechanism provided to express conformance to a "standard view" and no means to define such a standard view. This proposal addresses both those needs as well as adding more flexibility to the Structured View to address varied deployment scenarios.

In this document, an application interface is a definition of the characteristics of a BACnet interface for a particular type of application or equipment function and its interoperability requirements. Certain types of specialized mechanical equipment that have similar characteristics across a range of vendors and implementations, such as variable frequency drives and thermostats, could be good candidates for the development of standard application interfaces.

A single Clause 12 object is not always the best way to present an application interface. Some vendor implementations do not rely on single objects to represent the aggregate data points required for the particular application; instead those implementations use some combination of standard and/or non-standard object types to represent the collection of values that constitute the application interface. Some application interfaces, especially those that represent complex devices or sub-systems, might even span multiple BACnet devices.

Regardless of which deployment method is followed in order to represent the data points, it is still necessary to describe the functional requirements of the application in terms of a set of data values and their characteristics and behaviors. To simplify the process of developing application interfaces, it is desirable to abstract the functional specification and table of data values into individually publishable documents.

The solution outlined here, and in other sections of this addendum, provides:
- a human readable (document) format for the definition of interfaces
- a machine readable (XML) format for the definition of interfaces
- a BACnet protocol discoverable method of representing an instance of an interface (Structured View)
- a new service for efficient traversal of interface instances (ReadPropertyIndirect)

[Add to **Clause 3.2**, p. 2]

application: a collection of logical functionality within a system. A particular application may be contained or realized within a single device or may involve multiple devices acting together. Examples of applications include, but are not limited to: air handlers, variable frequency drives, rooftop units, special mechanical equipment, stairwell pressurization, room lighting, zone and building access control, and energy management.

application interface: a mechanism for defining the BACnet-visible characteristics of particular applications, and their description in both normative and informative terms. That functionality includes:

(a) a standard name that uniquely identifies the application interface;
(b) the version of the application interface;
(c) a description of an object model for realizing that functionality including expected ranges and datatypes of parametric values provided to or by the interface;
(d) a description of the functional behavior provided by the interface;
(e) supportive informative diagrams, tables, and background text.

[Add new **Clause 26**]

26 APPLICATION INTERFACES

Applications are collections of logical functionality within a system. A particular application may be contained or realized within a single device or may involve multiple devices acting together. An application may represent a program that implements a particular strategy, for example, load control, energy management, or control of a specific kind of device. Applications may also represent simple or complex physical equipment. As an abstract concept, applications can be standardized just as information objects can be in BACnet.

BACnet's standardized set of data structures, called objects, provide an interface for sharing information and control within and between building systems. BACnet objects may be used to exchange information through both standard object types (Clause 12) and non-standardized proprietary object types. However, it is common for many kinds of building system devices to contain many individual operating parameters and other kinds of information or control and configuration adjustments. As a result, the complete functionality of a given building system device may require many separate object property values. Individual implementations may choose to aggregate similar functionality in different groupings or object models.

An Application Interface is a standard mechanism in BACnet for defining the characteristics of particular applications and their description in both normative and informative terms. That functionality includes:

(a) a standard name for an Application Interface or functionality set;
(b) the version of the application interface;
(c) a description of an object model for realizing the functionality of the application interface including expected ranges and datatypes of parametric values provided to or by the interface;
(d) a description of the functional behavior provided;
(e) supporting informative diagrams, tables, and background text.

The functionality of a given application interface may include the use of standard Clause 12 object types applied in specific ways, or it may be that no specific consensus exists regarding a single object model. If no single object is appropriate, the application interface provides a means of indirectly documenting a collection of related object property values that are appropriate to the application. These aggregated values may or may not exist in a single Clause 12 object in a given application, and the application interface provides a mechanism for discovering the binding of a particular value in the context of that application at runtime.

When there is consensus about the general behavior of an application functionality set, but no consensus on a specific object model, the deferred binding mechanism provides a method for standardizing those parts of the application that can be agreed upon, and more flexibility in representing the object model parameters. The adoption of a more abstract standard for a given application may spur industry support for, and more common use of, that application interface standard which may in turn lead to eventual consensus about the object model, which may lead to proposals to create new Clause 12 object types that capture the defacto common usage of a more indirect application interface normative specification.

As a result, Clause 12 objects have a purpose that is independent of application interfaces and in no way mutually exclusive with them.

An application interface may not require an object model at all, and may simply describe a behavior or procedure that should be implemented or followed in a consistent manner by applications that want to implement that standard interface. The interface may define, for example, the use of specific BACnet services, or common data file formats, or procedures. In short, an application interface may define any common collection of functionality that is used to achieve a particular purpose so as to standardize the way that two or more BACnet devices may interact. Since application interfaces are application-driven instead of communication-driven, they provide another dimension for standardization and interoperability.

26.1 Application Interface Catalog

All standard Application Interfaces shall be documented in Annex B using the format described in this clause.

26.2 Application Interface Name

Each application interface shall be associated with a standardized formal name. To ensure uniqueness, an Application Interface Name shall begin with a vendor identification code (see Clause 23) in base-10 integer format, followed by a

hyphen. All subsequent characters are administered by the organization registered with that vendor identification code. The vendor identification code that prefixes the interface name shall indicate the organization that publishes and maintains the interface definition. This vendor identification code need not have any relationship to the Vendor_Identifier of the device that implements this application interface.

For example, a "Variable Frequency Drive" Application Interface standardized by ASHRAE might be named "0-VFD," and the second version of a "Rooftop Unit" Application Interface defined by the organization that has vendor identification code 555 might be named "555-Rooftop Unit.2."

26.3 Application Interface Purpose

Each application interface shall include a descriptive summary of the purpose or functionality provided by that interface. This may include diagrams or an overview when appropriate.

26.4 Interface Table

Each application interface that makes use of an object model shall define one Interface Table that lists one or more information values (data points) that represent inputs to the application interface (for control or configuration) or outputs from the interface (results and status). The binding of a given instance of an application interface to specific object properties that provide the interface table values will generally occur at runtime.

The following information shall be provided for each value in an Interface Table:

(a) numeric ID (required)
(b) name for the value (required)
(c) datatype for the value (required)
(d) conformance code (required)
(e) range restrictions
(f) engineering units (if applicable)
(g) volatility
(h) usage

An example of an Interface Table for a hypothetical pump controller with an optional power meter interface is shown in Table 26-1. See below for a description of each column of the table.

Table 26-1. Example Interface Table

ID	Name	Datatype	Conf. Code	Range/Units	Vol	Usage
1	Enabled Monitor	BINARY	R	1 = "Enabled", 0 = "Disabled"	V	Status
2	Fault Monitor	BINARY	R	1 = "Fault", 0 = "No Fault"	V	Status
3	Flow Rate GPM	FLOAT	C	GPM	V	Status
4	Flow Rate LPM	FLOAT	C	LPM	V	Status
5	Running Seconds	UNSIGNED	OW	Seconds	N	Status
6	Enable-Disable Command	BINARY	RP	1 = "Enable", 0 = "Disable"	V	Control
7	Reset Request	BINARY	RW	Off (0) to On (1) transition initiates reset	V	Self-Clearing
8	Most Recent Fault Code	Unsigned16	R	(0-65535)	V	Status
9	Flow Rate Setpoint GPM	FLOAT	CP	GPM	V	Control
10	Flow Rate Setpoint LPM	FLOAT	CP	LPM	V	Control
11	Power Meter	Interface	O	555-POWER		Status

Notes for Table 26-1:
1. At least one of the values with IDs 3 and 4 must be implemented.
2. Exactly one of the values with IDs 9 and 10 must be implemented.

26.4.1 ID

Each interface table value shall be assigned a positive integer that uniquely identifies the value. Values shall be listed in the table in numeric order, beginning with 1.

An important application of the numeric ID is to be the index of an array that contains a run-time reference to the BACnet property or array element that contains the value in a real device. See Clause 26.6.1.

26.4.2 Name

Each interface table value shall be given a descriptive name that is unique within the interface and that describes its function, e.g., "Reset Request." The name may be translated or otherwise localized as necessary.

