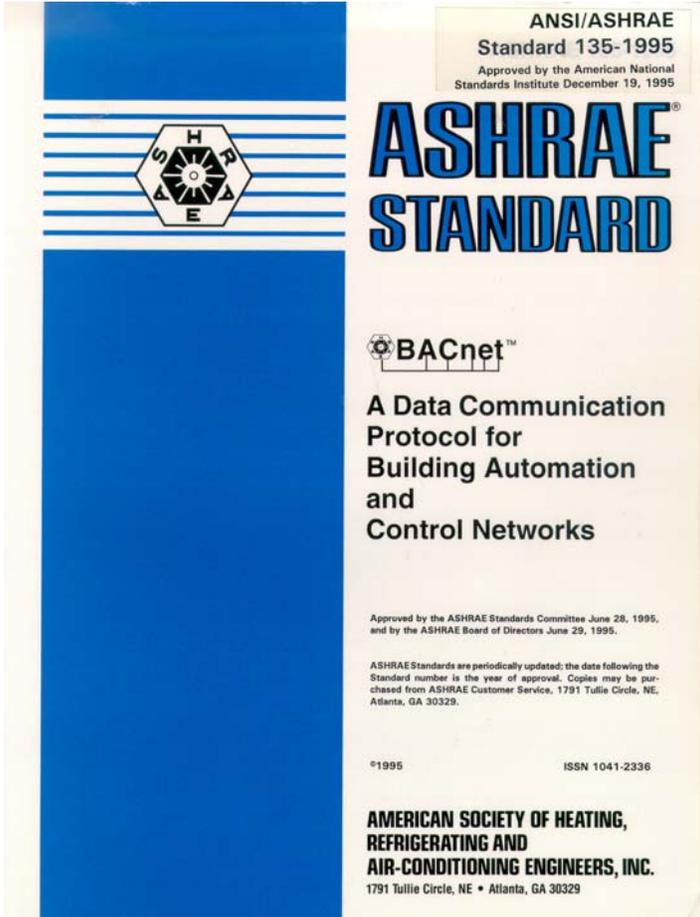


Final Report Compilation for Extending BACnet for Lighting and Utility Interfacing



TECHNICAL REPORT

October 2003
P-500-03-096-A11



Gray Davis, Governor

CALIFORNIA ENERGY COMMISSION

Prepared By:

Architectural Energy Corporation
Vernon A. Smith
Boulder, CO

*National Institute of Standards and
Technology*

Steven Bushby
Gaithersburg, MD

CEC Contract No. 400-99-011

Prepared For:

Christopher Scruton
Contract Manager

Nancy Jenkins

PIER Buildings Program Manager

Terry Surles

PIER Program Director

Robert L. Therkelsen

Executive Director

DISCLAIMER

This report was prepared as the result of work sponsored by the California Energy Commission. It does not necessarily represent the views of the Energy Commission, its employees or the State of California. The Energy Commission, the State of California, its employees, contractors and subcontractors make no warrant, express or implied, and assume no legal liability for the information in this report; nor does any party represent that the uses of this information will not infringe upon privately owned rights. This report has not been approved or disapproved by the California Energy Commission nor has the California Energy Commission passed upon the accuracy or adequacy of the information in this report.

Acknowledgements

Steven Bushby, David Holmberg, and Stephen Treado with NIST conducted this research project.

Preface

The Public Interest Energy Research (PIER) Program supports public interest energy research and development that will help improve the quality of life in California by bringing environmentally safe, affordable, and reliable energy services and products to the marketplace.

The Program's final report and its attachments are intended to provide a complete record of the objectives, methods, findings and accomplishments of the Energy Efficient and Affordable Commercial and Residential Buildings Program. This attachment is a compilation of reports from Project 3.4, Extending BACnet for Lighting and Utility Interfacing, providing supplemental information to the final report (Commission publication #P500-03-096). The reports, and particularly the attachments, are highly applicable to architects, designers, contractors, building owners and operators, manufacturers, researchers, and the energy efficiency community.

This document is one of 17 technical attachments to the final report, consisting of the final research report from Project 3.4:

- [*Proposed Amendments for Extending the BACnet Standard to Include Lighting Control and Interfacing Building Systems with Utilities. \(May 2003\)*](#)

The Buildings Program Area within the Public Interest Energy Research (PIER) Program produced this document as part of a multi-project programmatic contract (#400-99-011). The Buildings Program includes new and existing buildings in both the residential and the nonresidential sectors. The program seeks to decrease building energy use through research that will develop or improve energy-efficient technologies, strategies, tools, and building performance evaluation methods.

For the final report, other attachments or reports produced within this contract, or to obtain more information on the PIER Program, please visit www.energy.ca.gov/pier/buildings or contact the Commission's Publications Unit at 916-654-5200. The reports and attachments, as well as the individual research reports, are also available at www.archenergy.com.

Abstract

Project 3.4, *Extending BACnet for Lighting and Utility Interfacing,*

This project focused on developing software objects for the BACnet HVAC standard to include lighting controls and utility meters. The concept was to use the BACnet communications protocol to promote an open (as contrasted with proprietary) control scheme that would include lighting and energy meters as well as HVAC. NIST worked closely with the ASHRAE standards committee responsible for the BACnet standard and a number of international organizations to create consensus for the scope and functionality of the proposed objects. Progress was made in both areas.

- Two lighting control features, one to allow grouping of lighting control commands and one to interface to the DALI lighting protocol, should be part of the BACnet standard before the end of 2003.
- Two utility interface features related to remote meter reading were recently published for public comment.
- Integration of lighting controls and utility meters into the BACnet standard will promote energy conservation by giving building operators the opportunity to work on a single controls platform.

This document includes the final technical report from the research.

Task Report for the

**Energy Efficient and Affordable Small
Commercial and Residential Buildings
Research Program**

A Public Interest Energy Research Program

Sponsored by the California Energy Commission

**Project 3.4 – Extending BACnet for Lighting
Control and Interfacing Building
Systems with Utilities**

Task 3.4.3 – Finalize and Propose BACnet Amendments

Steven T. Bushby, Mechanical Systems and Controls Group

David G. Holmberg, Mechanical Systems and Controls Group

Stephen J. Treado, Mechanical Systems and Controls Group

May 27, 2003

Prepared for
Architectural Energy Corporation



National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Table of Contents

Executive Summary	1
Lighting Applications Progress Update	2
Utilities Interactions Progress Update	2
Draft Lighting Objects	5
Attachment A: Multiplexer Object	5
Attachment B: Lighting Control (DALI) Object	10
Draft Utility Integration Objects	12
Attachment C: Accumulator Object	12
Attachment D: Pulse Converter Object	27
Attachment E: Load Control Object	40
Attachment F: RTP Encoding	56
Attachment G: Logical Entity Object	60

Executive Summary

This report on Task 3.4.3 – Finalize and Propose BACnet Amendments – summarizes the accomplishments in extending the capabilities of the BACnet communication protocol to meet the needs of lighting control systems and interactions between building automation systems and utility companies.

The BACnet communication protocol for building automation and control systems was developed by ASHRAE and is now an ANSI/ASHRAE standard, a Korean national standard, and a proposed ISO standard. It has been translated into Korean, Chinese, and Japanese. It has also been endorsed by the fire alarm industry in the United States as the preferred method of integrating fire systems with other building control systems. The BACnet standard is maintained by a standing ASHRAE committee under "continuous maintenance" procedures. This provides a mechanism for proposing enhancements to the standard at any time.

Although BACnet has become well established in the commercial heating ventilating, and air conditioning (HVAC) controls market and is growing in acceptance for lighting controls, fire alarm systems, and access control systems, features specifically designed for lighting control applications and for communication between the building and utility providers are still under development.

One goal of this effort was to establish a working liaison between lighting control experts and the ASHRAE BACnet committee for the purpose of identifying additions or changes to the BACnet standard that should be made to improve its ability to meet the needs of the lighting control industry. Although BACnet is believed to be well suited for lighting control applications there were no lighting control experts directly involved in its development. A Lighting Applications Working Group (LA-WG) was formed in ASHRAE SSPC 135 to provide a mechanism for building an industry consensus for action. Companies currently making BACnet lighting control systems participate directly in the working group. Liaison relationships have been established with the National Electrical Manufacturers Association (NEMA) and with the lighting controls committee of the Illuminating Engineering Society of North America (IESNA).

The BACnet Lighting Working Group has conducted several workshops to solicit comments and other input from interested stakeholders, as well as a number of working meetings to coalesce the information and develop approaches and methods for implementing the enhancements to accommodate lighting control. Two new draft lighting objects have been developed and are in the committee review process. These are a multiplexer object type and a lighting control object type. It is believed that the addition of these two objects will substantially improve the potential for using BACnet for lighting control.

Another goal of this effort was to develop proposed enhancements to the BACnet protocol that can facilitate the exchange of information between utilities and building systems. A Utilities Interactions Working Group (UI-WG) was formed in ASHRAE SSPC 135 to provide a mechanism for building an industry consensus for action. The UI-WG has established close ties to interested parties in Europe through a liaison to Committee for European Normalization Technical Committee 247 (CEN TC 247), and in Japan through a liaison with the Institute of Electrical Installation Engineers of Japan (IEIEJ). This international cooperation makes it likely that the end product will become part of an international standard. The involvement of building automation system manufacturers makes it likely that the changes will be implemented in commercial products.

The UI-WG developed two new object types that provide a simple interface to utility meters with a pulsed output. These objects have been published for public review and comment. The UI-WG has made significant progress on the proposed Load Control Object (LCO) which allows demand limiting signals to

be received from the utility and sent to devices on the network. There has also been progress made on defining the transport from utility to building—the Logical Entity Object (LEO), which is part of a proposed utility-to-building interface that fits in with work proceeding in the XML working group, has been drafted and reviewed. The issue of how to best link to the utility, really a business to business data transfer issue, is being hotly contested with the SSPC. It is likely that the utility connection will be the first application requiring communication with a non-BACnet speaking outside partner which will require transport services that are still under debate. The Electric Power Research Institute (EPRI) is also working on these issues within their Integrated Energy and Communications Systems Architecture (IECSA) project that hopefully the UI-WG can give some input on. There has also been a project working with EPRI to demonstrate a Real Time Pricing (RTP) signal from the utility to a building customer with feedback from the electric meter. This project has served to define the RTP encoding in BACnet and has given some definition to the LEO.

Lighting Applications Progress Update

The initial efforts by the BACnet Lighting Working Group involved identifying and consolidating typical tasks for and requirements of building lighting control systems. Existing BACnet capabilities were examined to determine which lighting control functions could be implemented using current objects and services, and which, if any, would require or substantially benefit from enhancements or additions to the BACnet protocol. This activity resulted in the identification of a number of lighting control functions that could not be easily implemented in BACnet, if at all, due to insufficient functionality or excessive network traffic implications. In particular, lighting controllers typically use a group of writable objects as a way of simplifying control, and as a way of reducing network traffic. These are known as groups, zones, or areas, depending on the manufacturer.

Grouping a set of outputs allows control of areas, such as hallways or rooms. This makes setting up a system easier, since the controller only needs to know about adjusting the lights in an area, such as a hallway or a room, and does not need to know that output 1 and output 3 on the load panel control the hallway lights. When a system of lighting controllers is used, they are typically networked together. When multiple networked controllers attempt to control lights in unison in a single area, and the network contains any delays, there may be noticeable aberrations in the lighting to the occupants of the area. The LA-WG has drafted a Multiplexer Object Type to address these needs, and the current revision of this object is presented in Attachment A.

Another desirable enhancement to BACnet is a lighting control object that would allow dimming, switching and ramping, and which would be compatible with the emerging DALI standard for digital electronic ballasts. This object would provide network visibility and controllability for individual lighting fixtures. A new draft Lighting Control (DALI) object has been developed and, along with the Multiplexer Object, is in the BACnet review process to address the above issues (see Attachment B). These objects are expected to be approved as new addenda before the end of the calendar year.

Utilities Interactions Progress Update

In June of 2000 ASHRAE SSPC 135 formed the Utility Interactions Working Group (UI-WG) to deliberate proposed enhancements to BACnet that would allow communication between building automation and control systems and utility providers. As part of this activity an effort has been made to bring electric utility experts into the process and to engage international standards organizations that expressed an interest in working with ASHRAE in this area. Connections with the US utility industry have been made through the Electric Power Research Institute (EPRI). EPRI is working on utility industry communication standards and addressing utility-building connectivity. The international connections are through liaisons to Committee for European Normalization Technical Committee 247 (CEN TC 247) and the Institute of Electrical Installation Engineers of Japan (IEIEJ). Both of these

organizations are involved with the adoption of BACnet within their domain and also as a world standard through the International Organization for Standardization (ISO). CEN and the IEIEJ have expressed an urgent need for new BACnet capabilities to interface to utility meters.

Early discussions revealed multiple applications for communication with electric utilities and utility electric meters. The final report from ASHRAE research project 1011, Utility/Energy Management and Control Systems (EMCS) Communication Protocol Requirements, presented many potential points for utility-EMCS communication. Of these, the applications that were identified as most pressing are billing management and peak load management. A consensus emerged that the best way to proceed was to begin by defining a new BACnet object or objects that represent the information exchange needed for peak load management and billing management.

Proposals for new BACnet objects to meet these applications have passed through several rounds of iteration and discussion. The initially developed meter object saw multiple iterations, and finally settled as two separate objects: an "Accumulator Object" and a "Pulse Converter Object". Together these two objects meet most of the needs of both US and international participants in providing a simple functionality to allow building control systems to read the pulse from an electric utility meter. This in turn provides billing and power management capabilities to the building control system. These two objects were approved at the ASHRAE June 2002 meeting by SSPC 135 for inclusion in the next BACnet addendum for public review. This has since been incorporated in Addendum C and is presently in public review. There have been some changes, and the latest versions are provided as Attachments C and D.

Work has also been progressing on a BACnet object to allow peak load management. This Load Control Object (LCO) is still under continued deliberation within the UI-WG but will hopefully be released for public review this year. The object allows for a command to be sent by a utility (or building master controller) to direct a building controller or equipment controller to reduce load. LCO objects can be nested to provide management flexibility and also a way to scale up to large complex systems. The current version is provided as Attachment E.

At the June 2002 meeting, the BACnet committee also identified the need to better understand the utility industry meter standard ANSI C12.19. A subset of the UI-WG was selected to review the C12.19 standard and identify what work has already been done that might be adapted to allow a more enhanced BACnet/ meter interface, providing functionality beyond what is provided in the simple Accumulator and Pulse Converter objects. The early effort to develop the Accumulator and Pulse Converter objects provides for simple functionality useful throughout the world, while additional developments will probably be more regional in scope (as with the US ANSI C12.19 meter standard). The issue at hand presently is how best to interface to the meter—via what protocol and network transport, with what BACnet interface: a BACnet C12 Meter object, or via a standardized data structure that would allow access to the entire meter tables, or a data structure that would provide only a subset of the data. Work continues to address the best method to interface meters to the building control system and how to represent the meter data within BACnet.

A third utility-building service identified for implementation is Real Time Pricing (RTP). This service is outlined in the report cited earlier, RP-1011, and in the past year this service has been seen significant efforts made on the EPRI side. Dr. Martin Burns has prepared an XML schema of the RTP data structure in an effort to address the utility-to-building transport issue. In addition, NIST has worked with him on a demonstration project to demonstrate communication between a utility and meter, and between utility and building control system. NIST has prepared a BACnet object, the Logical Entity object (LEO), that may be used in the interface between utility and building, and was used in the demonstration project. A table was prepared demonstrating the transformation from the utility data structure to the BACnet encoding

required in the LEO. This table is provided as attachment F along with the RTP XML schema. The current version of the LEO is provided at attachment G.

The international involvement in the committee's activity makes it likely that the end products will become an international standard. The involvement of building automation system manufacturers and utility representatives makes it likely that the changes will be implemented in commercial products.

Draft Lighting Objects

Attachment A: Multiplexer Object

Multiplexing a Writable Property

References:

ANSI/ASHRAE Standard 135-2001, "BACnet[®], A Data Communication Protocol for Building Automation and Control Networks"

Background

Lighting controllers typically use a group of writable objects as a way of simplifying control, and as a way of reducing network traffic. These are known as groups, zones, or areas, depending on the manufacturer.

Grouping a set of outputs allows control of areas, such as hallways or rooms. This makes setting up a system easier, since the controller only needs to know about adjusting the lights in an area, such as a hallway or a room, and does not need to know that output 1 and output 3 on the load panel control the hallway lights.

When a system of lighting controllers is used, they are typically networked together. When multiple networked controllers attempt to control lights in unison in a single area, and the network contains any delays, there may be noticeable aberrations in the lighting to the occupants of the area.¹

Grouping in BACnet

BACnet supports two ways to perform grouping. These are the Group Object for reading a set of values from a group of object properties, and the Command Object for writing a set of values to a group of object properties based on the action code.

The standard explicitly states that the Group Object "is a read only property; it cannot be used to write a set of values to the members of the group." Therefore, it cannot be used for our purposes.

The command object seems useful, but it requires an action code (index) be written to it so that it will perform predefined commands for each code. For lighting we could use indices 0 to 100 to represent percentage levels 0% to 100%, and other indices to represent stepping commands. For simple lighting groups, this would work great. However, when we add dimmers into the mix, and want to pass a fade time, this no longer works. If we use relays and want to pass a timeout value (time until relinquish) along with the level, this no longer works.

Perhaps the best solution to sending multiple property requests is BACnet's WritePropertyMultiple service. Since the properties must be modified in the order specified, there would not be any problems with atomicity when the sequence of the properties is critical². A "feature" of WPM that makes it such that it would not fulfill the purpose outlined here is that when executing a WritePropertyMultiple, only properties up to the first failed property are written. In this particular use case, stopping subsequent property writes due to errors is not desirable. Another drawback of using WritePropertyMultiple is that simple devices may not have that service available to them.