26.4.3 Datatype

Each interface table value shall specify the type of data that the value must provide. The following datatypes are allowed:

(a) Any Clause 21 Application Type (a.k.a. primitive type), except NULL
(b) Any Clause 21 Base Type (a.k.a. constructed type)
(c) A flexible type, as described below
(d) An Application Interface, indicated by "Interface"
(e) A one-dimensional array of any permitted Clause 21 datatype or flexible type, indicated by "Array of XXX"
(f) A list of any permitted Clause 21 datatype or flexible type, indicated by "List of XXX"

Flexible types are intended to allow implementers some flexibility in choosing a datatype to represent a value, although this puts some additional burden on consumers of an Application Interface. If the Interface Table specifies a flexible type for a value, a device implementing the value may choose any of the permitted implementation datatypes, as long as the chosen datatype is capable of meeting all of the other requirements of the value (range, precision, etc.). The flexible types are described in Table 26-2.

Table 26-2. Flexible Types

Flex. Type	Range of Values	Permitted Implementation Datatypes
BINARY	0 or 1	BACnetBinaryPV, Signed Integer, or Unsigned Integer
BOOL	FALSE or TRUE	Boolean (recommended); alternatively 0 (representing FALSE) or a non-zero value (representing TRUE), returned as Unsigned Integer or Signed Integer
ENUM	1, 2, 3, 4…	Enumerated, Unsigned Integer, or Signed Integer
FLOAT	Any real number	Real (recommended) or Double; Signed Integer or Unsigned Integer may be used in some cases
SIGNED	Integer	Signed Integer (recommended); alternatively Real or Double
UNSIGNED	Non-negative integer	Unsigned Integer (recommended); alternatively Signed Integer, Real, or Double

Note that a value with the ENUM type may not take on the value 0 (zero), whereas BACnet's Enumerated datatype permits the use of 0 (zero).

Although double-precision floating point numbers ("Double") are permitted for the implementation of some flexible types, they should only be used in implementations if a single-precision floating point number ("Real") lacks the required range or precision.

26.4.4 Conformance Code

Each Interface Table row shall include a conformance code constructed using one or more of the component codes listed in Table 26-3. The conformance code specifies certain minimum requirements for the implementation of a value.

Table 26-3. Conformance Code Components

Code	Definition
R	Required
O	Optional
C	Conditionally required
W	Writable
P	Writable with Priority
D	Deprecated (not present)

Every conformance code contains one of the following component codes: “R”, “O”, “C”, or “D”. The least constraining conformance code is "O", meaning that the value is optional in an implementation of the Application Interface. Conditional values (code “C”) are not always present but are required to be present if certain specified conditions are met. Required values (code “R”) shall be present in every implementation. Deprecated values (code “D”) shall not be present in a conforming implementation of the Application Interface; those values were defined in a previous version of the Application Interface.

A conformance code may also contain either “W” or “P” indicating that the value shall be writable if present in an implementation of the Application Interface. Writable values (code “W” or “P”) shall be implemented using properties that allow modification using the WriteProperty service, and also allow modification using the WritePropertyMultiple service if the device supports the execution of the WritePropertyMultiple service. Values that are writable with priority (code “P”) shall be implemented using a commandable property (see Clause 19.2).

26.4.5 Range and Precision Requirements

It is recommended that the range of permissible values shall be specified for each Interface Table value.
For enumerated values, specify each value that may be used, e.g., 1="Off", 2="On", 3="Auto". A separate table should be used if there are more than a few possible values.

For values that have a numeric datatype, in some cases it may be useful to also specify the minimum precision required by the application.

26.4.6 Engineering Units

Each Interface Table value shall specify at most one permitted engineering unit. Only the units corresponding to those listed in the BACnetEngineeringUnits production in Clause 21 may be used.

In cases in which it is desirable to have more than one permitted engineering unit, define one Interface Table value for each permitted engineering unit, and require that exactly one be present in an implementation.

26.4.7 Volatility

Each Interface Table value may have a specified volatility. “N” shall specify a non-volatile value that should be unchanged by a soft reset or power cycle of the device that contains the value. “V” shall specify a volatile value that is not expected to be retained across a device reset or power cycle.

26.4.8 Usage

Each interface table value may specify whether the value is a result produced by the application (Status), an input to the application (Control), a trigger input to the application (Self-Clearing), or an application configuration parameter (Configuration).

Status values are typically read only. Control values are expected to be written at whatever frequency is needed to control the device, and they are often implemented by a commandable property. Self-clearing values are typically intended to allow an action to be initiated. Configuration values are not intended to be frequently changed, and individual devices may require restarting before the new configuration takes effect or may restrict how often the configuration is changed.

26.5 Application Interface Description

Whether the application interface includes an Interface Table or not, it shall include a detailed description of the functional behavior of what is expected of the application. This description may include supportive diagrams, tables, and narrative as necessary.

26.6 Implementation of an Application Interface in a Device or System

An Application Interface will typically be entirely implemented within a single BACnet device, but in some cases, its functionality may be spread among multiple BACnet devices.

If the Application Interface contains an Interface Table, each implemented value shall conform to all of the requirements listed in the Interface Table and the accompanying descriptive text. Each value defined by the application interface that is present within an implementation is typically implemented by a single BACnet property, but a value that is not an array or a list may be implemented by a single element of an array contained in a BACnet property. Values that are specified to be arrays shall be implemented by a BACnet property containing an array of the specified datatype. Values that are specified to be lists shall be implemented by a BACnet property containing a list of the specified datatype.

A value that is defined to be an Application Interface will be implemented as specified in the definition of that Application Interface.

All implemented values that are contained in BACnet devices shall be readable using the ReadProperty service and also readable using the ReadPropertyMultiple service if the device supports the execution of the ReadPropertyMultiple service. Double-precision floating point numbers should be used in implementations only when required by the application.

26.6.1 Using Structured View Objects with Application Interfaces

It is recommended that each implementation of an Application Interface that defines an Interface Table include a corresponding Structured View object.

The primary purpose of the Structured View object is to provide references to the objects, properties, and other data sources that implement the values in the Interface Table. The ID of a value in the Interface Table specifies the index of the array element within the Structured View object's Subordinate_List property that contains the reference corresponding to that table value. If a value is specified to be an Application Interface in the Interface Table, then the corresponding reference within the Subordinate_List property should point to another Structured View object if the reference is initialized.

If the Structured View object contains an Interface_Name property, then the value shall contain the Application Interface Name of the corresponding Application Interface. See Clause 26.2.

26.7 Recommendations for Application Interface Design and Maintenance

Organizations that have a BACnet vendor identification code may define their own application interfaces, or they may extend a standard interface.

If an organization intends to extend a standard Application Interface, it is recommended that this be done in such a way that any conforming implementation of the organization's Application Interface also conforms to the requirements of the standard Application Interface. If the standard Application Interface to be extended defines an Interface Table, that table should not be modified with few exceptions (e.g., to make a value required); if new values are needed, the organization should define an additional Interface Table to contain the values that are not part of the standard Application Interface.

Modifying any existing application interface should be done with caution in order to reduce the potential for interoperability problems with existing implementations. Recommended practice is as follows:

(a) Never change the IDs of existing values in the Interface Table.
(b) Do not remove any values from the Interface Table. Values that will no longer be implemented should be marked with the conformance code "D" (deprecated).
(c) If it is necessary to make a significant change to the function of a particular value in an Interface Table, deprecate the old value and add a new value.

(d) Include a version number in the Application Interface Name, and advance the version number if a substantive change is made to the definition of an Application Interface.

135-2010*ap*-2 Enhance Structured View Object

Rationale

See section 1 rationale for overall concept of Application Interfaces.

This section defines the extensions to the existing Structured View object to support the deployment scenarios and requirements for Application Interfaces.

This proposal changes the datatype of the Subordinate_List from BACnetDeviceObjectReference to a new datatype, BACnetReference. The new datatype is backward compatible at the binary encoding level with BACnetDeviceObjectReference when used to reference objects. However, it has new options to include references to properties, including array index.

Despite their flexibility, the extensions discussed here stop short of full metadata information for workstations and very advanced clients to work with. What is proposed is the minimum that is needed for something like a binding tool or network diagnostic browser to work with these interface members. Richer metadata (writability, limits, volatility, etc.), is available in the XML interface definitions and it was not the intent to reproduce all that in binary form.

Client Usage:

This proposal attempts to simplify life for the client by making subordinates positionally accessible and making them always a pointer. So client code always does the same thing every time regardless of whether the subordinate is local or remote, standard object, or proprietary property.
The client notices the Interface_Name, and thereby knows the meaning of the subordinates and their positions (indexes). If the subordinate is primitive, it reads the reference and then reads or writes what it is pointing at. If the subordinate is itself an interface, then it reads the reference, which points to another Structured View, and the process repeats. Alternately, once the interface type is known, the client can use the ReadPropertyIndirect service (see section 3) and traverse multiple indirections in one step.

Positional access:

If we assume that BACnet Application Interfaces and the Building Blocks that make them up are each independently extensible over time, then, recognizing that it is not possible to "flatten" a collection of independently extensible child blocks into a unchanging positions for the parent interface, positional (indexed) access to interface members can only be achieved if the main interface and the child Building Blocks are modeled as separate views. But since this is usually good practice anyway, this is not a disadvantage, and positional access allows very simply and efficient traversal.

[Change **Clause 12.29**, p.301]

12.29 Structured View Object Type

The Structured View object type defines a standardized object that provides a container to hold references to subordinate ~~objects~~ *data*, which may include other Structured View objects, thereby allowing multilevel hierarchies to be created. The hierarchies are intended to convey a structure or organization such as a geographical distribution or application organization. Subordinate ~~objects~~ *data* may reside in the same device as the Structured View object or in other devices on the network.

Structured View objects may be used to create a collection of data that conforms to a BACnet Application Interface definition (see Clause xxx).

The Structured View object and its properties are summarized in Table 12-34 and described in detail in this subclause.

Table 12-34. Properties of the Structured View Object Type

Property Identifier	Property Datatype	Conformance Code
...	...	...
Subordinate_List	BACnetARRAY[N] of BACnet~~**DeviceObject**~~Reference	R
Subordinate_Annotations	BACnetARRAY[N] of CharacterString	O
Interface_Name	*CharacterString*	*O*[1]
...	...	...
Referenced_By	*List of BACnetReference*	*O*
...	...	...
Extended_Subordinate_List	*BACnetARRAY[N] of BACnetReference*	*O*
Extended_Subordinate_Annotations	*BACnetARRAY[N] of CharacterString*	*O*
Extended_Interface_Name	*CharacterString*	*O*

[1] *Required if the Structured View conforms to a BACnet Application Interface definition*

12.29.7 Subordinate_List

This property, *of type* ~~is a~~ BACnetARRAY of BACnet~~DeviceObject~~*Reference*, ~~that defines~~ *specifies the location of* the members of the ~~current~~ Structured View.

By including references to 'child' Structured View objects, multilevel hierarchies may be created. *To avoid recursion, it is recommended that a single Structured View object should be referenced only once in the hierarchy.*

For references of type 'object' and 'property', if ~~If~~ the optional device identifier is not present for a particular Subordinate_List member, then that object must reside in the same device that maintains the Structured View object. *For references of type 'property', if the optional object identifier is not present, then the referenced property resides in the Structured View object itself.* If Subordinate_List is writable using WriteProperty services, the Subordinate_List may optionally be restricted to reference-only objects in the local device. ~~To avoid recursion, it is suggested that a single Structured View object should be referenced only once in the hierarchy.~~

When conforming to a BACnet Application Interface, the value of each subordinate is ultimately a single property value. If the property is referred directly with the 'property' choice, then no other properties are assumed to be related to the referenced property. If the property is referred to indirectly using the 'object' choice, then the implied property is the Present_Value of the referenced object and all associated properties in that object are relevant. If the 'uri' choice is used, then the exact location of the property value and any associated values are determined by the type of URI and the type of data structure at the referenced path.

If the size of the Subordinate_List array is changed, the size of the Subordinate_Annotations array, if present, shall also be changed to the same size. Uninitialized Subordinate_List array elements shall *use the 'object' or 'property' choice with the object identifier field having* ~~be given~~ the instance number 4194303, *or shall use the 'uri' choice with a zero length string.*

A Subordinate_List array element *the has a 'uri' choice with a string length of zero or* whose *object identifier* instance number is equal to 4194303 shall be considered uninitialized and shall be ignored.

[Add new Clauses, p.303]

12.29.X1 Interface_Name

This property, of type CharacterString, is the name of a BACnet Application Interface to which this object conforms. A BACnet Application Interface defines the meaning of the members of the Subordinate_List property.

To ensure uniqueness, an interface name shall begin with a vendor identifier code (see Clause 23) in base-10 integer format,

followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the interface name shall indicate the organization that publishes and maintains the interface definition. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

12.29.X2 Referenced_By

This property, of type List of BACnetReference, is a list of pointers to subordinates of other Structured Views that are pointing to this Structured View object. This allows users of this Structured View object to easily find associated Structured Views, for example, to link an HVAC source to its consumers or an electric meter to its submeters.

As an example, if a Structured View for a VAV Terminal has a subordinate for "Air_Source" which points to the Structured View of an Air Handler, that Air Handler's Structured View could, in turn, have a pointer to the "Air_Source" subordinate property of the Structured View of the VAV Terminal in its Referenced_By property. This way, an observer starting with either Structured View would know of the relationship between the two.

12.29.X3 Extended_Subordinate_List

This property, of type BACnetARRAY of BACnetReference, specifies the location of additional members of the Structured View. When a Structured View conforms to a BACnet Application Interface, the meaning of the Subordinate_List is determined by the definition of the BACnet Application Interface specified in the Interface_Name property. The Extended_Subordinate_List is therefore used to hold additional proprietary members of the view, and the set of additional members is optionally indicated by the Extended_Interface_Name property.

By including references to ‘child’ Structured View objects, multilevel hierarchies may be created. To avoid recursion, it is recommended that a single Structured View object should be referenced only once in the hierarchy

For references of type 'object' and 'property', if the optional device identifier is not present for a particular Extended_Subordinate_List member, then that object must reside in the same device that maintains the Structured View object. For references of type 'property', if the optional object identifier is not present, then the referenced property resides in the Structured View object itself. If Extended_Subordinate_List is writable using WriteProperty services, the Extended_Subordinate_List may optionally be restricted to reference-only objects in the local device.

If the size of the Extended_Subordinate_List array is changed, the size of the Extended_Subordinate_Annotations array, if present, shall also be changed to the same size. Uninitialized Extended_Subordinate_List array elements shall use the 'object' or 'property' choice with the object identifier field having been given the instance number 4194303, or shall use the 'uri' choice with a zero length string.

An Extended_Subordinate_List array element that has a 'uri' choice with a string length of zero or whose object identifier instance number is equal to 4194303 shall be considered uninitialized and shall be ignored.

12.29.X4 Extended_Subordinate_Annotations

This property, of type BACnetARRAY of CharacterString, shall be used to define a text string description for each member of the Extended_Subordinate_List. The content of these strings is not restricted.

12.29.X5 Extended_Interface_Name

This property, of type CharacterString, is the name of a proprietary extension to which this object conforms. A proprietary extension defines the contents of the Extended_Subordinate_List property.

To ensure uniqueness, an interface name shall begin with a vendor identifier code (see Clause 23) in base-10 integer format, followed by a dash. All subsequent characters are administered by the organization registered with that vendor identifier code. The vendor identifier code that prefixes the interface name shall indicate the organization that publishes and maintains the interface definition. This vendor identifier need not have any relationship to the vendor identifier of the device within

which the object resides.

[Add new construction to Clause 21, p.626]

BACnetReference ::= CHOICE { -- tags [0]&[1] compatible with BACnetDeviceObjectReference

```
        object          SEQUENCE {
                        deviceIdentifier        [0] BACnetObjectIdentifier OPTIONAL,
                        objectIdentifier        [1] BACnetObjectIdentifier,
                        },
        property        [2] SEQUENCE {
                        deviceIdentifier        [0] BACnetObjectIdentifier OPTIONAL,
                        objectIdentifier        [1] BACnetObjectIdentifier OPTIONAL,
                        propertyIdentifier      [2] BACnetPropertyIdentifier,
                        propertyArrayIndex      [3] Unsigned OPTIONAL,
                        },
        uri             [3] CharacterString
}
```

135-2010*ap*-3 Add New Service ReadPropertyIndirect

Rationale

See section 1 rationale for overall concept of Application Interfaces.