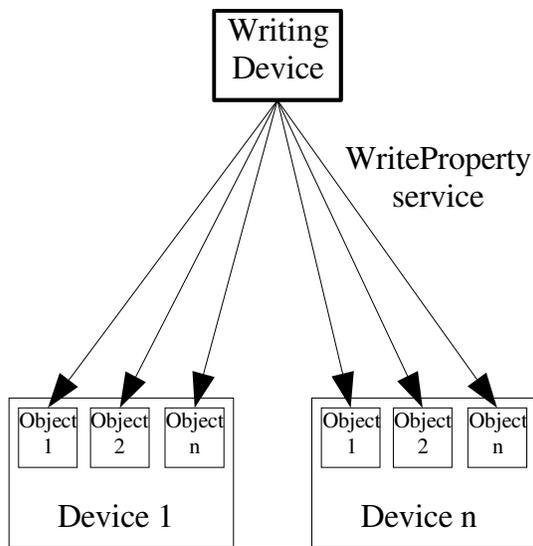
¹I had this problem at a theater/restaurant in Los Angeles. There were four lighting panels networked together using BACnet over ARCNET 156kbps over EIA-485. The house lights in the stage area were controlled by dimmer outputs from each of the four panels. The control algorithm was such that a switch input would send network messages to each output that it controlled. When the sound booth light switch was used to raise or lower the lights, one side of the room would start to raise, and then the other side of the room would start to raise. By the time the lights were adjusted, they were all at different levels. Each press and release of the button resulted in hundreds of network messages. The solution was to limit the amount of network traffic. I successfully changed the control algorithm to only send 3 remote messages – one to each remote cabinet – and the message was sent to the object that represented a group of dimmers in that cabinet.

²Sending the level of a light along with the fade time (amount of time duration for the light to go from its current level to the new lighting level) is an example of a critical property writing sequence. If the level is written before the fade time, the light would not respond in the desired manner.

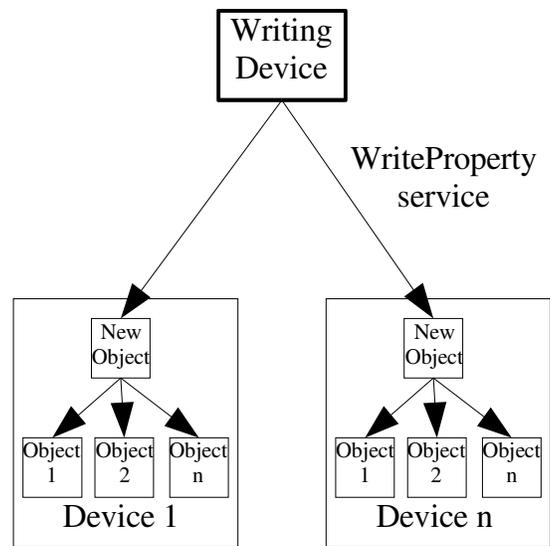
Writing to Multiple Objects

A new BACnet object that is based on the internal workings of the schedule object could help to solve the problem of writing a single value to multiple objects and properties. The new object could also help significantly reduce network traffic. The new object can be thought of as a writable Group object. It would allow any value written to the new object to be written to each of the objects in its member list. Since the WriteProperty service has an optional priority parameter, the priority can also be passed to the objects in its member list. Although this object in itself would not solve the multiple property problem, we should be able to use this object in conjunction with WritePropertyMultiple.

An issue that is unclear is whether a group of objects may be formed by using any combination of object types. This increases the complexity of the internal workings of this object, since it will have to determine how to convert properties that contain disparate datatypes. The most common case of this is the Present_Value property. For an Analog Output and Analog Value object, the Present_Value property is defined as a REAL. For a Binary Output object, the Present_Value property is defined as a BACnetBinaryPV. For a Multi-state Output object, the Present_Value property is defined as an Unsigned. This problem is similar to the Present_Value property in the Schedule object, which simply restricts the property to any primitive datatype.



Existing method of writing the same value to multiple objects in the same device using WriteProperty services.



Proposed method of writing the same value to multiple objects in the same device using WriteProperty services.

[add to 12, pp.127-232]

12.X Multiplexer Object Type

The Multiplexer object type defines a standardized object whose properties represent a collection of other objects properties. A Multiplexer object is used to simplify the exchange of information between BACnet Devices by providing a shorthand way to specify all members of a group at once. A group may be formed using any combination of object types. The Multiplexer object and its properties are summarized in Table 12-X and described in detail in this subclause.

This object uses the 'Priority' parameter of the WriteProperty service to command the referenced properties.

Table 12-X. Properties of the Multiplexer Object Type

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Any	W
Description	CharacterString	O
List_Of_Object_Property_References	List of BACnetDeviceObjectPropertyReference	R
Profile_Name	CharacterString	O

Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object-type class. The value of this property shall be Multiplexer.

Present_Value (Commandable)

This property indicates the current value of the Multiplexer object, or of the objects in the List_Of_Object_Property_References, i.e., the value most recently written to the referenced object property of all members of the List_Of_Object_Property_References. This may be any primitive datatype. As a result, analog, binary, and enumerated values may be written. If the List_Of_Object_Property_References is empty, then the value of this property will be that which would have been most recently written to the List_Of_Object_Property_References.

Although this property is commandable, there is no corresponding Priority_Array or Relinquish_Default property. The values that are written to the Present_Value property are passed to the properties listed in the List_Of_Object_Property_References. If the value is written to the Present_Value multiple times, the value gets passed to the listed properties multiple times.

The initial value of the Present_Value property is NULL. This value does not automatically get written to the properties listed in the List_Of_Object_Property_References.

Description

This property is a string of printable characters whose content is not restricted.

List_Of_Object_Property_Reference

This property specifies the Device Identifiers, Object Identifiers and Property Identifiers of the properties to be written with the Present_Value property when the Present_Value property is written.

If this property is writable, it may be restricted to only support references to objects inside of the device containing the Multiplexer object. If the property is restricted to referencing objects within the containing device, an attempt to write a reference to an object outside the containing device into this property shall cause a Result(-) to be returned with an error class of PROPERTY and an error code of OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED.

If this property is set to reference an object outside the device containing the Multiplexer object, the method used for writing to the referenced property value for the purpose of controlling the property is a local matter. The only restriction on the method of writing to the referenced property is that the Multiplexer device be capable of using WriteProperty for this purpose so as to be interoperable with all BACnet devices.

Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

[add to 19, p.342]

19.2.1.1 Commandable Properties

The prioritization scheme is applied to certain properties of objects. The standard commandable properties and objects are as follows:

<u>OBJECT</u>	<u>COMMANDABLE PROPERTY</u>
Analog Output	Present_Value
Binary Output	Present_Value
Multi-state Output	Present_Value
Multi-state value	Present_Value
Analog Value	Present_Value
Binary Value	Present_Value
<i>Multiplexer</i>	<i>Present_Value</i>

The designated property of these objects is commandable (prioritized) by definition. Individual vendors, however, may decide to apply prioritization to any of the vendor-specified properties. These additional commandable properties shall have associated Priority_Array and Relinquish_Default properties with appropriate names. See 23.3.

[add to 21, BACnetObjectType pp.397-398]

```

BACnetObjectType ::= ENUMERATED {
    ...
    multi-state-value          (19),
    multiplexer                (n)
    notification-class         (15),
    ...
    -- see life-safety-zone    (22),
    -- see multiplexer         (n)
    ...
}

```

-- Enumerated values 0-127 are reserved for definition by ASHRAE. Enumerated values
 -- 128-1023 may be used by others subject to the procedures and constraints described
 -- in Clause 23.

```

BACnetObjectTypesSupported ::= BIT STRING {
    ...
    -- multi-state-value      (19),
    multiplexer                (n),
    notification-class        (15),
    program                    (16),
}

```

```

    schedule                (17),
    -- trend-log            (20),
-- Objects added after 1995
    averaging               (18),
    multi-state-value       (19),
    trend-log               (20)
    life-safety-point       (21),
    life-safety-zone        (22),
    multiplexer             (n),
    }

```

[add to ANNEX C pp.428-438]

```

MULTIPLEXER ::= SEQUENCE {
    object-identifier       [75] BACnetObjectIdentifier,
    object-name              [77] CharacterString,
    object-type              [79] BACnetObjectType,
    present-value            [85] ABSTRACT-SYNTAX.&Type, -- Any datatype
    description              [28] CharacterString OPTIONAL,
    list-of-object-property-references [54] SEQUENCE OF BACnetDeviceObjectPropertyReference,
    profile-name             [167] CharacterString OPTIONAL
}

```

[add to ANNEX D pp.439-456]

ANNEX D - EXAMPLES OF STANDARD OBJECT TYPES (INFORMATIVE)

D.X Example of a Multiplexer Object

The following is an example of a Multiplexer object that is used for controlling a group of Binary Output objects.

```

Property:    Object_Identifier = (Multiplexer, Instance 3)
Property:    Object_Name = "Lighting Group 3"
Property:    Object_Type = MULTIPLEXER
Property:    Present_Value = 100
Property:    Description = "Hallway Lights Floor 3 Building 1"
Property:    List_Of_Object_Property_References = (((Device, Instance 12),(Binary Output, Instance 9),
Present_Value)
                                                    ((Device, Instance 12),(Binary Output, Instance 8),
Present_Value)
                                                    ((Device, Instance 12),(Binary Output, Instance 7),
Present_Value))

```

Attachment B: Lighting Control (DALI) Object

Following is the partial object definition of the DALI object. The conformance code column (Optional, Required) was not discussed at this meeting and will need to be reviewed at a future meeting.

Table 12-x. Properties of the DALI Object Type

Present_Value	BACnetLightingState	W
Device_Type	CharacterString	O
Luminance	REAL	O

12.x.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.x.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.x.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be DALI.

12.x.4 Present_Value

This property, of type BACnetLightingState, defines the current commanded state of the DALI object as defined below:

Command: Description

 Power Off: Cuts electrical power to the device. The last command and its priority are saved for use when the device is later Powered On.

Power On: Applies electrical power to the device setting its luminance to the last commanded value.

Ramp Up: Raises the luminance level at a given rate. The ramp up continues until another command is received or it reaches its maximum luminance. Ramp up can be applied with Power Off or Power On. When Powered Off the resulting luminance level is saved for use when the device is later Powered On.

Ramp Down: Lowers the luminance level at a given rate. The ramp down continues until another command is received or it reaches its minimum luminance. Ramp down can be applied with Power Off or Power On. When Powered Off the resulting luminance level is saved for use when the device is later Powered On.

Step Up: The luminance level is stepped up by a percent of the fade rate. The step up continues until another command is received or it

reaches its maximum luminance. Step up can be applied with Power Off or Power On. When Powered Off the resulting luminance level is saved for use when the device is later Powered On.

Step Down: The luminance level is stepped down by a percent of the fade rate. The step down continues until another command is received or it reaches its minimum luminance. Step down can be applied with Power Off or Power On. When Powered Off the resulting luminance level is saved for use when the device is later Powered On. If minimum luminance is reached, the light remains Powered On.

Recall Maximum: Power is applied to the device if not already applied. and set to maximum luminance.

Recall Minimum: Power is applied to the device if not already applied. and set to minimum luminance.

Step Down + Off: The luminance level is stepped down by a percent of the fade rate. The step down continues until another command is received or it reaches its minimum luminance. Step down can be applied with Power Off or Power On. When Powered Off the resulting luminance level is saved for use when the device is later Powered On. If minimum luminance is reached during step down, the light is Powered Off.

On + Step Up: Power is applied to the light if not already applied. The luminance level is then stepped up by a percent of the fade rate. The step up continues until another command is received or it reaches its maximum luminance.

Go to Level: Power is applied to the light if not already applied.. The luminance level is then set as a percentage 0 to 100. If the level set is 0 the light remains Powered On.

Hold Level: Stops further level change for all commands.

Blink: If the device is Powered On, it turns Power Off for the warning period. Power is then reapplied at the same luminance level prior to the Blink command.

On + Ramp Up: Power is applied to the light if not already applied. The luminance level is then ramped up from the minimum luminance by a percent of the fade rate. The ramp up continues until another command is received or it reaches its maximum luminance.

12.x.6 Device_Type

This property, of type CharacterString, is a text description of the physical device connected to the analog output. It will typically be used to describe the type of device attached to the analog output.

12.x.7 Luminance

This property, of type REAL, indicates the actual value, in engineering units, of the output. This value is the linearized percentage (0..100) of the device's light output (LUMENS) of the object; 0 being dimmest, 100 brightest.

Draft Utility Integration Objects

Attachment C: Accumulator Object

135c-4. Add a new Accumulator Object Type.

Rationale

There is need for a standard object to represent inputs indicating various measured values by means of a pulsed output for applications ranging from system monitoring and control to billing. Many such applications use small, limited controllers incapable of performing floating-point calculations. This factor was considered in the development of this object type.

Addendum 135c-4

[Add a new **Clause 12.1**, p. 130, and renumber the existing Clause 12.1 and subsequent clauses, including tables and figures]

12.1 Accumulator Object Type

The Accumulator object type defines a standardized object whose properties represent the externally visible characteristics of a device that indicates measurements made by counting pulses.

This object maintains precise measurement of input count values, accumulated over time. The accumulation of pulses represents the measured quantity in unsigned integer units. This object is also concerned with the accurate representation of values presented on meter read-outs. This includes the ability to initially set the Present_Value property to the value currently displayed by the meter (as when the meter is installed), and to duplicate the means by which it is advanced, including simulating a modulo-N divider prescaling the actual meter display value, as shown in Figure 12-1.

Typical applications of such devices are in peak load management and in accounting and billing management systems. This object is not intended to meet all such applications. Its purpose is to provide information about the quantity being measured, such as electric power, water, or natural gas usage, according to criteria specific to the application.

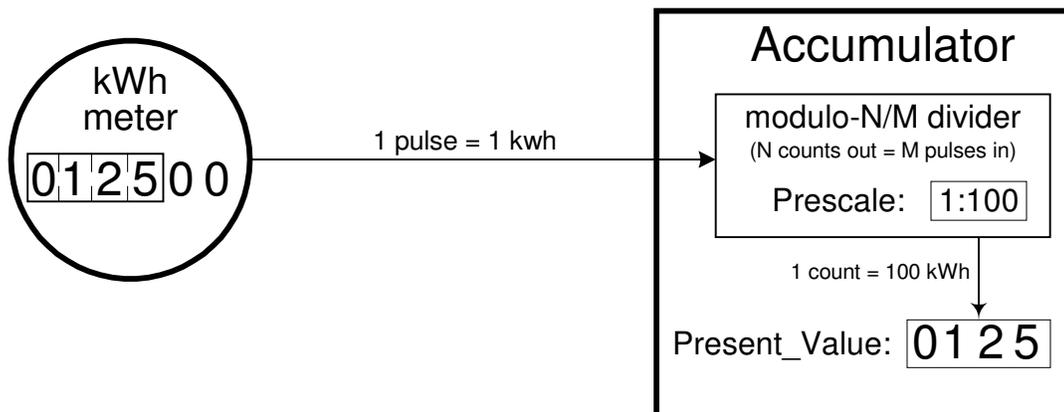


Figure 12-1. Example of an Accumulator object

The object and its properties are summarized in Table 12-1 and described in detail in this subclause. [Tom, I rejected this change because it would make the language inconsistent with dozens of similar cases in Clause 12. I made the same decision in the other instances where you made this change.]

Table 12-1. Properties of the Accumulator Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Present_Value	Unsigned	R ¹
Description	CharacterString	O
Device_Type	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Scale	BACnetScale	R
Units	BACnetEngineeringUnits	R
Prescale	BACnetPrescale	O
Max_Pres_Value	Unsigned	R
Value_Change_Time	BACnetDateTime	O ²
Value_Before_Change	Unsigned	O ^{2,3}
Value_Set	Unsigned	O ^{2,3}
Logging_Record	BACnetAccumulatorRecord	O
Logging_Object	BACnetObjectIdentifier	O
Pulse_Rate	Unsigned	O ^{1,4}
High_Limit	Unsigned	O ⁴
Low_Limit	Unsigned	O ⁴
Limit_Monitoring_Interval	Unsigned	O ⁴
Notification_Class	Unsigned	O ⁴
Time_Delay	Unsigned	O ⁴
Limit_Enable	BACnetLimitEnable	O ⁴
Event_Enable	BACnetEventTransitionBits	O ⁴
Acked_Transitions	BACnetEventTransitionBits	O ⁴
Notify_Type	BACnetNotifyType	O ⁴
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O ⁴
Profile_Name	CharacterString	O

¹ This property is required to be writable when Out_Of_Service is TRUE.

² These properties are required if either Value_Before_Change or Value_Set is writable.

³ Either Value_Before_Change or Value_Set may be writable, but not both.

⁴ These properties are required if the object supports intrinsic reporting.

12.1.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.1.2 Object_Name

This property, of type `CharacterString`, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the `Object_Name` shall be restricted to printable characters.

12.1.3 Object_Type

This property, of type `BACnetObjectType`, indicates membership in a particular object type class. The value of this property shall be `ACCUMULATOR`.

12.1.4 Present_Value

This property, of type `Unsigned`, indicates the count of the input pulses, prescaled if the `Prescale` property is present, acquired since the value was most recently set by writing to the `Value_Set` property.

The value of this property shall remain in the range from zero through `Max_Pres_Value`. All operations on the `Present_Value` property are performed modulo $(\text{Max_Pres_Value}+1)$.

This property shall be writable when `Out_Of_Service` is `TRUE`.

12.1.5 Description

This property, of type `CharacterString`, is a string of printable characters whose content is not restricted.

12.1.6 Device_Type

This property, of type `CharacterString`, is a text description of the physical device represented by the Accumulator object. It will typically be used to describe the type of sensor represented by the Accumulator.

12.1.7 Status_Flags

This property, of type `BACnetStatusFlags`, represents four Boolean flags that indicate the general "health" of an Accumulator object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{`IN_ALARM`, `FAULT`, `OVERRIDDEN`, `OUT_OF_SERVICE`}

where:

`IN_ALARM` Logical FALSE (0) if the `Event_State` property has a value of `NORMAL`, otherwise logical TRUE (1).

`FAULT` Logical TRUE (1) if the `Reliability` property is present and does not have a value of `NO_FAULT_DETECTED`, otherwise logical FALSE (0).