The proposed use of Structured View objects to represent application interfaces increases the amount of effort (via indirection) required to access property values.

This proposal defines a new service which would allow simple traversal of an object hierarchy in order to read values in a more efficient manner.

[Insert **Clause 15.X**, p. 489]

15.X ReadPropertyIndirect Service

The ReadPropertyIndirect service allows a client to traverse a hierarchy of objects within a single device when the client knows the path through the objects. This supports efficient consumption of hierarchical application interfaces.

When the end of the path is reached, the service returns the value of the referenced property. If, in traversing the path, the path refers to an object in another device, the service will return the depth read and the reference that leaves the device. This allows the client to continue the path traversal through a request to the referenced device.

This service may be used to traverse any object type that contains an array, list, or array of lists of references. Examples of such properties are Subordinate_List and Group_Members and for the purpose of describing this service, these properties are referred to as reference lists. Note that the List_Of_Group_Members found in the Group object is not considered a reference list property due to the complex structure of the List_Of_Group_Members.

Every BACnet device that is capable of containing Structured View objects that represent application interfaces shall execute this service.

15.X.1 Structure

The structure of the ReadPropertyIndirect primitive is shown in Table 15-X1. The terminology and symbology used in this table are explained in Clause 5.6.

Table 15-X1. Structure of ReadPropertyIndirect Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	M	M(=)		
Property Identifier	C	C(=)		
Property Array Index	C	C(=)		
Path	M	M(=)		
Result(+)			S	S(=)
Object Identifier			M	M(=)
Property Identifier			M	M(=)
Property Array Index			C	C(=)
Path			M	M(=)
Depth Traversed			M	M(=)
Referenced Object Identifier			M	M(=)
Referenced Property Identifier			C	C(=)
Referenced Array Index			C	C(=)
Referenced List Index			C	C(=)
Referenced Property Value			M	M(=)
Error Type			C	C(=)

Result(-) Error Type			S M	S(=) M(=)

15.X.1.1 Argument

This parameter shall convey the parameters for the ReadPropertyIndirect confirmed service request.

15.X.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, identifies the object in which the path starts.

15.X.1.1.2 Property Identifier

This optional parameter, of type BACnetPropertyIdentifier, identifies the array or list property that corresponds to the initial entry in the Path parameter.

If the ‘Property Identifier’ parameter is not present, then the property shall be inferred by the object type being read using the same rules for inferring a property identifier during path traversal.

15.X.1.1.3 Property Array Index

If the property identified above is of datatype array, this optional parameter, of type Unsigned, shall indicate the array index of the element of the property referenced by this service. If the 'Property Array Index' is omitted, this shall mean that the entire array shall be referenced.

If the property identified above is not of datatype array, this parameter shall be omitted.

When the property identified above is a reference list and is not an array of reference lists, under most circumstances, this property is not included. If it is included, then the identified element would not be a reference list, and the service treats the property in the same way that it treats any non-reference list property.

If this parameter is present and has a value of 0, then the reference is to the length of the array property and is treated as a non-reference list property.

15.X.1.1.4 Path

The 'Path' parameter is a sequence of one or more Unsigned values that specify the indices into the arrays or lists of references. The nth entry in the sequence is the index used for the nth array or list encountered while traversing the hierarchy of objects.

15.X.1.2 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded in whole or in part. A successful result includes the following parameters.

15.X.1.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, specifies the object that was read. It is equal to the service request parameter of the same name.

15.X.1.2.2 Property Identifier

This parameter, of type BACnetPropertyIdentifier, shall identify that property that was read. If a ‘Property Identifier’ was specified in the request, this parameter shall be equal to that parameter’s value, otherwise it shall indicate the inferred property.

15.X.1.2.3 Property Array Index

If a 'Property Array Index' was specified in the request, this parameter, of type Unsigned, shall be present and shall be equal to that parameter’s value. Otherwise it shall be omitted.

15.X.1.2.4 Path

This parameter, of type sequence of Unsigned, specifies the path that was requested to be read. The whole path provided in the request is returned in the result regardless of the depth that was successfully read.

15.X.1.2.5 Depth Traversed

This parameter, of type Unsigned, specifies the depth in the path that was successfully read. If the complete path is traversed, this value will be one more than the number of items in the path.

15.X.1.2.6 Referenced Object Identifier

This parameter, of type BACnetObjectIdentifier, specifies the last object in the path that was successfully read.

15.X.1.2.7 Referenced Property Identifier

This parameter, of type BACnetPropertyIdentifier, specifies the last property in the path that was successfully read.

15.X.1.2.8 Referenced Array Index

This parameter, of type Unsigned, specifies the array index into the last property in the path that was successfully read. If the last property that was successfully read is an array property, and a single element or the size of the array was read, then this parameter shall be present.

15.X.1.2.9 Referenced List Index

This parameter, of type Unsigned, specifies the position in last property in the path that was successfully read. If the last property that was successfully read is a list property, and a single element of the list was read, then this parameter shall be present. When this parameter is present, it is equal to the entry in the Path service parameter that identified the list item to read.

If the last read property is not a list property, or the complete list was referenced, then this parameter shall be absent.

15.X.1.2.10 Referenced Property Value

This parameter conveys the value read from the last referenced property.

15.X.1.2.11 Error Type

This parameter is included whenever the complete path is not traversed. It consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are documented in the description of Error Type of an Result(-) response.

15.X.1.3 Result(-)

The 'Result(-)' parameter shall indicate that the service request has failed in its entirety. The reason for the failure shall be specified by the 'Error Type' parameter.

15.X.1.4 Error Type

This parameter consists of two component parameters: (1) the 'Error Class' and (2) the 'Error Code'. See Clause 18. The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
Specified object does not exist.	OBJECT	UNKNOWN_OBJECT
Specified property does not exist.	PROPERTY	UNKNOWN_PROPERTY
The ‘Property Identifier’ parameter, or the property field of the current reference, is not present and the property to read cannot be implied by the object type.	PROPERTY	NO_PROPERTY_SPECIFIED
The specified property is currently not readable by the requester.	PROPERTY	READ_ACCESS_DENIED

The next property on the path is not a reference list property, or the next reference is not initialized.	PROPERTY	END_OF_PATH
An array index is provided but the property is not an array of lists of references.	PROPERTY	PROPERTY_IS_NOT_AN_ARRAY
An array index is provided that is outside the range existing in the property.	PROPERTY	INVALID_ARRAY_INDEX
An index from the Path parameter is outside the range of items in property for that level of the path.	PROPERTY	INVALID_ARRAY_INDEX
The next object on the path is located in a different device.	OBJECT	PATH_LEAVES_DEVICE
The next reference on the path is a URI in a BACnetReference.	OBJECT	PATH_LEAVES_DEVICE
The property value is too large to return in the response	PROPERTY	VALUE_TOO_LONG

15.X.2 Service Procedure

The responding BACnet-user shall first verify the validity of the 'Object Identifier', 'Property Identifier', and 'Property Array Index' parameters and return a 'Result(-)' response with the appropriate error class and code if the object or property is unknown, or if the specified property is currently inaccessible for another reason.

The path shall be traversed as far as possible and a Result(+) is returned which indicates the last value read. The path is a list of indices into reference list properties. The service parameters Object Identifier, Property Identifier, and Array Index specify the reference list that corresponds with the first index in the Path service parameter. The value is read from the specified entry in the reference list and is used as the reference along with the next index in the Path service parameter, if there is one. This repeats until an error is encountered, the path leaves the device, or the end of the path is reached. When the end of the path is reached, the resulting reference is read, if it references an object in the device, and the value is returned.

When reading a reference along the path, if the reference does not include a Property Identifier, then the next Property Identifier shall be inferred from the type of the referenced object and whether or not the end of the path has been reached. When inferring the reference property in a standard object type, only standard properties for the object type as documented in Clause 12 are considered. Thus, non-standard reference list properties cannot be implied and must be explicitly referred to.

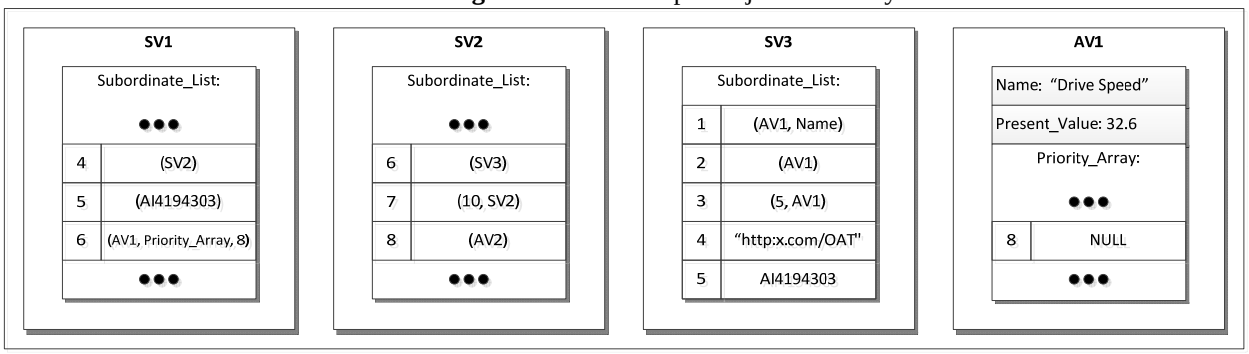
Table 15-X2. Inferring Property Identifiers

Object	Middle Of Path	End Of Path
Has a single reference list property and no Present_Value property.	Reference list property	Reference list property
Has a single reference list property and a Present_Value property.	Reference list property	Present_Value
Has no reference list property and a Present_Value property.	Present_Value	Present_Value
Has multiple reference list properties and a Present_Value property.	No property can be inferred.	Present_Value
Has multiple reference list properties and no Present_Value property.	No property can be inferred.	No property can be inferred.

No reference list properties and no Present_Value.	No property can be inferred.	No property can be inferred.

If the path cannot be completely traversed, the last property along the path that can be read shall be indicated along with its value. The reason that the path could not be completely read shall be returned. If the path is not traversed because it leaves the device, the Referenced Property Value returned shall be a BACnetReference so that clients that do not know the datatype of the reference list property are able to consume the returned reference.

Figure 15-X1. Example Object Hierarchy



Using the example object hierarchy in Figure 15-X1, the Table 15-X3 shows the expected return values.

Table 15-X3. Example Service Return Values

Object Identifier	Path	Depth	Returned Reference	Referenced Property Value	Error Type
SV1	(4,6,2)	4	(AV1, Present_Value)	36.2	-
SV1	(4,6,2,9)	4	(AV1, Present_Value)	36.2	PROPERTY, END_OF_PATH
SV1	(6)	2	(AV1, Priority_Array, 8)	NULL	-
SV1	(4,6)	3	(SV3, Subordinate_List)	((AV1, Name), (AV1) (5, AV1), ...)	-
SV1	(4,6,32)	2	(SV3, Subordinate_List)	((AV1, Name), (AV1) (5, AV1), ...)	PROPERTY, INVALID_ARRAY_INDEX
SV1	(4,7)	2	(SV2, Subordinate_List, 7)	(10, SV2)	PROPERTY, PATH_LEAVES_DEVICE
SV2	(7, 1)	1	(SV2, Subordinate_List, 7)	(10, SV2)	PROPERTY, PATH_LEAVES_DEVICE
SV2	(6, 3)	2	(SV3, Subordinate_List, 3)	(5, AV1)	PROPERTY, PATH_LEAVES_DEVICE
SV1	(4,6,4)	3	(SV3, Subordinate_List, 4)	"http:x.com/OAT"	PROPERTY, PATH_LEAVES_DEVICE
SV1	(5)	1	(SV1, Subordinate_List, 5)	(AI4194303)	PROPERTY, NO_PROPERTY_SPECIFIED
SV1	(4,8)	2	(SV2, Subordinate_List, 8)	(AV2)	OBJECT, UNKNOWN_OBJECT
SV4	(1,2,3,4)	-	-	-	OBJECT, UNKNOWN_OBJECT
AV1	(1,2,3,4)	1	(AV1, Present_Value)	36.2	PROPERTY, END_OF_PATH

[Change **Clause 18.3** p 520]

18.3 Error Class - PROPERTY

...

NO_PROPERTY_SPECIFIED – *The operation was not successful due a property reference containing a device or object instance equal to 4194303 or missing required fields required to fully identify the target of the operation.* ~~No data was logged due to a device or object instance equal to 4194303 in the list of logged properties.~~

...

DUPLICATE_OBJECT_ID - An attempt has been made to write to an Object_Identifier property with a value that is already in use in a different Object_Identifier within the same device.

PATH_LEAVES_DEVICE *– In traversing a path, the device has encountered a reference to an object in a different device.*

END_OF_PATH *– In traversing a path, the end of the path was reached prematurely.*

OTHER - This error code is returned for a reason other than any of those previously enumerated for this Error Class.

[Change **BACnet-Confirmed-Service-Request** production in **Clause 21**, p 565]

BACnet-Confirmed-Service-Request ::= CHOICE {

...

readProperty	[12] ReadProperty-Request,
readPropertyIndirect	*[X] ReadPropertyIndirect-Request,*
readPropertyMultiple	[14] ReadPropertyMultiple-Request,

...

-- Services added after 1995

-- readRange	[26] see Object Access Services
-- lifeSafetyOperation	[27] see Alarm and Event Services
-- subscribeCOVProperty	[28] see Alarm and Event Services
-- getEventInformation	[29] see Alarm and Event Services
-- *readPropertyIndirect*	*[X] see Object Access Services*

}

[Add new productions to **Clause 21**, p 570]

ReadPropertyIndirect-Request ::= SEQUENCE {

objectIdentifier	[0] BACnetObjectIdentifier,
propertyIdentifier	[1] BACnetPropertyIdentifier,
propertyArrayIndex	[2] Unsigned OPTIONAL, --used only with array datatype
path	[3] SEQUENCE OF Unsigned

}

ReadPropertyIndirect-ACK ::= SEQUENCE {

objectIdentifier	[0] BACnetObjectIdentifier,
propertyIdentifier	[1] BACnetPropertyIdentifier,
propertyArrayIndex	[2] Unsigned OPTIONAL, --used only with array datatype
depthTraversed	[3] Unsigned,
referencedObjectIdentifier	[4] BACnetObjectIdentifier,
referencedPropertyIdentifier	[5] BACnetPropertyIdentifier,
referencedArrayIndex	[6] Unsigned OPTIONAL, --used only with array datatype
referencedListIndex	[7] Unsigned OPTIONAL, --used only with list datatype
referencedPropertyValue	[8] ABSTRACT-SYNTAX.&Type,
errorType	[9] Error OPTIONAL -- provided when path not fully traversed

}

[Change **Clause 21**, p 577]

Error ::= SEQUENCE {
...
error-code ENUMERATED { -- see below for numerical order
...
end-of-path *(X),*
...
path-leaves-device *(X+1),*
...
-- see end-of-path *(X),*
-- see path-leaves-device *(X+1),*
...
}
}

[Change **Clause 21**, p 628]

BACnetServicesSupported ::= BIT STRING {
...
readProperty (12),
-- *readPropertyIndirect* *(X),*
readPropertyMultiple (14),
...
-- Services added after 1995
...
getEventInformation (39), -- Alarm and Event Service
readPropertyIndirect *(X) -- Object Access Service*
}

135-2010*ap*-4 Define Machine-Readable Definitions for Application Interfaces

Rationale

See section 1 rationale for overall concept of Application Interfaces.

This section defines the format for machine-readable (XML) definitions of Application Interfaces.

[Insert new **Annex X** (note that 'X' is a place-holder, not the actual annex letter)]

ANNEX X – CSML DEFINITION OF APPLICATION INTERFACES (INFORMATIVE)

(This Annex is not part of this standard but is included for informative purposes only.)