`OVERRIDDEN` Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the `Present_Value` and `Reliability` properties are no longer tracking changes to the physical input. Otherwise, the value is logical FALSE (0).

`OUT_OF_SERVICE` Logical TRUE (1) if the `Out_Of_Service` property has a value of `TRUE`, otherwise logical FALSE (0).

12.1.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting and if the Reliability property is not present, then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be “fault” events.

12.1.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value property or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER }

12.1.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the physical input that the object represents is not in service. This means that the Present_Value and Pulse_Rate properties are decoupled from the physical input and will not track changes to the physical input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the physical input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value, Pulse_Rate and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value, Pulse_Rate or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the physical input.

12.1.11 Scale

This property, of type BACnetScale, indicates the conversion factor to be multiplied with the value of the Present_Value property to provide a value in the units indicated by Units. The choice of options for this property determine how the scaling operation (which is performed by the client reading this object) is performed:

<u>Option</u>	<u>Datatype</u>	<u>Indicated Value in Units</u>
floatScale	REAL	Present_Value x Scale
integerScale	INTEGER	Present_Value x 10 ^{Scale}

12.1.12 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of the Present_Value when multiplied with the scaling factor indicated by Scale. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.1.13 Prescale

This property, of type BACnetPrescale, presents the coefficients that are used for converting the pulse signals generated by the measuring instrument into the value displayed by Present_Value. The conversions are performed using integer arithmetic in such a fashion that no measurement-generated pulse signals are lost in the conversion.

These coefficients might simply document a conversion performed prior to the reception of the input pulses by the Accumulator object, or they might actually be used by the Accumulator to convert input pulses into the value displayed by Present_Value. Whichever is done is a local matter.

The coefficients are as follows:

multiplier The numerator of the conversion factor expressed as a ratio of integers.
moduloDivide The denominator of the conversion factor expressed as a ratio of integers.

The conversion algorithm is performed as follows, utilizing a non-displayed variable called an accumulator:

For each input pulse:

 Add the value of 'multiplier' to an accumulator and then,
 while the accumulator is greater than or equal to the value of 'moduloDivide':
 Increment the value of Present_Value by one, and
 decrease the value of the accumulator by the value of 'moduloDivide'.

This procedure supports non-integral ratios of measurement pulses to Present_Value. For example, in an electrical metering application, the output of the voltage- and current-measuring systems might be 9000/1200 (scale / voltage*current) pulses per kWh, requiring the Accumulator object to accumulate 2/15 kWh/pulse. With this algorithm such pulses can be accurately accumulated and displayed when the units of Present_Value are KILOWATT_HOURS.

12.1.14 Max_Pres_Value

This property, of type Unsigned, indicates the maximum value of the Present_Value property.

12.1.15 Value_Change_Time

This read-only property, of type BACnetDateTime, shall be present if the Present_Value property is adjustable by writing to the Value_Before_Change or Value_Set properties. It represents the date and time of the most recent occurrence of such a write operation. If no such write has yet occurred, this property shall have wildcard values for all date and time fields.

12.1.16 Value_Before_Change

This property, of type Unsigned, indicates the value of the Present_Value property just prior to the most recent write to the Value_Set or Value_Before_Change properties. If no such write has yet occurred, this property shall have the value zero. If this property is writable, the Value_Set property shall be read-only.

If this property is writable, the following series of operations, for which the associated properties are present, shall be performed atomically by the object when this property is written:

- (1) The value of Present_Value shall be copied to the Value_Set property.
- (2) The value written to Value_Before_Change shall be stored in the Value_Before_Change property.
- (3) The current date and time shall be stored in the Value_Change_Time property.

While this series of operations is being performed, it is critical that any other process not change the Present_Value, Value_Set and Value_Before_Change properties.

12.1.17 Value_Set

This property, of type Unsigned, indicates the value of the Present_Value property after the most recent write to the Value_Set or Value_Before_Change properties. If no such write has yet occurred, this property shall have the value zero. If this property is writable, the Value_Before_Change property shall be read-only.

If this property is writable, the following series of operations, for which the associated properties are present, shall be performed atomically by the object when this property is written:

- (1) The value of Present_Value shall be copied to the Value_Before_Change property.
- (2) The value written to Value_Set shall be stored in both the Value_Set and Present_Value properties.
- (3) The current date and time shall be stored in the Value_Change_Time property.

While this series of operations are being performed, it is critical that any other process not change the Present_Value, Value_Set and Value_Before_Change properties.

12.1.18 Logging_Record

This read-only property, of type BACnetAccumulatorRecord, is a list of values that must be acquired and returned “atomically” in order to allow proper interpretation of the data.

If the Logging_Object property is present, then, when Logging_Record is acquired by the object identified by Logging_Object, this list of values shall be saved and returned when read by other objects or devices. If the Logging_Object property is present and Logging_Record has not yet been acquired by the object identified by Logging_Object, ‘timestamp’ shall contain all wildcards, ‘present-value’ and ‘accumulated-value’ shall contain the value zero, and ‘accumulator-status’ shall indicate STARTING.

The list of values (‘timestamp’, ‘present-value’, ‘accumulated-value’, and ‘accumulator-status’) shall be acquired from the underlying system when they reflect a stable state of the device (for example, they shall not be acquired when Present_Value has just been incremented but the corresponding increment of ‘accumulated-value’ has not yet occurred).

The items returned in the list of values are:

timestamp	The local date and time when the data was acquired.
present-value	The value of the Present_Value property.
accumulated-value	The short term accumulated value of the counter. The algorithm used to calculate accumulated-value is a function of the value of accumulator-status. If this is the initial read, the value returned shall be zero.
accumulator-status	An indication of the reliability of the data in this list of values.

The accumulator-status parameter may take on any of the following values:

{NORMAL, STARTING, RECOVERED, ABNORMAL, FAILED}

where the values are defined as follows:

NORMAL No event affecting the reliability of the data has occurred during the period from the preceding to the current qualified reads of the Logging_Record property. In this case ‘accumulated-value’ shall be represented by the expression:

$$\text{(accumulated-value)} = \text{(Present_Value}_{\text{current}}) - \text{(Present_Value}_{\text{previous}})$$

STARTING This value indicates that the data in Logging_Records is either the first data to be acquired since startup by the object identified by Logging_Object (if ‘timestamp’ has non-wildcard values) or that no data has been acquired since startup by the object identified by Logging_Object (in which case ‘timestamp’ has all wildcard values).

RECOVERED One or more writes to Value_Before_Change or Value_Set have occurred since Logging_Record was acquired by the object identified by Logging_Object. For the case of a single write, ‘accumulated-value’ shall be represented by the expression:

$$\text{‘accumulated-value’} = \text{(Present_Value}_{\text{current}} - \text{Value_Set}) +$$

(Value_Before_Change – Present_Value_{previous})

ABNORMAL The accumulation has been carried out, but some unrecoverable event such as the clock's time being changed by a significant amount since Logging_Record was acquired by the object identified by Logging_Object. (How much time is considered significant shall be a local matter.)

FAILED 'accumulation-value' is not reliable due to some problem. The criteria for returning this value are a local matter.

Changes in the value of 'accumulator-status' shall occur only when the Logging_Record is acquired by the object identified by Logging_Object.

12.1.19 Logging_Object

This property, of type BACnetObjectIdentifier, indicates the object in the same device as the Accumulator object which, when it acquires Logging_Record data from the Accumulator object, shall cause the Accumulator object to acquire, present and store the data from the underlying system.

12.1.20 Pulse_Rate

This property, of type Unsigned, shall indicate the number of input pulses received during the most recent period specified by Limit_Monitoring_Interval. The mechanism that associates the input signal with the value indicated by this property is a local matter.

This property shall be writable when Out_Of_Service is TRUE.

12.1.21 High_Limit

This property, of type Unsigned, shall specify a limit that Pulse_Rate must exceed before an event is generated. This property is required if this object supports intrinsic reporting.

12.1.21.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

- (a) Pulse_Rate must exceed High_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the HighLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.1.21.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, Pulse_Rate must fall below High_Limit before a TO-NORMAL event is generated under these conditions:

- (a) Pulse_Rate must remain below High_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the HighLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-NORMAL flag must be set in the Event_Enable property.

12.1.22 Low_Limit

This property, of type Unsigned, shall specify a limit that Pulse_Rate must fall below before an event is generated. This property is required if this object supports intrinsic reporting.

12.1.22.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

- (a) Pulse_Rate must fall below Low_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the LowLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.1.22.2 Conditions for Generating a TO-NORMAL Event

Once Pulse_Rate has fallen below the Low_Limit, the Pulse_Rate must become greater than Low_Limit before a TO-NORMAL event is generated under these conditions:

- (a) Pulse_Rate must become greater than Low_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the LowLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-NORMAL flag must be set in the Event_Enable property.

12.1.23 Limit_Monitoring_Interval

This property, of type Unsigned, specifies the monitoring period in seconds for determining the value of Pulse_Rate. The use of a fixed or sliding time window for detecting pulse rate is a local matter. This property is required if this object supports intrinsic reporting.

12.1.24 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if this object supports intrinsic reporting.

12.1.25 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that Pulse_Rate must remain outside the range from Low_Limit through High_Limit, before a TO-OFFNORMAL event is generated, or within the same band before a TO-NORMAL event is generated. This property is required if this object supports intrinsic reporting.

12.1.26 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of High_Limit and Low_Limit offnormal events and their return to normal. This property is required if this object supports intrinsic reporting.

12.1.27 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Accumulator objects, transitions to the High_Limit or Low_Limit Event_States are considered to be "offnormal" events. This property is required if this object supports intrinsic reporting.

12.1.28 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgements for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Accumulator objects, transitions to High_Limit and Low_Limit Event_State are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

- (a) upon receipt of the corresponding acknowledgement;
- (b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgement is expected);
- (c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgement is expected).

This property is required if this object supports intrinsic reporting.

12.1.29 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if this object supports intrinsic reporting.

12.1.30 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have X'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if this object supports intrinsic reporting.

12.1.31 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

[Change **Table 13-2** p. 237]

Table 13-2. Standard Objects ~~that~~ That May Support Intrinsic Reporting

Object Type	Criteria	Event Type
<i>Accumulator</i>	<i>If Pulse_Rate exceeds range from Low_Limit through High_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable, OR Pulse_Rate returns to range from Low_Limit through High_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable</i>	<i>UNSIGNED_RANGE</i>
Analog Input, Analog Output, Analog Value,	If Present_Value exceeds range between High_Limit and Low_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable, OR Present_Value returns within the High_Limit – Deadband to Low_Limit + Deadband range for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable	OUT_OF_RANGE
...		

[Change **Table 13-3** p. 238]

Table 13-3. Standard Object Property Values Returned in Notifications

Object	Event Type	Notification Parameters	Referenced Object's Properties
<i>Accumulator</i>	<i>UNSIGNED_RANGE</i>	<i>Exceeding_Value</i> <i>Status_Flags</i> <i>Exceeded_Limit</i>	<i>Pulse_Rate</i> <i>Status_Flags</i> <i>Low_Limit or High_Limit</i>
Analog Input, Analog Output, Analog Value	OUT_OF_RANGE	Exceeding_Value Status_Flags Deadband Exceeded_Limit	Present_Value Status_Flags Deadband Low_Limit or High_Limit
...			

[Change **Table 13-4**, p. 238]

Table 13-4. Notification Parameters for Standard Event Types

Event Type	Notification Parameters	Description
...		
CHANGE_OF_LIFE_SAFETY	New_State New_Mode Status_Flags Operation_Expected	The new value of the referenced property The new mode of the referenced object The Status_Flags of the referenced object The next operation requested by the referenced object
<i>UNSIGNED_RANGE</i>	<i>Exceeding_Value</i> <i>Status_Flags</i> <i>Exceeded_Limit</i>	<i>The value that exceeded a limit</i> <i>The Status_Flags of the referenced object</i> <i>The limit that was exceeded</i>

[Change **Clause 13.3**, p.239]

13.3 Algorithmic Change Reporting

...

The following event type algorithms are specified in this standard because of their widespread occurrence in building automation and control systems. They are:

- (a) CHANGE_OF_BITSTRING
- (b) CHANGE_OF_STATE
- (c) CHANGE_OF_VALUE
- (d) COMMAND_FAILURE
- (e) FLOATING_LIMIT
- (f) OUT_OF_RANGE
- (g) BUFFER_READY
- (h) CHANGE_OF_LIFE_SAFETY
- (i) *UNSIGNED_RANGE*

...

[Add **Clause 13.3.9**, p. 245]

13.3.9 UNSIGNED_RANGE Algorithm

An UNSIGNED_RANGE occurs if the referenced property leaves the range of values from Low_Limit through High_Limit parameters and remains there for Time_Delay seconds. If the transition is to a value above

High_Limit or below Low_Limit, the Event Enrollment object generates a TO-OFFNORMAL transition. The event notification shall show an 'Event Type' of UNSIGNED_RANGE.

An UNSIGNED_RANGE clears when the referenced property attains a value from Low_Limit through High_Limit and remains there for Time_Delay seconds. The Event Enrollment object generates a TO-NORMAL transition. The event notification shall show an 'Event Type' of UNSIGNED_RANGE. See Figure 13-10.

[Add new Figure 13-10, p.245, and renumber existing Figure 13-10 and subsequent figures]

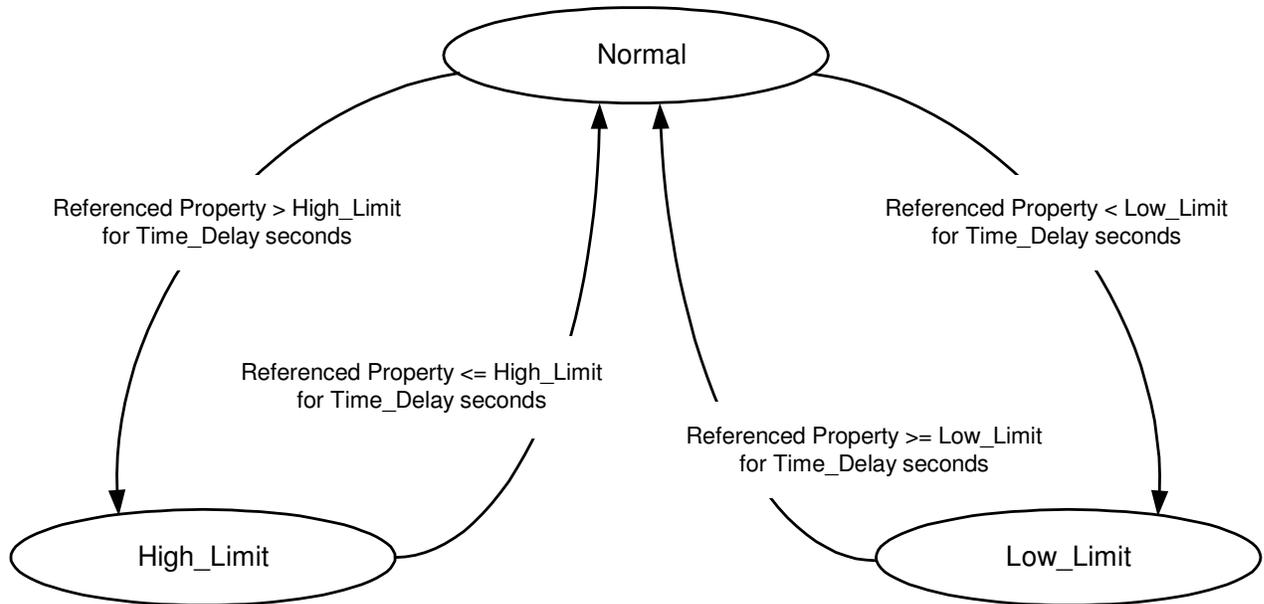


Figure 13-10. UNSIGNED_RANGE algorithm.

[Add new BACnetAccumulatorRecord production to **Clause 21**, p. 387]

```

BACnetAccumulatorRecord ::= SEQUENCE {
  timestamp          [0] BACnetDateTime,
  presentValue       [1] Unsigned,
  accumulatedValue   [2] Unsigned,
  accumulatorStatus [3] ENUMERATED {
    normal           (0),
    starting         (1),
    recovered        (2),
    abnormal         (3),
    failed           (4)
  }
}
  
```

[Change BACnetEventParameter production in **Clause 21**, p.393]

```

BACnetEventParameter ::= CHOICE {

-- These choices have a one-to-one correspondence with Event_Type enumeration
...
    change-of-life-safety [8] SEQUENCE {
        time-delay [0] Unsigned,
        list-of-life-safety-alarm-values [1] SEQUENCE OF BACnetLifeSafetyState,
        list-of-alarm-values [2] SEQUENCE OF BACnetLifeSafetyState,
        mode-property-reference [3] BACnetDeviceObjectPropertyReference
    † },

    unsigned-range [9] SEQUENCE {
        time-delay [0] Unsigned,
        low-limit [1] Unsigned,
        high-limit [2] Unsigned
    }
}

-- CHOICE [6] has been intentionally omitted. It parallels the complex-event-type CHOICE [6] of the
-- BACnetNotificationParameters production which was introduced to allow the addition of proprietary event
-- algorithms whose event parameters are not necessarily network-visible.

```

[Change BACnetEventType production in **Clause 21**, p.394]

```

BACnetEventType ::= ENUMERATED {
    change-of-bitstring (0),
    change-of-state (1),
    change-of-value (2),
    command-failure (3),
    floating-limit (4),
    out-of-range (5),
    -- complex-event-type (6), -- see comment below
    buffer-ready (7),
    change-of-life-safety (8),
    unsigned-range (9),
    ...
}

-- Enumerated values 0-63 are reserved for definition by ASHRAE. Enumerated values
-- 64-65535 may be used by others subject to the procedures and constraints described
-- in Clause 23. It is expected that these enumerated values will correspond to the use of the
-- complex-event-type CHOICE [6] of the BACnetNotificationParameters production.
-- The last enumeration used in this version is 8, 9.

```

[Change BACnetNotificationParameters production in **Clause 21**, p.396]

```

BACnetNotificationParameters ::= CHOICE {

-- These choices have a one-to-one correspondence with Event_Type enumeration
...
    change-of-life-safety [8] SEQUENCE {
        new-state [0] BACnetLifeSafetyState,
        new-mode [1] BACnetLifeSafetyMode,
        status-flags [2] BACnetStatusFlags,
        operation-expected [3] BACnetLifeSafetyOperation
    † },

```

```

unsigned-range      [9] SEQUENCE {
                        exceeding-value    [0] Unsigned,
                        status-flags      [1] BACnetStatusFlags,
                        exceeded-limit     [2] Unsigned
                    }
    }
    
```

[Add new BACnetPrescale production to **Clause 21**, p. 399]

```

BACnetPrescale ::= SEQUENCE {
    multiplier      [0] Unsigned,
    moduloDivide    [1] Unsigned
}
    
```

[Add new BACnetScale production to **Clause 21**, p. 405]

```

BACnetScale ::= CHOICE {
    floatScale      [0] REAL,
    integerScale    [1] INTEGER
}
    
```

[Add to **Annex C**, p. 428]

ANNEX C - FORMAL DESCRIPTION OF OBJECT TYPE STRUCTURES (INFORMATIVE)

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

```

ACCUMULATOR ::= SEQUENCE {
    object-identifier    [75] BACnetObjectIdentifier,
    object-name          [77] CharacterString,
    object-type          [79] BACnetObjectType,
    present-value        [85] Unsigned,
    description          [28] CharacterString OPTIONAL,
    device-type          [31] CharacterString OPTIONAL,
    status-flags         [111] BACnetStatusFlags,
    event-state          [36] BACnetEventState,
    reliability           [103] BACnetReliability OPTIONAL,
    out-of-service       [81] BOOLEAN,
    scale                [186] BACnetScale,
    units                [117] BACnetEngineeringUnits,
    prescale             [188] BACnetPrescale OPTIONAL,
    max-pres-value       [65] Unsigned,
    value-change-time    [192] BACnetDateTime OPTIONAL,
    value-before-change  [190] Unsigned OPTIONAL,
    value-set            [191] Unsigned OPTIONAL,
    logging-record       [184] BACnetAccumulatorRecord OPTIONAL,
    logging-device       [183] BACnetRecipient OPTIONAL,
    pulse-rate          [186] Unsigned OPTIONAL,
    high-limit           [45] Unsigned OPTIONAL,
    low-limit            [59] Unsigned OPTIONAL,
    limit-monitoring-interval [182] Unsigned OPTIONAL,
    event-type           [37] BACnetEventType,
    notification-class   [17] Unsigned OPTIONAL,
    time-delay           [113] Unsigned OPTIONAL,
    limit-enable         [52] BACnetLimitEnable OPTIONAL,
    event-enable         [35] BACnetEventTransitionBits OPTIONAL,
}
    
```


Attachment D: Pulse Converter Object

135c-5. Add a new Pulse Converter Object Type.

Rationale

There is need for a standard object to represent pulsed inputs indicating various measured values used for system monitoring and control.

Addendum 135c-5

[Add a new **Clause 12.22**, p. 224, and renumber the existing Clause 12.22 and subsequent clauses including tables and figures]

12.23 Pulse Converter Object Type

The Pulse Converter object type defines a standardized object that represents a process whereby ongoing measurements made of some quantity, such as electric power or water or natural gas usage, and represented by pulses or counts, might be monitored over some time interval for applications such as peak load management, where it is necessary to make periodic measurements but where a precise accounting of every input pulse or count is not required.

The Pulse Converter object might represent a physical input. As an alternative, it might acquire the data from the Present_Value of an Accumulator object, representing an input in the same device as the Pulse Converter object. This linkage is illustrated by the dotted line in Figure 12-3. Every time the Present_Value property of the Accumulator object is incremented, the Count property of the Pulse Converter object is also incremented.

The Present_Value property of the Pulse Converter object can be adjusted at any time by writing to the Adjust_Value property, which causes the Count property to be adjusted, and the Present_Value recomputed from Count. In the illustration in Figure 12-3, the Count property of the Pulse Converter was adjusted down to 0 when the Total_Count of the Accumulator object had the value 0070.

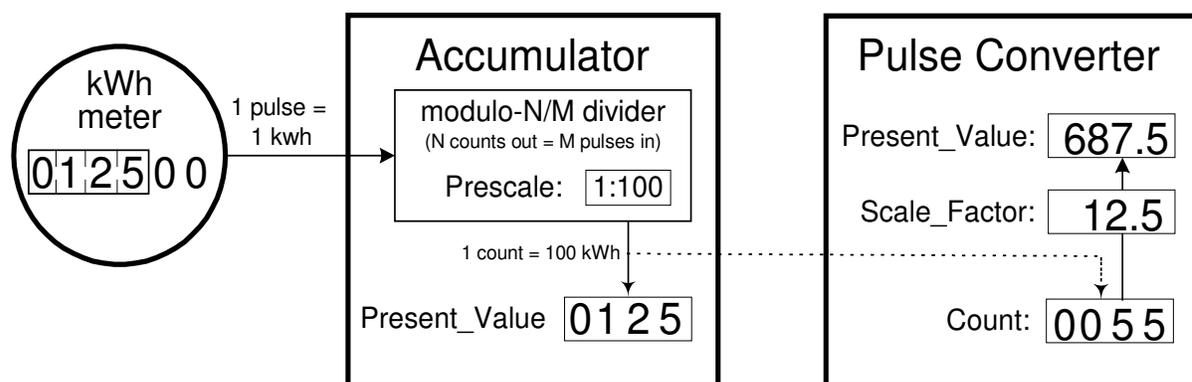


Figure 12-3. Relationship between the Pulse Converter and Accumulator objects.

The object and its properties are summarized in Table 12-27 and described in detail in this subclause.

Table 12-27. Properties of the Pulse Converter Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	REAL	R ¹
Input_Reference	BACnetObjectPropertyReference	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Out_Of_Service	BOOLEAN	R
Units	BACnetEngineeringUnits	R
Scale_Factor	REAL	R
Adjust_Value	REAL	W
Count	Unsigned	R
Update_Time	BACnetDateTime	R
Count_Change_Time	BACnetDateTime	R ²
Count_Before_Change	Unsigned	R ²
COV_Increment	REAL	O ³
COV_Period	Unsigned	O ³
Notification_Class	Unsigned	O ⁴
Time_Delay	Unsigned	O ⁴
High_Limit	REAL	O ⁴
Low_Limit	REAL	O ⁴
Deadband	REAL	O ⁴
Limit_Enable	BACnetLimitEnable	O ⁴
Event_Enable	BACnetEventTransitionBits	O ⁴
Acked_Transitions	BACnetEventTransitionBits	O ⁴
Notify_Type	BACnetNotifyType	O ⁴
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O ⁴
Profile_Name	CharacterString	O

¹ This property is required to be writable when Out_Of_Service is TRUE.

² These properties are required if Count_Before_Change is writable.

³ These properties are required if the object supports COV reporting.

⁴ These properties are required if the object supports intrinsic reporting.

12.23.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.23.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.23.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be PULSE_CONVERTER.

12.23.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.23.5 Present_Value

This property, of type REAL, indicates the accumulated value of the input being measured. It is computed by multiplying the current value of the Count property by the value of the Scale_Factor property. The value of the Present_Value property may be adjusted by writing to the Adjust_Value property. The Present_Value property shall be writable when Out_Of_Service is TRUE.

12.23.6 Input_Reference

This optional property, of type BACnetObjectPropertyReference, indicates the object and property (typically an Accumulator object's Present_Value property) representing the actual physical input that is to be measured and presented by the Pulse Converter object. The referenced property should have a datatype of INTEGER or Unsigned.

If this property is not present, the Pulse Converter object directly represents the physical input.

12.23.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a Pulse Converter. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM	Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
FAULT	Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
OVERRIDDEN	Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device. In this context "overridden" is taken to mean that the Present_Value, Count and Reliability properties are no longer tracking changes to the input. Otherwise, the value is logical FALSE (0).
OUT_OF_SERVICE	Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.23.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting and if the Reliability property is not present, then the value of this property shall be NORMAL. If the Reliability property

is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.23.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Present_Value and/or Count properties or the operation of the physical input in question is "reliable" as far as the BACnet Device or operator can determine and, if not, why. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, NO_SENSOR, OVER_RANGE, UNDER_RANGE, OPEN_LOOP, SHORTED_LOOP, UNRELIABLE_OTHER, CONFIGURATION_ERROR}

If Input_Reference is configured to reference a property that is not of datatype Unsigned or INTEGER, or is otherwise not supported as an input source for this object, the Reliability property shall indicate CONFIGURATION_ERROR.

12.23.10 Out_Of_Service

The Out_Of_Service property, of type BOOLEAN, is an indication whether (TRUE) or not (FALSE) the input that the object directly represents, if any, is not in service. ("Directly represents" means that the Input_Reference property is not present in this object.) The Present_Value property is decoupled from the Count property and will not track changes to the input when the value of Out_Of_Service is TRUE. In addition, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property shall be decoupled from the input when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties may be changed to any value as a means of simulating specific fixed conditions or for testing purposes. Other functions that depend on the state of the Present_Value or Reliability properties shall respond to changes made to these properties while Out_Of_Service is TRUE as if those changes had occurred in the input.

If the Input_Reference property is present, the state of the Out_Of_Service property of the object referenced by Input_Reference shall not be indicated by the Out_Of_Service property of the Pulse Converter object.

12.23.11 Units

This property, of type BACnetEngineeringUnits, indicates the measurement units of the Present_Value property. See the BACnetEngineeringUnits ASN.1 production in Clause 21 for a list of engineering units defined by this standard.

12.23.12 Scale_Factor

This property, of type REAL, provides the conversion factor for computing Present_Value. It represents the change in Present_Value resulting from changing the value of Count by one.

12.23.13 Adjust_Value

This property, of type REAL, is written to adjust the Present_Value property (and thus the Count property also) by the amount written to Adjust_Value.

If this property is writable the following series of operations shall be performed atomically when this property is written:

- (1) The value written to Adjust_Value shall be stored in the Adjust_Value property.
- (2) The value of Count shall be copied to the Count_Before_Change property.
- (3) The value of Count shall be decremented by the value calculated by performing the integer division (Adjust_Value/Scale_Factor) and discarding the remainder.

(4) The current date and time shall be stored in the Count_Change_Time property.

A write to this property results in a change in the value of Present_Value. Whether the new value is computed as part of the atomic series of operations or when Present_Value is read is a local matter.

If Adjust_Value has never been written, it shall have a value of zero.

12.23.14 Count

This read-only property, of type Unsigned, indicates the count of the input pulses as acquired from the physical input or the property referenced by the Input_Reference property.

If the property referenced by Input_Reference property is present, has datatype Unsigned or INTEGER, and is supported as an input source for this object, the value of the Count property is derived from the referenced property. An increment by one count in the referenced property is reflected by an increment of one count in the Count property. The means by which this is done shall be a local matter. Because the value of the Pulse Converter object Count property may be changed by a write to the Adjust_Value property, the value of the Count property can be different from the value of the referenced property.

12.23.15 Update_Time

This read-only property, of type BACnetDateTime, reflects the date and time of the most recent change to the Count property as a result of input pulse accumulation and is updated atomically with the Count property. If no such change has yet occurred, this property shall have wildcard values for all date and time fields.

12.23.16 Count_Change_Time

This read-only property, of type BACnetDateTime, represents the date and time of the most recent occurrence of a write to the Adjust_Value property. If no such write has yet occurred, this property shall have wildcard values for all date and time fields.

12.23.17 Count_Before_Change

This property, of type Unsigned, indicates the value of the Count property just prior to the most recent write to the Adjust_Value properties. If no such write has yet occurred, this property shall have the value zero.

12.23.18 COV_Increment

This property, of type REAL, shall specify the minimum change in Present_Value that will cause a COV notification to be issued to subscriber COV-clients. This property is required if COV reporting is supported by this object.

12.23.19 COV_Period

The COV_Period property, of type Unsigned, shall indicate the amount of time in seconds between the periodic COV notifications performed by this object. This property is required if COV reporting is supported by this object.

12.23.20 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.23.21 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value must remain outside the band defined by the High_Limit and Low_Limit properties before a TO-OFFNORMAL event is generated or remain within the same band, including the Deadband property, before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.22 High_Limit

This property, of type REAL, shall specify a limit that the Present_Value must exceed before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.22.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

- (a) the Present_Value must exceed the High_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the HighLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.23.22.2 Conditions for Generating a TO-NORMAL Event

Once exceeded, the Present_Value must fall below the High_Limit minus the Deadband before a TO-NORMAL event is generated under these conditions:

- (a) the Present_Value must fall below the High_Limit minus the Deadband for a minimum period of time, specified in the Time_Delay property, and
- (b) the HighLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-NORMAL flag must be set in the Event_Enable property.

12.23.23 Low_Limit

This property, of type REAL, shall specify a limit below which the Present_Value must fall before an event is generated. This property is required if intrinsic reporting is supported by this object.

12.23.23.1 Conditions for Generating a TO-OFFNORMAL Event

A TO-OFFNORMAL event is generated under these conditions:

- (a) the Present_Value must fall below the Low_Limit for a minimum period of time, specified in the Time_Delay property, and
- (b) the LowLimitEnable flag must be set in the Limit_Enable property, and
- (c) the TO-OFFNORMAL flag must be set in the Event_Enable property.

12.23.23.2 Conditions for Generating a TO-NORMAL Event

Once the Present_Value has fallen below the Low_Limit, the Present_Value must exceed the Low_Limit plus the Deadband before a TO-NORMAL event is generated under these conditions:

- (a) the Present_Value must exceed the Low_Limit plus the Deadband for a minimum period of time, specified in the Time_Delay property, and
- (b) the LowLimitEnable flag must be set in the Limit_Enable property, and

- (c) the TO-NORMAL flag must be set in the Event_Enable property.

12.23.24 Deadband

This property, of type REAL, shall specify a range between the High_Limit and Low_Limit properties, which the Present_Value must remain within for a TO-NORMAL event to be generated under these conditions:

- (a) the Present_Value must fall below the High_Limit minus Deadband, and
- (b) the Present_Value must exceed the Low_Limit plus the Deadband, and
- (c) the Present_Value must remain within this range for a minimum period of time, specified in the Time_Delay property, and
- (d) either the HighLimitEnable or LowLimitEnable flag must be set in the Limit_Enable property, and
- (e) the TO-NORMAL flag must be set in the Event_Enable property

This property is required if intrinsic reporting is supported by this object.

12.23.25 Limit_Enable

This property, of type BACnetLimitEnable, shall convey two flags that separately enable and disable reporting of HighLimit and LowLimit offnormal events and their return to normal. This property is required if intrinsic reporting is supported by this object.

12.23.26 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Pulse Converter objects, transitions to the High_Limit or Low_Limit Event_States are considered to be "offnormal" events. This property is required if intrinsic reporting is supported by this object.

12.23.27 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgements for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. In the context of Pulse Converter objects, transitions to High_Limit and Low_Limit Event_State are considered to be "offnormal" events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

- (a) upon receipt of the corresponding acknowledgement;
- (b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgement is expected);
- (c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgement is expected).

This property is required if intrinsic reporting is supported by this object.

12.23.28 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.23.29 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have X'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.23.30 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

[Change **Clause 13.1**, p. 234]

13.1 Change of Value Reporting

Change of value (COV) reporting allows a COV-client to subscribe with a COV-server, on a permanent or temporary basis, to receive reports of some changes of value of some referenced property based on fixed criteria. If an object provides COV reporting, then changes of value of any subscribed-to properties of the object, in some cases based on programmable increments, trigger COV notifications to be sent to subscribing clients. Typically, COV notifications are sent to supervisory programs in COV-client devices or to operators or logging devices. Any object, proprietary or standard, may support COV reporting at the implementor's option.

COV subscriptions are established using the SubscribeCOV service or the SubscribeCOVProperty service. The subscription establishes a connection between the change of value detection and reporting mechanism within the COV-server device and a "process" within the COV-client device. Notifications of changes are issued by the COV-server when changes occur after the subscription has been established. The ConfirmedCOVNotification and UnconfirmedCOVNotification services are used by the COV-server to convey change notifications. The choice of confirmed or unconfirmed service is specified in the subscription.

When a BACnet standard object, of a type listed in Table 13-1, supports COV reporting, it shall support COV reporting for the property as listed in Table 13-1. At the implementor's discretion, COV reporting may also be supported for any other property of the object. For properties listed in Table 13-1 that have a REAL data type, the COV increment used to determine when to generate notifications will be the COV_Increment property of the object unless a COV_Increment parameter is supplied in the SubscribeCOVProperty service. For other properties that have a REAL data type, the COV increment to use when not supplied with the SubscribeCOVProperty service shall be a local matter. This is to allow multiple subscribers that do not require a specific increment to use a common increment to allow for the reduction of the processing burden on the COV-server. The criteria for COV reporting for properties other than those listed in Table 13-1 is based on the data type of the property subscribed to and is described in Table 13-1a.

If an object supports the COV_Period property and COV_Period is non-zero, it shall issue COV notifications to all subscribed recipients at the regular interval specified by COV_Period, in addition to the notifications initiated by the change of value of the monitored property. The value of the monitored property conveyed by the periodic COV notification shall be the basis for determining whether a subsequent COV notification is required by the change in value of the monitored property. If COV_Period is zero, the periodic notifications shall not be issued.

...
 [Change **Table 13-1**, p. 235]

Table 13-1. Standardized Objects ~~Which~~ *That* May Support COV Reporting

Object Type	Criteria	Properties Reported
...
Loop	If Present_Value changes by COV_Increment or Status_Flags changes at all	Present_Value, Status_Flags, Setpoint, Controlled_Variable_Value
<i>Pulse Converter</i>	<i>If Present_Value changes by COV_Increment or Status_Flags changes at all or If COV_Period expires</i>	<i>Present_Value, Status_Flags, Update_Time</i>

[Change **Table 13-2**, p. 237]

Table 13-2. Standard Objects ~~that~~ *That* May Support Intrinsic Reporting

Object Type	Criteria	Event Type
...
Analog Input, Analog Output, Analog Value Value, <i>Pulse Converter</i> ,	If Present_Value exceeds range between High_Limit and Low_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable, OR Present_Value returns within the High_Limit – Deadband to Low_Limit + Deadband range for longer than Time_Delay AND the new transition is enabled in Event_Enable and Limit_Enable	OUT_OF_RANGE
...