This annex recommends a format for describing Application Interfaces (Clause 26) in CSML (Control Systems Modeling Language, Annex Q). The intent of this annex is to provide a method for conveying the metadata associated with a given Application Interface in a machine-readable format that can be utilized by clients of an extended Structured View object (Clause 12.29) that realizes an interface, or by other client programs such as configuration workstations.

X.1 Introduction

Clause 26 specifies a standard process for defining and publishing Application Interfaces. The value of this process is that it provides a middle ground between the formal standardization of new Clause 12 objects and the private (and therefore non-interoperable) specification of proprietary objects deployed by many vendors. By adopting this process, organizations can specify object-like behavior for new applications or functional areas without mandating the precise implementation.

In order to realize the full potential of this process, the Structured View object was revised to allow fine-grained reference to arbitrary objects (including, importantly, other Structured View objects) or properties, in local or remote devices. This new flexibility allows Structured View objects to represent the realization, or "as-built" specification, of an instance of an Application Interface. The ReadPropertyIndirect service (Clause 15.x) was introduced to provide a more efficient way for clients to access the interface.

As in Annex Q.5, the terms "type" and "definition" are nearly synonymous and used interchangeably in this annex, but the term "definition" always refers to a referenceable element (a "typedef" in some languages). An "instance" is typically an element that refers to a definition element using the 'type' attribute and contains values (which may be assigned defaults in the definition element).

The process for creating an Application Interface definition may be summarized as follows:

(a) Extend the CSML definition of Structured View to create a new type
(b) Publish the CSML definition together with the table described in Clause 26
(c) Use the new definition to create instances of the Application Interface

The semantic meaning and data type information for the Application Interface elements are described by extending the Structured View object and defining a <Target> element for each subordinate. An extended Structured View of the requisite type that represents an instance of an interface is simply a container of references (bindings) to the objects and properties that represent the elements of the interface.

An organization defining a new interface is not required to describe this information in CSML, but it is strongly recommended. If a CSML description is provided, the format provided in this annex is the recommended one.

X.2 CSML Type Definitions

Clause 26.4.3 specifies that an Application Interface value may have one of the following types:

(a) Any Clause 21 Application Type, except NULL

(b) Any Clause 21 Base Type
(c) An Application Interface, indicated by "Interface"
(d) A one-dimensional array of any permitted datatype, indicated by "Array of <datatype>"

CSML represents the Clause 21 Application Types as Primitive Data using <Boolean>, <Unsigned>, <Integer>, <Enumerated>, etc., elements (Annex Q.3.11). CSML can also represent Constructed Data using <Sequence>, <Object>, <Array>, <List>, <SequenceOf>, etc., elements (Annex Q.3.12).

A digital file containing normative CSML for standard BACnet Base Types may be found at [web location TBD] and included by reference in CSML Application Interface definition files. By convention, CSML definitions are prefixed by vendor code. All ASHRAE-defined types begin with "0-". Application Interface types begin with "0-AIF-".

The purpose of a CSML Application Interface definition is to represent the required interface elements, specified in accordance with Clause 26, in a machine-readable format. As stated above, "Interface" is also an allowable datatype for an interface value and may refer to any definition based on extending the Structured View object type as described in the following sections. The definition for the revised Structured View object type is described first, followed by the constituent 0-BACnetReference and <Target> element descriptions. Finally, example definitions are given in section X.5.

X.2.1 0-Object-Base and 0-Object-StructuredView Type Definitions

A non-normative example definition for the "0-Object-StructuredView" is shown below to illustrate how definitions may be constructed from primitive types, and how more complex types can be defined by extending simpler ones.

```
<Definitions>
 <Object name="0-Object-MinimumBase" displayName="Minimum Base Object">
  <Documentation> This is the minimal BACnet Object.  All BACnet objects are
   required to have these three properties.  The object can be
   uniquely identified in its device by either the Object_Identifier property
   or the Object_Name property. </Documentation>
  <ObjectIdentifier name="object-identifier"
   type="0-Property-Object_Identifier" displayName="Object Identifier"/>
  <String name="object-name" type="0-Property-Object_Name"
   displayName="Object Name"/>
  <Enumerated name="object-type" type="0-Property-Object_Type"
   displayName="Object Type"/>
 </Object>
 <Object name="0-Object-CommonBase" extends="0-Object-MinimumBase"
  displayName="Common Base Object">
  <Documentation>Many BACnet objects share the properties defined here, in addition to
   the required minimum set.</Documentation>
  <String name="description" optional="true" displayName="Description"/>
  <String name="profile-name" optional="true" displayName="Profile Name"/>
 </Object>
 <Object name="0-Object-StructuredView" extends="0-Object-CommonBase"
  displayName="Structured View Object"
  comment="ANSI/ASHRAE 135-2010 pg. 720, as revised by DR-037">
  <Enumerated name="node-type" type="0-BACnetNodeType"
   displayName="Node Type"/>
  <String name="node-subtype" displayName="Node Subtype" optional="true"/>
  <Array name="subordinate-list" memberType="0-BACnetReference"
   displayName="Subordinate List"/>
  <Array name="subordinate-annotations" memberType="String"
   displayName="Subordinate Annotations" optional="true"/>
  <String name="interface-name" displayName="Interface Name" optional="true"/>
  <List name="referenced-by" memberType="String" displayName="Referenced By"
   optional="true"/>
  <Array name="extended-subordinate-list" memberType="0-BACnetReference"
   displayName="Extended Subordinate List"/>
  <Array name="extended-subordinate-annotations" memberType="String"
   optional="true" displayName="Extended Subordinate Annotations"/>
```

```
  <String name="extended-interface-name" optional="true"
   displayName="Extended Interface Name"/>
 </Object>
</Definitions>
```

X.2.2 0-BACnetReference Type Definition

In the Structured View object type, the elements in the 'subordinate-list' array are of the type BACnetReference. This allows fine-grained references (down to the property and index level) in either the local object, local device, or a remote device. Unbound (null) references are indicated by an objectIdentifier with value="4194304" or uri with value="" (empty string).

A non-normative example definition for 0-BACnetReference is shown below:

```
<Definitions>
 <Choice name="0-BACnetReference">
  <Choices>
   <!-- Object Ref (backward compatible with 0-BACnetDeviceObjectReference) -->
   <Sequence name="object">
    <ObjectIdentifier name="deviceIdentifier" contextTag="0" optional="true"/>
    <ObjectIdentifier name="objectIdentifier" contextTag="1"/>
   </Sequence>
   <!-- Property Ref -->
   <Sequence name="property" contextTag="2">
    <ObjectIdentifier name="deviceIdentifier" contextTag="0" optional="true"/>
    <ObjectIdentifier name="objectIdentifier" contextTag="1" optional="true"/>
    <Enumerated name="propertyIdentifier" contextTag="2"
     type="0-BACnetPropertyIdentifier"/>
    <Unsigned name="propertyArrayIndex" contextTag="3" optional="true"
     comment="Used only with array datatype. If omitted, the entire array is
     referenced."/>
   </Sequence>
   <!-- URI -->
   <String name="uri" contextTag="3"/>
  </Choices>
 </Choice>
</Definitions>
```

X.3 Application Interface Name

The value of the 'name' attribute of an extended Structured View object that defines an Application Interface shall be in the format specified in Clause 26.2, e.g., name="555-AIF-Example-VFD.1". This standardized formal name is used to correlate the metadata description with other representations of the interface, both written and network-visible. The document in which the interface metadata is defined should also have the same root name with a suffix of ".xml" appended, e.g., "555-AIF-Example-VFD.1.xml". The 'displayName' attribute of the extended Structured View object should succinctly convey the purpose of the interface, e.g., displayName="Example Variable Frequency Drive".

X.4 Interface Table Elements

Each row in an Application Interface table, as described in Clause 26.4, is represented by a <Sequence> element corresponding to a member of the 'subordinate-list' array in an extended Structured View object. The columns of the interface table correspond to attribute value settings of those members or their child elements. The column names defined in Clause 26.4 and the corresponding CSML attribute value settings are described in sections below.

X.4.1 <Sequence> Element

The ID (Clause 26.4.1) of an interface table entry is represented by the 'name' attribute value of the <Sequence> element that corresponds to that member in the 'subordinate-list' array. The 'name' attribute of each array member is a decimal integer, beginning with 1, corresponding to its position in the array. The name should be explicitly set to the corresponding ID in the

interface table. This value also indicates the index of the corresponding reference in the subordinate-list of an extended Structured View object that realizes that interface.