[Change **Table 13-3**, p. 238]

Table 13-3. Standard Object Property Values Returned in Notifications

Object	Event Type	Notification Parameters	Referenced Object's Properties
...
Analog Input, Analog Output, Analog Value Value, <i>Pulse Converter</i>	OUT_OF_RANGE	Exceeding_Value Status_Flags Deadband Exceeded_Limit	Present_Value Status_Flags Deadband Low_Limit or High_Limit
...

[Change BACnetObjectType production in **Clause 21**, pp. 397-398]

```

BACnetObjectType ::= ENUMERATED {
    accumulator          (23),
    analog-input         (0),
    ...
    program              (16),
    pulse-converter      (24),
    schedule              (17),
    -- see averaging     (18),
    -- see multi-state-value (19),

```

```

trend-log          (20),
-- see life-safety-point (21),
-- see life-safety-zone (22),
-- see accumulator (23),
-- see pulse-converter (24),
...
}
-- Enumerated values 0-127 are reserved for definition by ASHRAE. Enumerated values
-- 128-1023 may be used by others subject to the procedures and constraints described
-- in Clause 23.

```

[Change BACnetObjectTypesSupported production in **Clause 21**, pp. 398-399]

```

BACnetObjectTypesSupported ::= BIT STRING {
  -- accumulator (23),
  analog-input (0),
  ...
  program (16),
  -- pulse-converter (24),
  schedule (17),
  -- trend-log (20),
-- Objects added after 1995
  averaging (18),
  multi-state-value (19),
  trend-log (20)
  life-safety-point (21),
  life-safety-zone (22),
-- Objects added after 2001
  accumulator (23),
  pulse-converter (24)
}

```

[Change BACnetPropertyIdentifier production in **Clause 21**, pp. 399-403]
[Note: schedule-default (174) appears in Addendum 135-2001a PR2]

```

BACnetPropertyIdentifier ::= ENUMERATED {
  accepted-modes (175),
  acked-transitions (0),
  ...
  active-cov-subscriptions (152),
  adjust-value (176),
  alarm-value (6),
  ...
  controlled-variable-value (21),
  count (177),
  count-before-change (178),
  count-change-time (179),
  cov-increment (22),
  cov-period (180),
  cov-resubscription-interval (128),
  ...
  in-process (47),
  input-reference (181),
  instance-of (48),
  ...
  limit-enable (52),

```

<i>limit-monitoring-interval</i>	(182),
list-of-group-members	(53),
...	
log-interval	(134),
<i>logging-device</i>	(183),
<i>logging-record</i>	(184),
low-limit	(59),
...	
polarity	(84),
<i>prescale</i>	(185),
present-value	(85),
...	
protocol-version	(98),
<i>pulse-rate</i>	(186),
read-only	(99),
...	
resolution	(106),
<i>scale</i>	(187),
<i>scale-factor</i>	(188),
schedule-default	(174),
...	
update-interval	(118),
<i>update-time</i>	(189),
utc-offset	(119),
...	
valid-samples	(146),
<i>value-before-change</i>	(190),
<i>value-set</i>	(191),
<i>value-change-time</i>	(192),
variance-value	(151),
...	
-- see schedule-default	(174),
-- see <i>accepted-modes</i>	(175),
-- see <i>adjust-value</i>	(176),
-- see <i>count</i>	(177),
-- see <i>count-before-change</i>	(178),
-- see <i>count-change-time</i>	(179),
-- see <i>cov-period</i>	(180),
-- see <i>input-reference</i>	(181),
-- see <i>limit-monitoring-interval</i>	(182),
-- see <i>logging-device</i>	(183),
-- see <i>logging-record</i>	(184),
-- see <i>prescale</i>	(185),
-- see <i>pulse-rate</i>	(186),
-- see <i>scale</i>	(187),
-- see <i>scale-factor</i>	(188),
-- see <i>update-time</i>	(189),
-- see <i>value-before-change</i>	(190),
-- see <i>value-set</i>	(191),
-- see <i>value-change-time</i>	(192),
...	
}	

-- The special property identifiers all, optional, and required are reserved for use in the ReadPropertyConditional and

-- ReadPropertyMultiple services or services not defined in this standard.

--
 -- Enumerated values 0-511 are reserved for definition by ASHRAE. Enumerated values 512-4194303 may be used by
 -- others subject to the procedures and constraints described in Clause 23. The highest enumeration used in this version is ~~474~~. 192.

[Add to **Clause 21**, BACnetReliability, pp. 404-405]

[Note: configuration-error (10) is defined in Addendum 135-2001a PR2]

```
BACnetReliability ::= ENUMERATED {
    ...
    multi-state fault      (9),
    configuration-error    (10),
    ...
}
```

[Add to **Annex C**, p. 437]

```
PROGRAM ::= SEQUENCE {
    ...
}
```

```
PULSE-CONVERTER ::= SEQUENCE {
    object-identifier      [75]   BACnetObjectIdentifier,
    object-name            [77]   CharacterString,
    object-type            [79]   BACnetObjectType,
    description            [28]   CharacterString OPTIONAL,
    present-value          [85]   REAL,
    input-reference        [181]  BACnetObjectPropertyReference OPTIONAL,
    status-flags           [111]  BACnetStatusFlags,
    event-state            [36]   BACnetEventState,
    reliability            [103]  BACnetReliability OPTIONAL,
    out-of-service        [81]   BOOLEAN,
    units                  [117]  BACnetEngineeringUnits,
    scale-factor           [188]  REAL,
    adjust-value           [176]  REAL,
    count                  [177]  Unsigned,
    update-time           [189]  BACnetDateTime,
    count-change-time     [179]  BACnetDateTime,
    count-before-change   [178]  Unsigned,
    cov-increment         [22]   REAL OPTIONAL,
    cov-period            [180]  Unsigned OPTIONAL,
    notification-class    [17]   Unsigned OPTIONAL,
    time-delay            [113]  Unsigned OPTIONAL,
    high-limit            [45]   REAL OPTIONAL,
    low-limit             [59]   REAL OPTIONAL,
    deadband              [25]   REAL OPTIONAL,
    limit-enable          [52]   BACnetLimitEnable OPTIONAL,
    event-enable          [35]   BACnetEventTransitionBits OPTIONAL,
    acked-transitions     [0]    BACnetEventTransitionBits OPTIONAL,
    notify-type           [72]   BACnetNotifyType OPTIONAL,
    event-time-stamps     [130]  SEQUENCE OF BACnetTimeStamp OPTIONAL,
    profile-name          [167]  CharacterString OPTIONAL
}
```

```
SCHEDULE ::= SEQUENCE { ...
```

[Add new **Annex D.22**, p. 455, and renumber existing Annex D.22 and subsequent clauses]

D.22 Example of a Pulse Converter object

Property: Object_Identifier = (Pulse Converter, Instance 1)
 Property: Object_Name = "Meter 5"
 Property: Object_Type = PULSE_CONVERTER
 Property: Description = ""
 Property: Present_Value = 125.0
 Property: Input_Reference = ((Accumulator, Instance 1), Present_Value)
 Property: Status_Flags = {FALSE, FALSE, FALSE, FALSE}
 Property: Event_State = NORMAL
 Property: Out_Of_Service = FALSE
 Property: Units = LITERS_PER_HOUR
 Property: Scale_Factor = 0.5
 Property: Adjust_Value = 500.0
 Property: Count = 250
 Property: Update_Time = (10-JUL-01,11:40:21.0),
 Property: Count_Change_Time = (10-JUL-01,11:30:01.0),
 Property: Count_Before_Change = 523
 Property: COV_Increment = 10.0
 Property: COV_Period = 3600
 Property: Notification_Class = 5
 Property: Time_Delay = 0
 Property: High_Limit = 1000.0
 Property: Low_Limit = 0.0
 Property: Deadband = 0.0
 Property: Limit_Enable = {FALSE, TRUE}
 Property: Event_Enable = {TRUE, FALSE, TRUE}
 Property: Acked_Transitions = {TRUE, TRUE, TRUE}
 Property: Notify_Type = ALARM
 Property: Event_Time_Stamps = ((12-JUL-01,18:50:21.2),
 (*-*.*,*:*:*.*),
 (12-JUL-01,19:01:34.0))

Attachment E: Load Control Object

Load Control Object Type

History:

Michael Kintner-Meyer and Martin Burns published a report (ASHRAE Research Project 1011-RP) on Utility/EMCS Integration in August 1999. They looked at requirements for applications between utilities and commercial customers. The report built on the Common Application Services Model (CASM) developed by the Electric Power Research Institute (EPRI).

SSPC135 is beginning to integrate some of the applications introduced in 1011-RP. Many of the applications make use of transactions similar to the existing read/write/notification operations available in BACnet. The Utilities Integration Working Group decided at the Buffalo Grove interim meeting in October 2001 to begin with the Load Control object, discussed in Sections 4.8 and 6.7 of 1011-RP. The Load Control object allows a BACnet client, such as a utility company, to request that a BACnet device shed a portion of its power load for a period of time. The mechanism of how this load is shed, including the selection of what internal loads will be shed to accomplish the overall load shed goal, is completely hidden from the client.

At the Honolulu meeting the basic format of the object was confirmed, although the emphasis of the discussion was on the need (as communicated by Marty Burns) to enable the utility to send a single uniform message to all customers that would not require implementation of the BACnet stack, but instead something like an email with the basic load shed parameters. The only feedback to the utility would be perhaps an acknowledgment receipt. Otherwise the utility simply monitors the power meter. As for specifics in the object: Stop_Time reverts back to Shed_Duration since this requires only one fixed time (Start_Time) and this can be both randomized and relative to “now”. The Full_Duty_Baseline property is probably not needed since the agreed level with utility might actually be a profile rather than some fixed level. Finally, the five properties (Level, Start Time, Duration, Randomized Start, Duty Window) included in the shed signal from the utility should be considered optional. We also discussed the need for an end-to-end solution; only solving the BACnet Object definition will not suffice. We need to specify a means of transport for getting the utility requests into the facility.

Based on discussions of several interested parties after the Honolulu meeting, additional changes were made.

- The Full_Duty_Baseline, Actual_Shed_Amount, and Actual_Start_Time properties were made optional, because it was felt that these properties are not important to the basic functionality of the object, although some manufacturers may want to support them to provide additional information about the load management. Shed_Mode was made required because its value is closely tied with the interpretation of Shed_Level.
- The Shed_Level (used to be Shed_Request), Start_Time, Shed_Duration, Random_Start, and Duty_Window properties were all made optional. The parameters of the shed request are now contained in the writable property Shed_Request. This is written to initiate load shedding. The optional properties are to provide easier visibility of the five load shed parameters, at the discretion of the implementer. (This has been superseded. Shed_Request was eliminated).
- The Shed_Request property now contains an updated list of the currently configured load shed parameters. After starting at the default values for each component, each write to Shed_Request will modify the values of the components that it contains, leaving the other components unchanged. Thus, Shed_Request contains the latest load shed information. (Shed_Request was eliminated).

Points for discussion in Columbus:

- Originally, all of the load shed parameters were separate. Because of problems with the atomicity of load shed requests, they were combined into one parameter. For the ease of viewing the properties, it was decided to leave the individual properties in place as optional, read-only properties. (again, superseded)
- Originally, any new load shed parameters that were received would not cause a change to the load shedding until the corresponding Start_Time was reached. This was because the Start_Time was intended to be written with

every load shed request. With all parameters being optional, it seems that any new load shed parameters would have to be immediately effective, because a corresponding Start_Time may not be issued in the request.

- The default value of Shed_Request is questionable. The value of each component is taken from the default value of the optional properties, but if the optional properties don't exist, we are taking the value that they would have if they existed. It has been suggested that this isn't consistent with the way things are done elsewhere in the specification.
- The discussions in Honolulu and afterward have indicated that the Load Control object is now to play a dual role. It will operate in a "within-building" mode, where a BACnet device controlling a potentially sheddable load may have a Load Control object, which can be commanded to shed power by some local facility master device. Alternately, the Load Control object is to serve as the interface to the utility company, although this transport mechanism is not yet defined, and is likely not to involve the currently defined communications stack. This sounds suspiciously like the dual roles that the Counter object was once attempting to fill, which lead to quite a lot of headaches before the functionality was broken into two distinct objects. It seems that the best approach would be to restrict the scope of the problem that we are trying to solve with one object, so that the problem can be solved cleanly and efficiently. If another object is needed to solve a different problem, then so be it.

Guidance received from SSPC 135 in Columbus (from UIWG minutes dated 9/26/02 by J. Martocci):

There were two major topics that needed clarification: 1) how to guarantee atomicity when setting the 5 controlling parameters for the object and 2) how to transport the requested control information from the Utility to the various Facilities. The discussion provided direction for the former, however, we did not tackle the latter.

With regard to Atomicity in setting the 5 operative parameters of the Load Control object:

1. This topic has been debated before in similar cases and never implemented.
2. The committee agrees that the Write Multiple service is not atomic. However, the committee is concerned that defining the service for this case, and not reviewing all other cases already established in the standard would be problematic.

The object as currently specified has the 5 operational parameters defined as individual optional read-only properties. A sixth property (Shed_Request) is the amalgamation of the 5 parameters to be used for atomicity purposes. We discussed all the pros/cons of the approaches. Other than creating a new or updated service, all options had good and bad points. A straw poll was taken as to how the object should get defined with the following results:

- 4 – voted to keep the 5 parameters (but make them required) and keep the Shed_Request property as drafted.
- 2 – voted to keep the Shed-Request property, but eliminate the 5 properties.
- 6 – voted to keep the 5 properties, and drop the Shed_Request property

The suggested actions given by the committee to the working group is as follows...

1. The document should be redrafted eliminating the Shed_Request property.
2. The UI Working group should review the 5 parameters and perform a sensitivity analysis on the inputs to determine the possible negative effects that could occur by setting them at slightly different times.
3. The narrative of the object should note the dependency of these parameters and give guidance that the implementer should use a Write Multiple service request to perform the operations giving near atomic support. *NOTE: It was further suggested that a discussion of this nature be added in a more prominent section of the standard (e.g. Clause 12) so that it would apply as a general rule rather than unique for this application.*
4. Review the COV requirement needs for the 5 parameters. If needed we may want to group these parameters in a COV to reduce network overhead.
5. Dave Robin, Carl Neilson and Bill Swan had previously put together an 'object locking' document that would insure atomicity. Dave, Carl and Bill will dust off this document and review it for a possible long-term implementation strategy to handle atomicity.

Grant Wichenko noted that the Load and Real-time pricing parameters (e.g. KW Ratings, on-time, off-time) were not included in the object. Jerry mentioned that the direction given in the Redmond meeting was that the actual shedding algorithms not be prescribed in the standard and be a local matter. We also see the Real-time pricing information to be another object that is referenced to determine how the intra-facility shedding (and how much) will

occur. This opened a new discussion about the need for an end-to-end design that captures actions from the Utility down to the loads. The committee took this as an action item to be developed prior to the next committee review.

Dave Robin mentioned that he would have expected a dialog to occur between the LCO objects within the facility. This was not in the existing description since the Working Group had been focusing on the Utility to Facility interface. Dave is correct though that once within the facility, the LCOs will need to communicate to determine how the shedding across the facility is proceeding.

Per guidance above:

1. Shed_Request property taken out.
2. Sensitivity analysis:

Presently we have Shed_Level not being reset after execution completed. One could subsequently write to Start_Time and have the same shed again. If reset, then Shed_Level must also be written to each time. If we assume a stateless controller (client), then said client will be forced to always write Shed_Level whether the value of Shed_Level has changed or not. If it is willing to keep a record of what it did last time, then it might not need to write Shed_Level, but what is gained really? Suppose we always use Shed_Level = 0 for normal, =1 for normal shed and =2 for major shed, then I would like default to be 0. But then I am still forced to write to Shed_Level to initiate the shed. I could leave it at =1 and that would mean most of the time I wouldn't have to write to it to shed, but I still need to be stateful, and sometimes need to write to it. Is bandwidth that tight? On the other hand, resetting to defaults would insure that if a write is received on some other property, but somehow the write to Shed_Level is dropped that the result is nothing happens, rather than some unintended shed. (this would only be important if someone *didn't* use WPM, see next paragraph).

Shed_Level
Start_Time
Shed_Duration
Random_Start
Duty_Window

We are supposed to receive these 5 properties (or subset) in a WPM. If that is the case, how could all of them not be received at the same time (other than the milliseconds between writes required by the device to process the WPM)? It seems that in general the first three properties are likely to be written to regularly, while the last two stick to preset values. So now a WPM arrives writing the first three. They have to arrive in order, yes? They have to arrive in the same packet, yes? If so, we have only a very short delay, and no possibility that, for example, Start_Time has been written while Shed_Level was dropped. If only Shed_Duration is written, then one can be positive that Shed_Level and Start_Time were not intended to be written.

So, let's say there is a slow device, and Shed_Level gets written, but it takes a while to write Start_Time and Shed_Duration. When does the LCO know to start doing something? Maybe the problem is not the LCO—the device receives a message that is then addressed to the properties of the LCO, but the LCO doesn't see the packet headers. It most likely will only see the individual writes to the properties. So then, how does it know when the writes are done?

One possibility is to require writing Shed_Level each time, and the using of WPM mandatory, and also to specify the WPM order such that Shed_Level is written last. In this case, the LCO knows that it can act once Shed_Level has been written. WPM guarantees that all properties (or none) have been written. Seeing Shed_Level written insures that the shed command has been received completely. This is how the object is written now.

There is no sensitivity analysis beyond this, if I understand things correctly. Changes to the document related to the required use of the WPM service are highlighted in yellow.

3. Narrative-

The narrative has been updated to indicate that the shed control is accomplished solely through the use of the WritePropertyMultiple service request. Changes related to this request are highlighted in green.

4. COV

The LCO is currently set up for COV reporting whenever the Present_Value, or any of the five load shed parameters changes. This allows a facility manager to be informed whenever a load shed request comes in and is sent to the devices in his building. Changes related to this request are highlighted in green.