X.4.2 <Target> Element

While the 0-BACnetReference type shown above is sufficient to represent fine-grained references to elements on any device, it cannot represent metadata such as the datatype of the reference. This is accomplished by specifying a <Target> element for each member of the 'subordinate-list' array.

The <Target> element is similar in construction to <MemberType> for elements of a collection, but is used for a construct that contains a reference. It may be applied to a <Sequence> or <String> element and is used to convey datatype, range limits, engineering units, etc., of the referent. Each member in the 'subordinate-list' array is represented by a <Sequence> element having a name equal to the subordinate's numeric ID (which is equivalent to the index in the array). A <Target> element is enclosed within each <Sequence> element as shown in the examples below.

X.4.3 Flexible Datatypes

Clause 26.4.3 permits the datatype of an interface table entry to be specified as a "flexible" type. This allows the bound value for a realized interface table entry to be any one of the allowable types. A client reading this value at runtime must be prepared to accept any of the allowable types and to coerce the datatype of the returned value if necessary. The flexible types are defined in the following table.

Table X-1. Flexible Datatypes

Flexible Type Designation	Implementation Options
Binary	BOOLEAN, ENUMERATED, or Unsigned where 0 equals false and 1 equals true
Float	Real (preferred) or Double
Multi-state	Unsigned (preferred), ENUMERATED, REAL, or Double in positive integral increments, excluding zero. e.g., 1,2,3... or 1.0, 2.0, 3.0...
Signed	INTEGER (preferred), REAL, or Double in integral values. e.g., ...,-2, -1, 0, 1, 2... or ...,-2.0, -1.0, 0.0, 1.0, 2.0, 3.0...
Unsigned	Unsigned (preferred), INTEGER, REAL, or Double in integral values, including zero. e.g., 0,1,2,3... or 0.0, 1.0, 2.0, 3.0...

The primitive types that constitute these choices may be restricted by 'minimum', 'maximum', or 'resolution' attributes (Annex Q.3.6). The range of allowable values may be dictated by the application or by the specified engineering units (e.g., the normal range of "percent" is 0 – 100).

In CSML Application Interface definitions, flexible types are represented by an XML tag for the preferred primitive type, e.g., Binary is represented as a <Boolean> element. The client must determine at run time, and particularly before writing to a referenced property, its actual type. In the case where a Signed or Unsigned value is realized as a Float, values shall be rounded up or down to the nearest integer value.

X.4.3.1 Binary Datatype

The Binary datatype is represented by a <Boolean> element as in the following example:

```
<Sequence name="1">
 <Target>
  <Boolean name="run-stop-monitor" displayName="RUN-STOP Monitor"
   volatility="volatile" variability="status">
   <NamedValues>
    <Boolean name="stop" value="false" displayName="STOP"/>
    <Boolean name="run" value="true" displayName="RUN"/>
   </NamedValues>
```

```
  </Boolean>
 </Target>
</Sequence>
```

The displayName/value bindings are determined by the Range/Units values specified in the corresponding Application Interface table entry.

X.4.3.2 Float Datatype

The Float datatype is represented by a <Real> element as in the following example:

```
<Sequence name="4">
 <Target>
  <Real name="output-speed" displayName="Output Speed"
   volatility="volatile" variability="status" minimum="0.0"
   maximum="100.0" units="percent"/>
 </Target>
</Sequence>
```

Note that in this example the specified units dictate the allowable range of values.

X.4.3.3 Multi-State Datatype

The Multi-state datatype is represented by an <Enumeration> element as in the following example:

```
<Sequence name="3">
 <Target>
  <Enumerated name="hand-auto-reference" displayName="HAND-AUTO Reference"
   volatility="volatile" variability="status">
   <NamedValues>
    <Unsigned name="off" value="1" displayName="Off"/>
    <Unsigned name="on" value="2" displayName="On"/>
    <Unsigned name="auto" value="3" displayName="Auto"/>
   </NamedValues>
  </Enumerated>
 </Target>
</Sequence>
```

displayName/value bindings are determined by the Range/Units values specified in the corresponding Application Interface table entry.

X.4.3.4 Signed Datatype

The Signed datatype is represented by an <Integer> definition as in the following example:

```
<Sequence name="17>
 <Target>
  <Integer name="signed-example" displayName="Signed Example"
   minimum="0" maximum="255"/>
 </Target>
</Sequence>
```

X.4.3.5 Unsigned Datatype

The Unsigned datatype is represented by an <Unsigned> element as in the following example:

```
<Sequence name="9">
 <Target>
  <Unsigned name="running-seconds" displayName="Running Seconds"
```

```
   volatility="volatile" variability="status" units="seconds"/>
 </Target>
</Sequence>
```

In this example, the range is restricted to non-negative numbers by definition.

X.4.4 Other Datatypes

Interface table elements defined in terms of Clause 21 base types or primitive types other than the ones described above are referred to as having "defined" types. A reference to a defined primitive type is shown in the following example (note it has a default value):

```
<Sequence name="19">
 <Target>
  <Date name="start-date" displayName="Start Date" value="2011-06-30" />
 </Target>
</Sequence>
```

Another example (below) defines a reference to defined datatype that happens to be another interface. A run-time client of an extended Structured View object that realizes the parent interface would expect to find an object reference to another Structured View object at this position in the subordinate-list, with a corresponding interface-name. If no name is given in the interface definition, the client should still expect a reference to an extended Structured View object but must read its interface-name property in order to access the interface definition.

```
<Sequence name="20">
 <Target>
  <Object name="power-meter" displayName="Power Meter" type="555-AIF-Power"/>
 </Target>
</Sequence>
```

X.4.5 Name

The Name (Clause 26.4.2) of an interface table entry is represented by the value of the 'name' attribute of the datatype element within the <Target> that corresponds to that member in the 'subordinate-list' array. The 'displayName' attribute of each datatype element provides a short name that describes its function. The string must be unique within a given interface and should be consistent with the Name specified in the interface table.

X.4.6 Conformance Code

The conformance code (Clause 26.4.4) of an interface table entry is represented by a combination of attribute values for the datatype element within the <Target> corresponding to a member of the 'subordinate-list' array, as shown in the table below. 'X' represents a value of "true" or "false".

Table X-2. Conformance Code Attributes

Code	<Target> attribute settings
R	optional="false"[1] writeable=X commandable=X
W	optional="false"[1] writeable="true" commandable=X
P	optional="false"[1] writeable=X commandable="true"
O	optional="true" writeable=X commandable=X
OW	optional="true" writeable="true" commandable=X
OP	optional="true" writeable=X commandable="true"
C[2]	optional="false"[1] writeable=X commandable=X
CW[2]	optional="false"[1] writeable="true" commandable=X
CP[2]	optional="false"[1] writeable=X commandable="true"
D	optional=X writeable=X commandable=X deprecated="true"

[1] Default setting; need not be present.

[2] Condition attributes include 'associatedWith', 'requiredWith', etc. See Annex Q.3.1.

X.4.7 Range Restrictions and Engineering Units

Range restrictions (Clause 26.4.5) for an interface table entry are represented by setting the values of the 'minimum', 'maximum', and 'resolution' attributes of the datatype element within the <Target> corresponding to that 'subordinate-list' array member. These attributes are described in Annex Q.3.6.1, Q.3.6.2, and Q.3.6.5, respectively. Other attributes can be set for the child attributes as appropriate.

Engineering units (Clause 26.4.5) for an interface table entry are represented by the value of the 'units' attribute of the datatype element within the <Target> corresponding to that 'subordinate-list' array member. The value of the 'units' attribute should be one of the enumeration names of the 0-BACnetEngineeringUnits type. See Annex Q.3.7.1.

X.4.8 Volatility

The volatility (Clause 26.4.6) of an interface table entry is represented by the value of the 'volatility' attribute of the datatype element within the <Target> corresponding to that 'subordinate-list' array member (Annex Q.3.1.21). A value of "V" in the interface table is represented by volatility="volatile". A value of "N" in the interface table is represented by volatility="nonvolatile".