Guidance received from SSPC 135 in Chicago 1/25-7/2003:

The issue of the atomicity of the shed command was debated. The end result is that the state machine for the shed mechanism should be diagrammed, and leave as an implementation matter exactly how it is carried out, whether by enforcing a WPM, or not applying any command for a “waiting” period, or whatever.

The SHED_FAULT state is to be removed. The SHED_ALARM state is to be renamed to SHED_TARGET_NOT_MET. A list of states that are also considered alarm states is provided at the implementor’s discretion.

There was also a suggestion to possibly make the value commandable. The committee did not have a formal recommendation on this topic. As there is expected to only be one issuer of load control requests, this has not been included in the object.

There was a requirement to clean up language implying that there is a single load control object per device. This is not necessarily the case, and the proposal language should not imply this.

There was a requirement to combine the Shed_Mode and Shed_Amount properties into a single property, which would be a CHOICE data type, where the choice would implicitly describe the shed mode.

The state machine has now been described with a diagram, Figure 12-1. I have also updated the language in the beginning of the object to describe the important transitions in the diagram.

Guidance received from UIWG in Germantown, MD 4/28/2003

Add property State_Description. Make BACnetShedState a non-extensible enumeration. Additional information, such as that which would be provided by additional states, is to be conveyed through the State_Description property.

Change UnsuccessfulShedReconfigured transition to ShedConditionsChanged, and remove CanNowMeetShed transition. Remove ActiveShedReqReceived transition.

Compare Start_Time and current time on device restart to determine if shedding must begin. Also use language similar to the Schedule object to suggest that load shed properties should be persisted across device restart.

Remove footnote 1 reference to Out_Of_Service property; add Enable property (BOOLEAN default=TRUE) – if false, LCO ignores all requests and remains in the Shed_Inactive state, transitioning to it if necessary.

Remove 12.X.8.2.

Change ShedRequestReceived semantics. Should indicate only reception of shed request, not to indicate anything about the current shedding status.

Changes made from CG-001-5 to CG-001-6

- Added State_Description property.
- Made BACnetShed_State non-extensible.
- Added language about restorative behavior across restart/time change.
- Removed footnote 1.
- Removed Alarm_Values property.
- Added Enabled property.
- Reworked state diagram.

- Reworked state diagram description.
- Removed 12.X.8.2.
- Renumbered sections.
- Cleaned up ASN.1.

Issues to consider:

- I included the CanNowMeetShed transition, despite comments in Germantown, to handle the case where the load shed conditions change independent of any reconfiguration of the desired shed – what if a previously critical lighting panel has become noncritical, and can now shed its load?
- State diagram is missing some transitions from previous one – this is because of the new semantics where all writes cause a transition to SHED_REQUEST_RECEIVED for further evaluation.

The subject matter of this proposal (has has not) been verified by implementation.

Changes to ASHRAE Standard 135: (reference: BACnet2001publication draft.doc)

[Add new **12.X**, “Load Control Object Type, ca. p.196]

12.X Load Control Object Type

The Load Control object type defines a standardized object whose properties represent the externally visible characteristics of a mechanism for controlling load requirements in a BACnet device. A BACnet device can use a Load Control Object to allow external control over the shedding of a load that it controls. The mechanisms by which the loads are shed are not visible to the BACnet client. One or more objects may be used in the device to allow independent control over different sub-loads. The Load Control Object may also be used in a hierarchical fashion to control other Load Control Objects in other BACnet devices.

A BACnet client, such as a master controller or a utility company, can request that the Load Control Object shed a portion of its load for a specified time by writing to any or all of the five properties: Shed_Level, Start_Time, Shed_Duration, Random_Start, and Duty_Window. For any given shed request, each of these parameters is optional. Modification of these shed request parameters serves to configure the load shed command. Initial values of these properties, and values taken at the completion of a shed command execution, are as specified in the individual property descriptions.

The Load Control Object shed mechanism follows a state machine whose operation is displayed in Figure 12-1. This state machine captures the transitions that occur within the Load Control Object. When the Load Control Object is not currently shedding, the Present_Value property has the value SHED_INACTIVE. Receipt of a write to the properties Shed_Level, Shed_Duration, Random_Start, Duty_Window, or Start_Time shall cause the object to follow the ReceiveShedRequest transition to the SHED_REQUEST_RECEIVED state. In this state the Load Control Object will compare the Start_Time with the current time. If Start_Time is in the past, the object shall follow the BeginShed transition to the SHEDDING state. If Start_Time is in the future, the object shall remain in the SHED_REQUEST_RECEIVED state. At Start_Time plus a randomized offset defined by Random_Start, the object shall then follow the BeginShed transition to the SHEDDING state. Once the requested shed amount is achieved, the object shall follow the AbleToMeetShed transition to the SHED_TARGET_MET state. If the full requested shed amount cannot be shed, the object shall follow the CannotMeetShed transition to the SHED_TARGET_NOT_MET state. If the ability to meet the requested shed amount changes during the shedding, the CanNowMeetShed and CanNoLongerMeetShed transitions shall be followed between the SHED_TARGET_NOT_MET and SHED_TARGET_MET states. Once the Shed_Duration has expired, the object shall follow the FinishedSuccessfulShed or FinishedUnsuccessfulShed transition to the SHED_INACTIVE state.

Receipt of a write to Shed_Level, Shed_Duration, Random_Start, Duty_Window, or Start_Time while shedding is currently pending or active shall cause the object to transition to the SHED_REQUEST_RECEIVED state. This transition does not mandate anything about the state of the actual shedding, only that the new request must be processed to determine if shedding should continue or terminate. After processing the request, the object shall then transition to the SHED_INACTIVE or SHEDDING state as appropriate. From the SHEDDING state, the object shall transition to the SHED_TARGET_MET or SHED_TARGET_NOT_MET state as appropriate. This is a statement of the logical behavior of the state machine. In some devices the transitions may be rapid enough so as to be invisible to an external device.

This state machine only describes the behavior of the Load Control Object when the Enabled property has the value TRUE. See 12.X.15 for a description of the effect of this property. If the device is unable to comply fully with the shed request by shedding the entire amount of power load requested, it is a local matter whether the device sheds as much load as it can, or whether it does not shed any of its loads.

The Load Control Object shall exhibit restorative behavior across restart or time change of the BACnet device in which it resides. Upon restart of the device or change of the device time, the Load Control Object shall compare the current time with the Start_Time property to determine if load shedding should commence. If load shedding is to begin, the object shall transition to the SHEDDING state and begin shedding its loads.

While the Load Control object is designed to work independently, it is possible that there will exist within the building (or even within a device) a hierarchy of Load Control objects, where one particular Load Control object receives a load shed command from a higher level (say the utility), and the controller which hosts that object in turn will be responsible for managing and issuing requests to other Load Control objects. The management of this hierarchy, such as how to handle shed management of the low level Load Control objects, and how to handle shed failures at the low level objects, is a local matter. A dialog using the Load Control objects could be accomplished using alarm states to communicate ability to comply with a shed request, while the Actual_Shed_Amount property could report the amount that a sub-device (or sub-controller) is able to shed. Alternatively, this dialog could be performed separately from the Load Control objects.

Figure 12-1. State Diagram for Load Control Object.

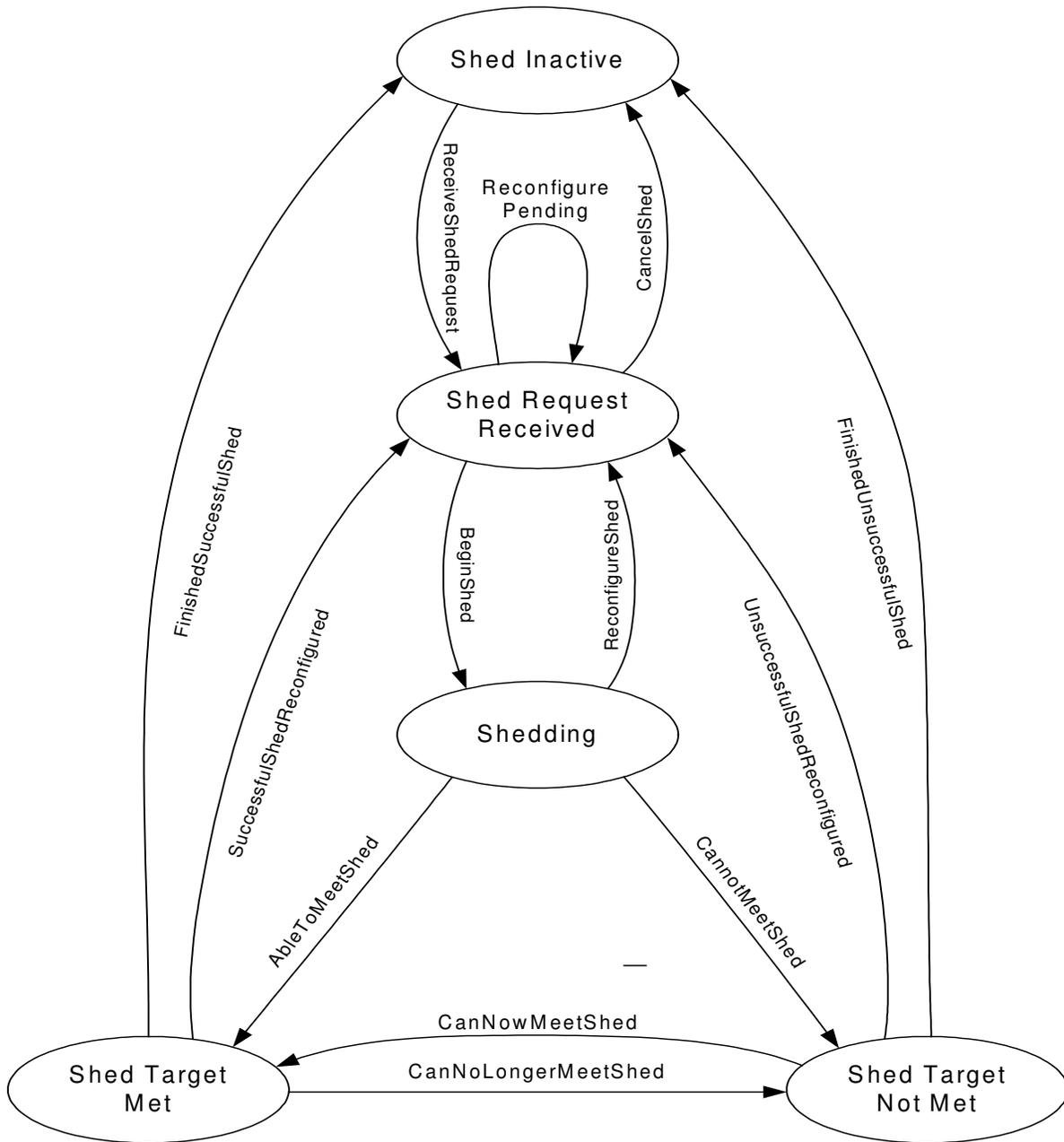


Table 12-X. Properties of the Load Control Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	BACnetShedState	R
State_Description	CharacterString	O
Status_Flags	BACnetStatusFlags	R
Event_State	BACnetEventState	R
Reliability	BACnetReliability	O
Shed_Level	BACnetShedLevel	W
Start_Time	BACnetDateTime	W
Shed_Duration	Unsigned	W
Random_Start	Unsigned	W
Duty_Window	Unsigned	W
Enabled	BOOLEAN	O
Full_Duty_Baseline	REAL	O
Actual_Start_Time	BACnetDateTime	O ¹
Actual_Shed_Amount	REAL	O ¹
Notification_Class	Unsigned	O ²
Time_Delay	Unsigned	O ²
Event_Enable	BACnetEventTransitionBits	O ²
Acked_Transitions	BACnetEventTransitionBits	O ²
Notify_Type	BACnetNotifyType	O ²
Event_Time_Stamps	BACnetARRAY[3] of BACnetTimeStamp	O ²
Profile_Name	CharacterString	O

¹These properties, if present, must be readonly.

²These properties are required if the object supports intrinsic reporting.

12.X.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.X.2 Object_Name

This property, of type CharacterString, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the Object_Name shall be restricted to printable characters.

12.X.3 Object_Type

This property, of type BACnetObjectType, indicates membership in a particular object type class. The value of this property shall be LOAD_CONTROL.

12.X.4 Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted.

12.X.5 Present_Value

This property, of type BACnetShedState, indicates the current load shedding state of the object. See Figure 12-1 for a diagram of the state machine governing the value of Present_Value.

12.X.6 State_Description

This property, of type CharacterString, is a string of printable characters whose content is not restricted. The State_Description provides additional information about the shed state of the Load Control Object.

12.X.7 Status_Flags

This property, of type BACnetStatusFlags, represents four Boolean flags that indicate the general "health" of a Load Control object. Three of the flags are associated with the values of other properties of this object. A more detailed status could be determined by reading the properties that are linked to these flags. The relationship between individual flags is not defined by the protocol. The four flags are

{IN_ALARM, FAULT, OVERRIDDEN, OUT_OF_SERVICE}

where:

IN_ALARM Logical FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).

FAULT Logical TRUE (1) if the Reliability property is present and does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).

OVERRIDDEN Logical TRUE (1) if the point has been overridden by some mechanism local to the BACnet Device, otherwise logical FALSE (0).

OUT_OF_SERVICE Logical TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

12.X.8 Event_State

The Event_State property, of type BACnetEventState, is included in order to provide a way to determine if this object has an active event state associated with it. If the object supports intrinsic reporting, then the Event_State property shall indicate the event state of the object. If the object does not support intrinsic reporting and if the Reliability property is not present then the value of this property shall be NORMAL. If the Reliability property is present and does not have a value of NO_FAULT_DETECTED, then the value of the Event_State property shall be FAULT. Changes in the Event_State property to the value FAULT are considered to be "fault" events.

12.X.9 Reliability

The Reliability property, of type BACnetReliability, provides an indication of whether the Load Control object is reliably reporting its compliance with any load shed requests. The Reliability property for this object type may have any of the following values:

{NO_FAULT_DETECTED, UNRELIABLE_OTHER}

12.X.10 Shed_Level

This property, of type BACnetShedLevel, indicates the desired load shedding. If the choice for Shed_Level is PERCENT, the value of Shed_Level is interpreted as a requested percentage of the baseline power usage to which the device is to attempt to reduce its load. The determination of the baseline power usage is a local

matter. It may be determined from the Full_Duty_Baseline property, if present. Or it may be defined by an hourly profile, agreed upon by the facility manager and the utility company in advance. If the choice for Shed_Level is LEVEL, the value of Shed_Level is used to set a preconfigured level of load shedding. The meaning of these shed levels is a local matter. If Shed_Mode is AMOUNT, the value of Shed_Level is to be interpreted as an amount, in kilowatts, by which to reduce power usage. Note that these amounts in all three cases are always absolute, not differential, so the previous value of Shed_Level is not used in determining the new shed level. The following table describes the default values and power targets for the different choices of Shed_Level:

Table 12-X. Shed_Level default values and power targets

Choice	Default value of Shed_Level	Power load target in kW
PERCENT]	100	(current baseline) * Shed_Level / 100
LEVEL	0	locally prespecified shed target for the given level
AMOUNT	0.0	(current power usage) - Shed_Level

If a load control command has been issued, and execution of the command has completed, Shed_Level shall be reset to the default value appropriate to the value of Shed_Level used for the last command.

12.X.11 Start_Time

This property, of type BACnetDateTime, indicates the time at which execution of the load control shall begin. Shedding begins at the Actual_Start_Time, which is later than the issued Start_Time by a random amount of time less than or equal to Random_Start. If no load control command has been written to this property, Start_Time shall contain all wildcard values. If a load control command has been issued, and execution of the command has completed, Start_Time shall be reset by the device to contain all wildcard values. If Start_Time contains wildcard values for the Date portion of the BACnetDateTime *only* (wildcards are acceptable for the seconds and hundredths fields of the Time portion), the device shall execute the load shed request at the specified time on every day and Start_Time shall *not* be reset by the device to all wildcard values. A load control command may be issued without Start_Time being set to indicate “start now”. In this case, the previous load control command is terminated and Start_Time shall be reset by the device to the current time.

12.X.12 Shed_Duration

This property, of type Unsigned, indicates the duration of the load shed action, starting at Start_Time plus a randomized offset. The units for Shed_Duration are minutes. If no load control command has been written to this property, Shed_Duration shall contain wildcard values. If a load control command has been issued, and execution of the command has completed, Shed_Duration shall be reset by the device to contain wildcard values. A load control command may be issued without Shed_Duration being set to indicate an indefinite length load shed request.

12.X.13 Random_Start

This property, of type Unsigned, indicates the maximum value of a random time offset to be applied to the start of execution of the load shed command. If this property is zero, the load shed command will execute at Start_Time, or as soon thereafter as the device is able to begin load shedding. If this property is greater than zero, the device will begin load shedding at a point in time offset from Start_Time by some value between zero and Random_Start minutes. Note that this offset does not affect Shed_Duration. If no load control command has been written to this property, Random_Start shall be set to some pre-agreed on value. If a load control command has been issued, and execution of the command has completed, Random_Start shall be reset by the device to this pre-agreed value. The units of Random_Start are minutes.

12.X.14 Duty_Window

This property, of type Unsigned, indicates the duty window over which the load shedding is to be realized. The units for Duty_Window are minutes. Duty_Window is used for performance measurement or compliance purposes. The average power consumption across the Duty_Window must be less than or equal to the requested

reduced consumption. Whether this window is fixed or sliding is a local matter. Duty_Window begins at the Actual_Start_Time. If no load control command has been written to this property, Duty_Window shall be set to some pre-agreed on value. If a load control command has been issued, and execution of the command has completed, Duty_Window shall be reset by the device to this pre-agreed value.

12.X.15 Enabled

This property, of type BOOLEAN, indicates whether the Load Control Object is currently enabled to respond to load shed requests. If Enabled is TRUE, the object will respond to load shed requests normally, and follow the state machine described in Figure 12-1. If Enabled is FALSE, the object will transition to the Shed_Inactive state if necessary and remain in that state. It will not respond to any load shed request while Enabled is FALSE.