X.4.9 Usage

The usage (Clause 26.4.7) of an interface table entry is represented by the value of the 'variability' attribute of the datatype element within the <Target> corresponding to that 'subordinate-list' array member (Annex Q.3.1.20). A value of "status" in the interface table is represented by variability="status". A value of "control" in the interface table is represented by variability ="operational-setting". A value of "configuration" in the interface table is represented by variability ="configuration-setting".

X.5 Examples

Standard Application Interfaces that appear in Annex B are defined in CSML using external XML files. Application Interfaces defined by other organizations use the same format, as shown below.

X.4.1 CSML for an Example Application Interface

File "555-AIF-Example-VFD.1.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<CSML defaultLocale="en" xmlns="http://bacnet.org/csml/1">

 <Definitions>
  <!-- An example Application Interface definition -->
  <Object name="555-AIF-Example-VFD.1" extends="0-Object-StructuredView"
   displayName="Variable Frequency Drive Example">

   <String name="interface-name" value="555-AIF-Example-VFD.1"/>

   <Array name="subordinate-list">
    <Sequence name="1">
     <Target>
      <Boolean name="run-stop-monitor" displayName="RUN-STOP Monitor"
       volatility="volatile" variability="status">
       <NamedValues>
        <Boolean name="stop" value="false" displayName="STOP"/>
        <Boolean name="run" value="true" displayName="RUN"/>
       </NamedValues>
      </Boolean>
     </Target>
```

```
</Sequence>

<Sequence name="2">
 <Target>
  <Boolean name="ok-fault-monitor" displayName="OK-FAULT Monitor"
   volatility="volatile" variability="status">
   <NamedValues>
    <Boolean name="ok" value="false" displayName="OK"/>
    <Boolean name="fault" value="true" displayName="Fault"/>
   </NamedValues>
  </Boolean>
 </Target>
</Sequence>

<Sequence name="3">
 <Target>
  <Enumerated name="hand-auto-reference" displayName="HAND-AUTO Reference"
   volatility="volatile" variability="status">
   <NamedValues>
    <Unsigned name="off" value="1" displayName="Off"/>
    <Unsigned name="on" value="2" displayName="On"/>
    <Unsigned name="auto" value="3" displayName="Auto"/>
   </NamedValues>
  </Enumerated>
 </Target>
</Sequence>

<Sequence name="4">
 <Target>
  <Real name="output-speed" displayName="Output Speed"
   volatility="volatile" variability="status" minimum="0.0"
   maximum="100.0" units="percent"/>
 </Target>
</Sequence>

<Sequence name="5">
 <Target>
  <Real name="pid-feedback" displayName="PID Feedback"
   volatility="volatile" variability="status" minimum="0.0" maximum="100.0"
   units="percent"/>
 </Target>
</Sequence>

<Sequence name="6">
 <Target>
  <Real name="output-current" displayName="Output Current"
   volatility="volatile" variability="status" minimum="0.0"
   units="amperes"/>
 </Target>
</Sequence>

<Sequence name="7">
 <Target>
  <Real name="output-power" displayName="Output Power"
   volatility="volatile" variability="status" minimum="0.0"
   units="kilowatts"/>
 </Target>
</Sequence>

<Sequence name="8">
 <Target>
  <Real name="kilowatt-hour-meter" displayName="Kilowatt-Hour Meter"
   volatility="volatile" variability="status" minimum="0.0"
```

```
      units="kilowatt-hours"/>
   </Target>
  </Sequence>

  <Sequence name="9">
   <Target>
    <Unsigned name="running-seconds" displayName="Running Seconds"
     volatility="volatile" variability="status" units="seconds"/>
   </Target>
  </Sequence>

  <Sequence name="10">
   <Target>
    <Real name="operating-temp-range" displayName="Operating Temp Range"
     volatility="volatile" variability="status" minimum="0.0" maximum="100.0"
     units="percent"/>
   </Target>
  </Sequence>

  <Sequence name="11">
   <Target>
    <Boolean name="run-stop-command" displayName="RUN-STOP Command"
     writable="true" volatility="volatile" variability="operational-setting">
     <NamedValues>
      <Boolean name="stop" value="false" displayName="STOP"/>
      <Boolean name="run" value="true" displayName="RUN"/>
     </NamedValues>
    </Boolean>
   </Target>
  </Sequence>

  <Sequence name="12">
   <Target>
    <Boolean name="pid-enable" displayName="PID Enable" writable="true"
     volatility="nonvolatile" variability="operational-setting">
     <NamedValues>
      <Boolean name="direct" value="false" displayName="Direct"/>
      <Boolean name="pid" value="true" displayName="PID"/>
     </NamedValues>
    </Boolean>
   </Target>
  </Sequence>

  <Sequence name="13">
   <Target>
    <Boolean name="fault-reset-command" displayName="Fault Reset Command"
     writable="true" volatility="volatile" variability="operational-setting">
     <NamedValues>
      <Boolean name="off" value="false" displayName="Off"/>
      <Boolean name="on" value="true" displayName="On"/>
     </NamedValues>
    </Boolean>
   </Target>
  </Sequence>

  <Sequence name="14">
   <Target>
    <Enumerated name="most-recent-fault-code"
     displayName="Most Recent Fault Code" volatility="volatile"
     variability="status">
     <NamedValues>
      <Unsigned value="1" name="noneRecorded" displayName="None Recorded"/>
      <Unsigned value="2" name="communicationError"
```

```
          displayName="Communication Error"/>
         <Unsigned value="3" name="overCurrent" displayName="Over Current"/>
         <Unsigned value="4" name="overTemperature"
          displayName="Over Temperature"/>
         <Unsigned value="5" name="overSpeedDeviation"
          displayName="Over Speed Deviation"/>
         <Unsigned value="6" name="overVoltage" displayName="Over Voltage"/>
         <Unsigned value="7" name="underVoltage" displayName="Under Voltage"/>
         <Unsigned value="8" name="shortCircuit" displayName="Short Circuit"/>
         <Unsigned value="9" name="groundFault" displayName="Ground Fault"/>
         <Unsigned value="10" name="motorOverload"
          displayName="Motor Overload"/>
         <Unsigned value="11" name="inverterOverload"
          displayName="Inverter Overload"/>
         <Unsigned value="12" name="overTorqueProtection"
          displayName="Over Torque Protection"/>
         <Unsigned value="13" name="externalFault"
          displayName="External Fault"/>
         <Unsigned value="14" name="operatorInterfaceError"
          displayName="Operator Interface Error"/>
         <Unsigned value="15" name="loadLoss" displayName="Load Loss"/>
         <Unsigned value="16" name="configurationError"
          displayName="Configuration Error"/>
         <Unsigned value="17" name="feedbackFailure"
          displayName="Feedback Failure"/>
         <Unsigned value="18" name="outputPhaseLoss"
          displayName="Output Phase Loss"/>
         <Unsigned value="19" name="motorStall" displayName="Motor Stall"/>
         <Unsigned value="20" name="powerUnitError"
          displayName="Power Unit Error"/>
         <Unsigned value="21" name="inputPhaseDCRipple"
          displayName="Input Phase / DC Ripple"/>
         <Unsigned value="22" name="internalDriveFailure"
          displayName="Internal Drive Failure"/>
        </NamedValues>
       </Enumerated>
      </Target>
     </Sequence>

     <Sequence name="15">
      <Target>
       <Real name="direct-setpoint" displayName="Direct Setpoint"
        writable="true" volatility="nonvolatile"
        variability="operational-setting" minimum="0.0" maximum="100.0"
        units="percent"/>
      </Target>
     </Sequence>

     <Sequence name="16">
      <Target>
       <Real name="pid-setpoint" displayName="PID Setpoint"
        writable="true" volatility="nonvolatile"
        variability="operational-setting" minimum="0.0" maximum="100.0"
        units="percent"/>
       </Target>
      </Sequence>
     </Array>
    </Object>
   </Definitions>
</CSML>
```

[Insert new **Clause Q.3.1.X**, p.945]

Q.3.1.X 'deprecated'

This optional attribute, of type xs:boolean, used only in definitions, indicates that an element may be present in instances of that definition, but should be ignored.

The default value of this attribute is "false".