12.X.16 Full_Duty_Baseline

This property, of type REAL, indicates the baseline power consumption value for this device, if a fixed baseline is used. Shed requests are with respect to this baseline, i.e., “percent of baseline” and “amount off baseline”. This may optionally be writable by an external device through the WriteProperty or WritePropertyMultiple service requests. The frequency with which this property is written (if writable) is a local matter. The units of Full_Duty_Baseline are kilowatts.

12.X.17 Actual_Start_Time

This property, of type BACnetDateTime, indicates the time at which execution of the load control actually begins after adding a random offset between zero and Random_Start minutes to Start_Time. If no load control command has been issued to this object, Actual_Start_Time shall contain wildcard values. If a load control command has been issued, and execution of the command has completed, Actual_Start_Time shall be reset by the device to contain wildcard values.

12.X.18 Actual_Shed_Amount

This property, of type REAL, indicates the actual amount of power that is currently being shed in response to a load shed request. This value is an average calculated over Duty_Window minutes. This may be a fixed window, or a sliding window, at the discretion of the implementer. This can be useful in determining the power consumption reduction in the event that the device is able to shed some, but not all, of the requested amount. The units for Actual_Shed_Amount are the same as the units for Shed_Level, i.e., dependent on the value of Shed_Mode.

12.X.19 Notification_Class

This property, of type Unsigned, shall specify the notification class to be used when handling and generating event notifications for this object. The Notification_Class property implicitly refers to a Notification Class object that has a Notification_Class property with the same value. This property is required if intrinsic reporting is supported by this object.

12.X.20 Time_Delay

This property, of type Unsigned, shall specify the minimum period of time in seconds that the Present_Value property may remain equal to SHED_TARGET_NOT_MET before a TO-OFFNORMAL event is generated, or not equal to SHED_TARGET_NOT_MET before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

12.X.21 Event_Enable

This property, of type BACnetEventTransitionBits, shall convey three flags that separately enable and disable reporting of TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

12.X.22 Acked_Transitions

This property, of type BACnetEventTransitionBits, shall convey three flags that separately indicate the receipt of acknowledgements for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. These flags shall be cleared upon the occurrence of the corresponding event and set under any of these conditions:

- (a) upon receipt of the corresponding acknowledgement;
- (b) upon the occurrence of the event if the corresponding flag is not set in the Event_Enable property (meaning event notifications will not be generated for this condition and thus no acknowledgement is expected);
- (c) upon the occurrence of the event if the corresponding flag is set in the Event_Enable property and the corresponding flag in the Ack_Required property of the Notification Class object implicitly referenced by the Notification_Class property of this object is not set (meaning no acknowledgement is expected).

This property is required if intrinsic reporting is supported by this object.

12.X.23 Notify_Type

This property, of type BACnetNotifyType, shall convey whether the notifications generated by the object should be Events or Alarms. This property is required if intrinsic reporting is supported by this object.

12.X.24 Event_Time_Stamps

This optional property, of type BACnetARRAY[3] of BACnetTimeStamp, shall convey the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. Time stamps of type Time or Date shall have X'FF' in each octet and Sequence number time stamps shall have the value 0 if no event notification of that type has been generated since the object was created. This property is required if intrinsic reporting is supported by this object.

12.X.25 Profile_Name

This property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

[Renumber **Figure 12-1**, “Loop object structure with its referenced objects”, p. 198]

Figure 12-2. Loop object structure with its referenced objects.

[Change text in **12.16** to reflect renumbering of Figures, p. 196]

... Figure 12-1 illustrates the relationship between the Loop object properties and the other objects referenced by the loop.

[Renumber **Figure 12-2**, “State Transitions for the program object”, p. 221]

Figure 12-3. State Transitions for the program object.

[Change text in 12.21.5 to reflect renumbering of Figures, p. 220]

... Figure 12-3 shows the valid state transitions and the resulting new Program_State.

[Changes to tables in 13.1, “Change of Value Reporting”, p. 235]

Table 13-1. Standardized Objects Which May Support COV Reporting

Object Type	Criteria	Properties Reported
...		
Loop	If Present_Value changes by COV_Increment or Status_Flags changes at all	Present_Value, Status_Flags, Setpoint, Controlled_Variable_Value
Load Control	<i>If Present_Value, Shed_Level, Start_Time, Shed_Duration, Random_Start, or Duty_Window changes at all</i>	<i>Present_Value, Status_Flags, Shed_Level, Start_Time, Shed_Duration, Random_Start, Duty_Window</i>

[Changes to tables in 13.2, “Intrinsic Reporting”, p. 237]

Table 13-2. Standard Objects that May Support Intrinsic Reporting

Object Type	Criteria	Event Type
...		
Binary Output, Multi-state Output	If Present_Value differs from Feedback_Value for longer than Time_Delay AND the new transition is enabled in Event_Enable	COMMAND_FAILURE
Load Control	<i>If Present_Value equals SHED_TARGET_NOT_MET or any value in Alarm_Values for longer than Time_Delay AND the new transition is enabled in Event_Enable</i>	<i>COMMAND_FAILURE</i>
Loop	If the absolute difference between Setpoint and Controlled_Variable_Value exceeds Error_Limit for longer than Time_Delay AND the new transition is enabled in Event_Enable	FLOATING_LIMIT
...		

[Changes to tables in 13.2, “Intrinsic Reporting”, p. 238]

Table 13-3. Standard Object Property Values Returned in Notifications

Object	Event Type	Notification Parameters	Referenced Object's Properties
...			
Binary Output, Multi-state Output	COMMAND_FAILURE	Command_Value Status_Flags Feedback_Value	Present_Value Status_Flags Feedback_Value
<i>Load Control</i>	<i>COMMAND_FAILURE</i>	<i>Command_Value Status_Flags Feedback_Value</i>	<i>Shed_Level Status_Flags Actual_Shed_Amount</i>
Loop	FLOATING_LIMIT	Referenced_Value Status_Flags Setpoint_Value	Controlled_Variable_Value Status_Flags Setpoint

		Error_Limit	Error_Limit
...			

[Add to **21**, BACnetObjectType, p. 398]

load-control (n),

[Add to **21**, BACnetObjectTypesSupported, p. 399]

load-control (n),

[Add to **21**, BACnetPropertyIdentifier (distributed alphabetically), pp. 400-404]

actual-shed-amount (n),
actual-start-time (n),
duty-window (n),
enabled (n),
full-duty-baseline (n),
random-start (n),
shed-duration (n),
shed-level (n),
state-description (n),

[Add comments to same production]

-see *actual-shed-amount* (n),
-see *actual-start-time* (n),
-see *duty-window* (n),
-see *enabled* (n),
-see *full-duty-baseline* (n),
-see *random-start* (n),
-see *shed-duration* (n),
-see *shed-level* (n),
-see *state-description* (n)

[add to **21**, new production, p. 406]

```
BACnetShedLevel ::= CHOICE {
    percent [0] Unsigned,
    level [1] Unsigned,
    amount [2] REAL
}
```

[add to **21**, new production, p. 406]

```
BACnetShedState ::= ENUMERATED {
    shed-inactive (0),
    shed-request-received (1),
    shedding (2),
    shed-target-met (3),
    shed-target-not-met (4),
}
```

[Add to **Annex C**, p. 435]

```

LOAD-CONTROL ::= SEQUENCE {
    object-identifier    [75] BACnetObjectIdentifier,
    object-name         [77] CharacterString,
    object-type         [79] BACnetObjectType,
    description         [28]CharacterString OPTIONAL,
    present-value       [85] BACnetShedState,
    state-description   [28]CharacterString OPTIONAL,
    status-flags       [111] BACnetStatusFlags,
    event-state         [36] BACnetEventState,
    reliability         [103] BACnetReliability OPTIONAL,
    shed-level          [n] BACnetShedLevel,
    start-time          [n] BACnetDateTime,
    shed-duration       [n] Unsigned,
    random-start        [n] Unsigned,
    duty-window         [n] Unsigned,
    enabled             [n] BOOLEAN,
    full-duty-baseline  [n] REAL OPTIONAL,
    actual-start-time   [n] BACnetDateTime OPTIONAL,
    actual-shed-amount  [n] REAL OPTIONAL,
    notification-class  [17] Unsigned OPTIONAL,
    time-delay          [113] Unsigned OPTIONAL,
    event-enable        [35] BACnetEventTransitionBits OPTIONAL,
    acked-transitions  [0] BACnetEventTransitionBits OPTIONAL,
    notify-type         [72] BACnetNotifyType OPTIONAL,
    event-time-stamps  [130] SEQUENCE OF BACnetTimeStamp OPTIONAL
    profile-name        [167] CharacterString OPTIONAL
}
    
```

[Add to **Annex D**, p. 450]

D.X Example of a Load Control object

```

Property: Object_Identifier = (Load_Control, Instance 1)
Property: Object_Name = "Load Control 1"
Property: Object_Type = LOAD CONTROL
Property: Description = "Chiller Load Control"
Property: Present_Value = SHED_TARGET_MET
Property: State_Description = "shed accomplished"
Property: Status_Flags = {FALSE, FALSE, FALSE, FALSE}
Property: Event_State = NORMAL
Property: Reliability = NO_FAULT_DETECTED
Property: Shed_Level = (Percent, 20)
Property: Start_Time = (17-APR-2001,10:00:00.0)
Property: Shed_Duration = 120
Property: Random_Start = 10
Property: Duty_Window = 30
Property: Enabled = TRUE
Property: Full_Duty_Baseline = 250.0
Property: Actual_Start_Time = (17-APR-2001, 10:03:00.0)
Property: Actual_Shed_Amount = 20
Property: Notification_Class = 3
Property: Time_Delay = 60
Property: Event_Enable = {TRUE, FALSE, TRUE}
Property: Acked_Transitions = {TRUE, TRUE, TRUE}
Property: Notify_Type = ALARM
    
```

Property: Event_Time_Stamps = ((12-JUL-01,18:50:21.2),
(*-*-*;*:*:*.*)
(12-JUL-01,19:01:34.0))

Changes to ASHRAE Standard 135.1P:

[Add to 4.5.4, Object Types Supported]

4.5.4 Object Types Supported

...

The standard objects may be any of:

... *Load Control*

[add new 4.5.10.X, "Load Control"]

4.5.10.X Load Control

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

[add]

7.3.2.X Load Control Object Tests

(Tests for the Load Control object to be written when the object itself is resolved.)

[end]

Attachment F: RTP Encoding

Name	RP1011 type	BACnet type	Tag	Len	Value	BACnet Encoding
Tariff	STRUCTURE	STRUCTURE	0	—	—	0E (0000 1 110, 1—class=context spec,110=open struc)
Tariff.id	UnivObjID	Octet string	0	var	2.16.840.1.99	0D 05 60 86 48 01 63 (101=len follows, 05=len, rest=binary rep)
Tariff.numMx	INT8U	U-INT	1	1	1	19 01 (0001tag 1 001len, 01 val)
Tariff.d	Char string	Char string	2	var	Price and sched	2D 0F “price and sched” (0F=15 characters)
Tariff.billToDat	REAL	REAL	3	4	1234.56	3C X’449A51EC” (C=len of 4, binary s,e,f see p.359)
Tariff.curr	Enum	Enum	4	1	134	49 86
Tariff.cmps	Array of Struct	SEQ of Struct	5	var	—	5E (open structure tag)
Tariff.cmps.pricing	STRUCTURE	STRUCTURE	1	—	—	1E (open structure tag)
Tariff.cmps.pricing.curPrice	REAL	REAL	0	4	0.0215	0C X’3CB020C5’
Tariff.cmps.pricing.curTier	INT8S	S-INT	1	1	4	19 04
Tariff.cmps.pricing.name	Char string	Char string	2	var	“Emeter.MX.r”	2D 0B “Emeter.MX.r”
Tariff.cmps.pricing.type	Enum	Enum	3	1	6	39 06
Tariff.cmps.pricing.u	Char string	Char string	4	var	“\$/kWh”	4D 05 “\$/kWh”
Tariff.cmps.pricing.numBins	INT16U	U-INT	5	1	24	59 18
Tariff.cmps.pricing.prices	Array REAL	SEQ of REAL	6	var	0.0241,0.0233,...	6E 44 X’3CC56D5D’ (first array ele, applic tag for REAL=4), 44 X’3CBEDFA4’ (2nd), ..., 44 X’ (24 th) 6F (close array tag)
Tariff.cmps.pricing.scale	REAL	REAL	7	4	1.00	7C X’3F800000’
Tariff.cmps.pricing.bins	array INT32	SEQ of U-INT	8	var	[empty]	8E, 8F (open close) (applic tag for U-INT=2)
						1F (close pricing structure)
Tariff.cmps.refProfile	STRUCTURE	STRUCTURE	2	—	—	2E
Tariff.cmps.refProfile.numIntv	INT16U	U-INT	0	1	24	09 18
Tariff.cmps.refProfile.delT	INT16U	U-INT	1	1	60	19 3C
Tariff.cmps.refProfile.counts	Array REAL	SEQ of REAL	2	var	0.0241,0.0233,...	2E, SEQ of 24 REALs, 2F (? XML gives units of INT32S, but text RP1011 p109 says units “same as defined in pricing component”?)
						2F (close refProfile structure)
Tariff.cmps.schd	STRUCTURE	STRUCTURE	3	—	—	3E
Tariff.cmps.schd.numSchd	INT16U	U-INT	0	1	24	09 18
Tariff.cmps.schd.schds	Array String	SEQ of String	1	var	01012003@00:00,0,1 01012003@01:00,0,2 ... 01012003@23:00,24”, 1F	1E (applic tag for Char String=7) 7D 10 “01012003@00:00,1”, 7D 10 “01012003@01:00,2”, ... 7D 11 01012003@23:00,24”, 1F
Tariff.cmps.schd.editMode	BOOL	BOOL	2	var	0	29 00
Tariff.cmps.schd.addEntry	Array String	SEQ of String	3	var	01012003@03:30,4	3E, 3F (empty)
						3F (close schd structure)
						5F (close cmps SEQ)
						0F (close tariff structure)

XML schema for RTP data structure

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- edited with XMLSPY v5 U (http://www.xmlspy.com) by Martin J Burns (Hypertek, Inc.) -->
-
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation=".\\acsipartial.xsd" />
- <!--
```

RP1011 Models

```

-->
- <xs:complexType name="Tariff">
  - <xs:sequence>
    <xs:element name="id" type="OBJID" />
    <xs:element name="numMx" type="INT8U" />
    <xs:element name="d" />
    <xs:element name="billToDat" type="FLOATINGPOINT32" />
    <xs:element name="curr" type="ENUMERATED" />
  - <xs:element name="cmps">
    - <xs:complexType>
      - <xs:sequence maxOccurs="unbounded">
        <xs:element name="pricing" type="Pricing" />
        <xs:element name="refProfile" type="Profile" />
        <xs:element name="schd" type="RTPSchd" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
- <xs:simpleType name="OBJID">
  - <xs:restriction base="xs:string">
    <xs:pattern value="[12](\[1-9][0-9]*)+" />
  </xs:restriction>
</xs:simpleType>
- <xs:complexType name="Pricing">
  - <xs:sequence>
    <xs:element name="curPrice" type="FLOATINGPOINT32" />
    <xs:element name="curTier" type="INT8" />
    <xs:element name="name" type="VISIBLESTRING" />
    = <xs:element name="type">
      = <xs:simpleType>
        = <xs:restriction base="ENUMERATED">
          <xs:enumeration value="Accumulated" />
          <xs:enumeration value="Maximum" />
          <xs:enumeration value="Minimum" />
          <xs:enumeration value="Differential" />
          <xs:enumeration value="QuarterHour" />
          <xs:enumeration value="Hourly" />
          <xs:enumeration value="Monthly" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="u" />
    <xs:element name="numBins" type="INT16" />
    = <xs:element name="prices">
      = <xs:complexType>
        = <xs:sequence>

```

```

        <xs:element name="price"
            type="FLOATINGPOINT32"
            maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="scale" type="FLOATINGPOINT32" />
= <xs:element name="bins">
    = <xs:complexType>
        = <xs:sequence>
            <xs:element name="bin" type="INT32"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
= <xs:complexType name="RTPSchd">
    = <xs:sequence>
        <xs:element name="numSchd" type="INT16U" />
        = <xs:element name="schds">
            = <xs:complexType>
                = <xs:sequence>
                    <xs:element name="schd" maxOccurs="unbounded"
                        />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="editMode" />
        <xs:element name="addEntry" />
    </xs:sequence>
</xs:complexType>
= <xs:complexType name="Profile">
    = <xs:sequence>
        <xs:element name="numIntv" type="INT16U" />
        <xs:element name="delT" type="INT16U" />
        = <xs:element name="counts">
            = <xs:complexType>
                = <xs:sequence>
                    <xs:element name="count" type="INT32"
                        maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
= <xs:complexType name="Schd">
    = <xs:sequence>
        = <xs:element name="schdHdr">

```

```

- <xs:complexType>
-   <xs:sequence>
-     <xs:element name="isEdit" type="BOOLEAN" />
-     <xs:element name="numTEV" type="INT16U" />
-     <xs:element name="numRDat" type="INT16U" />
-     <xs:element name="sup" />
-   </xs:sequence>
- </xs:complexType>
</xs:element>
- <xs:element name="timEvts" minOccurs="0">
-   <xs:complexType>
-     <xs:sequence>
-       <xs:element name="timEvt" type="TimEvt"
-         maxOccurs="unbounded" />
-     </xs:sequence>
-   </xs:complexType>
</xs:element>
- <xs:element name="rDats" minOccurs="0">
-   <xs:complexType>
-     <xs:sequence>
-       <xs:element name="rDat" type="RDat"
-         maxOccurs="unbounded" />
-     </xs:sequence>
-   </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="TimEvt" />
<xs:complexType name="RDat" />
</xs:schema>

```

Attachment G: Logical Entity Object

DRAFT: David Holmberg 4/23/03

Background

The motivation for this object comes from the utility integration efforts. The electric utility industry uses a communication standard called UCA (in the US—IEEE SCC36, and also IEC 61850 under development as an international standard) that uses name-based addressing (NBA) to communicate within electric substations. If a utility wants to send a native UCA message to a building (such as a real-time pricing RTP signal), how can the message be translated from NBA to the native BACnet flat ObjectID based addressing? The RTP signal is a very complex data structure.

Michael Kintner-Meyer and Martin Burns, in their report, “Utility/Energy Management and Controls System (EMCS) Communication Protocol Requirements” (ASHRAE Research Project 1011-RP, July 1999), presented a way to address these issues with their “Structured View” object. That proposed object contains a property with an array of object property names (standardized) and a second property listing respective pointers (BACnetObjectPropertyReference) to those object properties.

The proposed Logical Entity Object (LEO) has grown out of a discussion primarily between Holmberg, Burns, Isler, and Martocci. This document will try to make a case for the object and develop some case scenarios that demonstrate the LEO’s potential usefulness, before presenting the details of the object.

Problems it Addresses

- BACnet presents objects without hierarchy (flat object space). The proposed Logical Entity Object (LEO) provides a means to make devices and other entities (like distributed but related objects or data) as logical entities visible on the network.
- While the Global Group Object allows reading the values of group members, the LEO allows organizing those members into a structure with branches and leaves, allows reading and writing to those members, and stores information about the structure and members of the structure. It allows holding data locally in the device holding the LEO, or pointing to the data elsewhere on the network. It also allows storing a branch of the structure in another LEO and pointing to that.
- How does an external party like a non-original system integrator, or a service provider (or for that matter the original system vendor) make sense of a large complex facility with tens or hundreds of thousands of objects? There is no network visible way to view the physical, functional, or geographical hierarchies of a building. The LEO allows this information to be made network visible. This may be especially helpful in a building with multiple vendors, or for making system information available to outside partners.
- If one wants to write application code that accesses various data points on a network, the code must reference a proprietary look-up table to get object property IDs. The LEO allows this table to be network visible, providing a layer of indirection, and making the data in the data structure available for other applications.
- If an outside service provider gathers non-network visible data through a gateway from multiple vendors’ controllers, then standardizing the name-binding table promotes inter-operability in gateway design.
- How do I send a message to all the VAV boxes in the east side of a building? I must do a query on a database to find that information. If that information is in the LEO, then it is network visible and any device can use it to distribute messages.
- There *is* no VAV box object in BACnet. The description field of the Device Object is not standardized and not generally useful for a computer to identify a device. Even if it were, there is no function specific format to the Device Object that tells where the thermostat and damper position object properties are. The LEO can be a VAV box object.

Issues to be resolved

1. Cascading LEOs—presently the LEO allows pointing to another LEO in order to show connections (map) type information, but only allows reading and writing to that LEO indirectly from the pointing LEO using the NBA services, and only if reading writing *only* from/to that sub-LEO. Is this how we want it? What if both LEOs are held in the same device? The CASM model is that we put devices together with smaller building blocks. So we have a TemperatureSensorObject and then a ThermostatObject, and then a VAVBoxObject, and then an AHUObject. Each would be a LEO, but then the AHUObject might not have any R/W points since it is simply pointing to another LEO. Perhaps we define structures such that we have both pointers to other LEOs as well as data points from those lower structures that we care about and want visible in the upper-level LEO. Better solution?

2. Timeout—how do we specify timeouts so that the LEO has time to go and collect all the externally held data points, or worse if the LEO points to another LEO that has to collect data? The NBA service may return an error, but the external values may drop in correctly and within the transaction timeout. So the NBA service (or RP) times out, indicates a problem, but there is none. It is a matter of timeouts of cascaded requests which has not addressed so far in BACnet. Perhaps some kind of heartbeat wait signal that would be sent to indicate that a downstream process is still waiting and that the timeout should be delayed.

3. Linkage between utility and building—the utility would like to send one message out to all customers, irregardless of what BCS protocol they are using. The utility could speak UCA (their language) to a gateway at every facility, but that is expensive. They could speak BACnet/IP, except that there will always be customers that don't speak it. Web services seems to be the way that things are moving now to provide a means of moving information cross-platform. It removes the gateway (along with the BACnet specific LEO). BACnet still needs to develop a web services interface to take the incoming message and process it. Take for example the Load Control object (LCO) which has 5 parameters to pass from utility to BCS. The utility passes the parameters to a customer's IP address and port in some XML format, which form would include an identifier like "LCOcontrol". Then there is a LEO (with object name = LCOcontrol) with a structure matching the incoming message that the server (with the web services interface) passes the message to. The LCO then polls the LEO to get most recent control parameters.

4. LEO standardization issue—take the utility case. If the utility writes the LEO and gives it to a customer, then they also must write the application that sits on the customer site to get the data from that LEO and uses it to put together load control orders to on-site devices. So if this is all utility specific, why use BACnet?

- There might be other applications that want access to the data from the utility. Even if it is not in a "universal" format, it is still network visible.
- Ideally it will be standardized by us or some utility group so that anyone can write an application that works here as well as there. So, then why not just write a new Object (like a Real Time Pricing Object) where the BACnet committee standardizes it? Do we want a thousand Clause 12 objects? The hope is that we can have a "fast-track" that doesn't involve a full-blown object, but only a structure. We could even standardize only the basic structure, but allow additional branches to be added on. We only need to standardize a main branch structure and datatypes, and extension rules (if not general).
- The LEO simplifies the job of any gateway—it makes it easier for the utility to move their data structures in BACnet. Now they only change the envelope and they can pass the data structure to a LEO unmodified. Without the LEO they need a gateway to chop up the structure and write it to many locations.

5. What about LEOs listing any upstream LEOs that point to them?

6. Type enumeration: do we allow constructed types? Who enumerates them and how is the list updated and maintained? How are proprietary data types enumerated?

7. What is the DNS application as Dave Robins envisions it? We pass it a name and it returns a BACnet address? Can the LEO as it is written now do the DNS type function that Dave hopes to see? If not, can some changes make it work?

8. Profile issue. Why define the structure in the object? Usually don't need it. But that is why it is kept in Entity_Attributes separate from data. Some applications will use the LEO to discover structure as is the case for a LEO that stores configuration information which is in turn referenced by the OWS application that needs this information to present to the operator. Making the structure network visible also allows other parties to understand and access data, e.g. for commissioning applications. The Profile_Name property is intended for extending the object. But the entity definition is not related. We use some other property (Object_Name or Description?) to put a label on a given Logical Entity definition, which might include a vendor ID, name, and version number. The use of Profile_Name is to communicate extensions to an object, and someone might wish to add additional properties to the LEO, but this is separate from defining an entity.

9. Defining the NBA Read and Write services—see service descriptions following the LEO description.

10. What does **status** mean for a LEO? Can we have it act like the GGO? First, the GGO collects all states of the objects referenced into a property "Member_Status_Flags". Second, intrinsic reporting of the GGO is based on the value of "Member_Status_Flags". But the GGO is fundamentally different as it maintains its present value indicating members' present values. Likewise with status—it [can or does?] constantly monitor the status of members. The LEO only updates present value when requested.

So how can it notify others about status of its external members? It can't, unless we change the function of the LEO to act like the GGO to track member status. This would also help with the problem of time-outs of cascading reads—especially a problem if we have more than one cascaded LEO (i.e. I want to read a value that is in an external LEO that in turn points to an external LEO that in turn points to an externally held value—four reads must be initiated and results returned in sequence). What is easier—requiring LEO to stay current, or figuring out how to wait and not time-out? Or is time of an essence?

We could ignore externally held values and report status as the availability of internal data. But other objects with similar functions (Averaging Object, Schedule Object) don't have Status_Flags.

So, I think we:

- a) change LEO to keep track of member current state (values, status) like the GGO if we think this is necessary, or
- b) have no Status_Flags property and advise those who want to track status of members to add a branch in their structure that has pointers to all the status_flags properties of members for which they want to know status. Then they have status whenever they read PV.

Use cases for the LEO and examples

A. Reading and Writing Data

Read/Write access to data values is via the Present_Value property, and I may be reading values that are held internally (to the device holding the LEO) or externally (in a different device on the network).

1) Global Group Object (GGO) case—Read from a group of properties in externally held objects. Here the LEO points to all these properties. When a device receives a read request (either a RP of the PV or an NBA Read) of the LEO, it initiates reads of all the external values (unlike GGO which keeps an updated cache locally), and then returns the values. Unlike the GGO which returns values of datatype BACnetPropertyAccessResult which includes device, object and property IDs along with property value or access error, the LEO does not include DevObjPropID info since this is available in Entity_Attributes if needed. Additionally, the LEO offers a means to store other information about the group, and to put group members into a hierarchical arrangement. I can also read from a subset of the group.

2) Write data values via LEO to external object properties. The LEO takes a complex structure and flattens it out into the Present_Value array. In order to write to PV, I only need to know the data structure (which element of PV array corresponds to which named element) and the data type of that element. I do not need nor want to concern myself with *where* on the network the data is actually stored—the LEO provides indirection. The present tagged PV format ensures that the datatype is communicated with the value, allowing an array of mixed type.

3) Write to a LEO that holds data internally. An example of this is the utility RTP (real time pricing) case. The utility wants to write the values of a data structure to a LEO. Let's say it is the daily price signal which is an array of 24 price values of type REAL, and an array of schedule values of type BACnetDateTime. This will be transferred as a single array written to the LEO using the NBA Write service. The device holding the LEO will store the values internally. In this case, the LEO needs to have some datatype information stored with the value, but there is no associated DevObjPropID, thus the CHOICE in the BACnetLEReference. The problem is that there is no way in BACnet now to store datatype information. Thus the proposal for a new *BACnetDatatype* datatype that enumerates datatypes for the purpose of internally held data values.

4) Fourth case—reading from an internally held value. The example use case would be a campus weather station that gathers data from many sensors (which sensor outputs may or may not be included in the LEO data structure) and based on some internal algorithm comes up with an outdoor temp, a humidity value, a cloud covering value, etc., and presents these in the LEO as internally held values that I want to read. Whether I use the RP service or the NBA Read service, I am returned an array of type BACnetLEValue which gives me type along with each value in the array.

B. Mapping applications

In these use cases, the PV is perhaps not used. The LEO is instead used to store a hierarchical structure that describes some aspect of the building control system: functional or physical or geographical hierarchy. This information may be used at the OWS for presentation of the BAC system layout to a human or to a third party application that uses it to discover entities.

This use of the object de-emphasizes the data and focuses on the hierarchy of the structure. Many LEOs will have a mixed use—i.e. not every element of PV is a data element. By definition a branch header will not have a data value associated with it; it only shows organization. This is the mapping function.

C. Examples of possible implementations

Here are some ideas for potential applications of the LEO:

Third party software that works on the network without having to have configuration information entered in—it can access network visible configuration info, find devices, know what functionality they have and interact with them. For example, this might work with commissioning. After installation, wiring, building configuration databases (potentially in LEOs), commissioning is still required. If a standard LEO has been used for all VAV boxes (for example) and all of these are referenced in another LEO mapping the system, then a third party vendor can write automatic scripts that locate these standard LEOs and test the standard functionality of the devices. No human is needed to go over a written point list to manually set up point by point tests.

Fault Detection and Diagnostics (FDD). Ideally, an FDD algorithm runs in the little brain of some device, monitoring performance to catch problems (like stuck air damper or drifting temperature sensor). The algorithm will be specific to the device’s capabilities and sensors, and therefore developed by the vendor. So, an HVAC vendor could ship a VAV box controller that presents all the device data in a LEO with fixed names, even as locally configured addresses change. Then the vendor can ship standard fault-detection code that accesses the device data points using name-based addressing, not needing to be concerned with local addresses. If the vendor can access the network from outside, it could do a who-has request for all objects having a certain name that is fixed for their VAV box model, and then write updated FDD code to that device.

Utility interface. One application is to allow an outsider like a utility to pass messages to specialized objects like a Real-Time Pricing (RTP) LEO. The LEO provides a means of transferring a complex data structure into one BACnet object from which it subsequently is available to all devices on the network, and to all applications wanting the data. Somewhere there has to be a “structure-element” to “BACnetObjPropRef” binding. I can do that in a proprietary way or I can do it in a standardized BACnet way. Maybe the utility wants to monitor energy efficiency for compliance purposes, but someone else may want the same data for fault detection purposes. Maybe the utility wants my weather data for local conditions, but I want to use it to make it available on the company WAN. Maybe the utility wants to have some control of the onsite generator, but so do I. Therefore, multiple applications can use data for different purposes.

Weather Station: making a virtual weather station by assembling the various instruments and sensors available on the outside of a building and making this available (as a logical entity) on the network. Internal algorithms can use these sensors as inputs and these (locally calculated and held) values can also be incorporated into the weather station LEO, using any desired structure organization.

Map of the mechanical systems of a building: which AHU is connected to which VAV box and what sensors in which rooms. Where the chillers are and what their components are, etc. Then the example of the building map by room that presents temperature, occupancy, light status, and whatever else control related. If someone calls to report a cold temperature in meeting room C, then the secretary only cares about an interface that would let her adjust the temperature. However, if something is broken, the building manager wants to see which VAV box is malfunctioning and where it is.

Standardized LEOs: If we have universally known LEO data structures, I can write scripts to locate LEOs on a network and then read and write to them without needing to discover their structures or know their location. This will open a myriad of new applications.

12.X Logical Entity Object

The Logical Entity object type provides a container that holds complex data structures and which also describes the structure within the object. The data in the structure may be held locally in the device implementing the LEO, or the data values may be held externally in any object on the network. Data values are accessed by reading the Present_Value property, at which time the device holding the LEO reads any externally held values. Data values may also be written to Present_Value. The structure definition is contained in the BACnetLEAttributes property. The structure may have any number of branches and leaves. Only a leaf holds a data value. The LEO also offers the ability to have an externally held branch by pointing to another LEO. In this way it is possible to have a LEO with a structure that only points to other structures and serves as a system map, rather than as a data container. A typical LEO will have a data structure with branch headers (organization information) and leaves (data values).

The data within a LEO is accessible via Read Property and Write Property services acting on Present_Value. Alternatively, the data can be accessed via the Name Based Addressing Read service (NBA Read) and the Name Based Addressing Write service (NBA Write). With these services, the data structure can be navigated in a hierarchical manner by name, rather than by reference to the array index of Present_Value. In addition, this allows a subset of the Present_Value array to be returned corresponding to a branch of the data structure held in the logical entity.

Table 12-Y. Properties of the Logical Entity Object

Property Identifier	Property Datatype	Conformance Code
Object_Identifier	BACnetObjectIdentifier	R
Object_Name	CharacterString	R
Object_Type	BACnetObjectType	R
Description	CharacterString	O
Present_Value	Array of BACnetLEValue	R
Entity_Attributes	Array of BACnetLEAttributes	R
? Status_Flags	BACnetStatusFlags	R
? Member_Status_Flags	BACnetStatusFlags	R
? Event_State	BACnetEventState	R
? Reliability	BACnetReliability	O
? Out_Of_Service	BOOLEAN	R
? Time_Delay	Unsigned	O ¹
? Notification_Class	Unsigned	O ¹
? EventEnable	BACnetEventTransitionBits	O ¹
? AckedTransitions	BACnetEventTransitionBits	O ¹
? NotifyType	BACnetNotifyType	O ¹
? EventTimeStamps	BACnetARRAY[3] of BACnetTimeStamp	O ¹
Profile_Name	CharacterString	O

¹ These properties are required if the object supports intrinsic reporting.

12.X.1 Object_Identifier

This property, of type BACnetObjectIdentifier, is a numeric code that is used to identify the object. It shall be unique within the BACnet Device that maintains it.

12.X.2 Object_Name

This property, of type `CharacterString`, shall represent a name for the object that is unique within the BACnet Device that maintains it. The minimum length of the string shall be one character. The set of characters used in the `Object_Name` shall be restricted to printable characters.

12.X.3 Object_Type

This property, of type `BACnetObjectType`, indicates membership in a particular object type class. The value of this property shall be `LOGICAL_ENTITY`.

12.X.4 Description

This property, of type `CharacterString`, is a string of printable characters whose content is not restricted.

12.X.5 Present_Value

This property, an array of type `BACnetLEValue`, is used to read and write the values of the Logical Entity's data structure. These values may be held locally within the device hosting the LEO or externally in the property of any other object on the network, as indicated in the `Entity_Attributes` property. There is no caching function as with the Global Group Object—any Read Property request of `Present_Value` shall initiate a read of all externally held property values in order to form a reply with present values. Any Write Property request to `Present_Value` shall initiate writes to the properties of externally held values. The array index of `Present_Value` shall correspond to the array index of `Entity_Attributes`. Each element of the `Present_Value` array corresponds to the element of `Entity_Attributes` at the same index.

```
BACnetLEValue ::= CHOICE {
    value  [0] ABSTRACT-SYNTAX.&Type,
    error  [1] Error
}
```

Note: some elements of the `Present_Value` array will correspond to structure headings (branch names, identified by `BACnetLEReference CHOICE = 0`) within the data structure that have no associated value. In this case the value contained at that array index in `Present_Value` shall be “null”.

12.X.6 Entity_Attributes

This property holds an array of type `BACnetLEAttributes` that contains the definition of the entity structure and optionally gives location and descriptive information for individual elements. Within the `BACnetLEAttributes` sequence, the “name” parameter holds an arbitrary name for each node of the structure, where a node is defined as a leaf or the header for a branch. A branch may have an arbitrary number of nodes within it that can be leaf or branch nodes. In order to allow name based addressing, all nodes at the same level in the data structure shall have unique names. This implies that leaf node or branch node names directly beneath the same branch node must have unique names, while a branch may contain nodes with names the same as contained by other branches. It may be desired to have two or more leaves point to separate LEOs that are of some standard identical structure. In this case the “description” parameter field may be used to give the standard LEO name.

The “reference” parameter of the `BACnetLEAttributes` sequence is of type `BACnetLEReference` and holds one of four values: the *size* of a branch, type unsigned, where the size is defined as the number of nodes at the level directly beneath the branch header level, but not including nodes at lower levels; the *LEOaddress* (`BACnetDeviceObjectPropertyIdentifier`) where a branch contained in a sub-LEO is held externally; the *address* (`BACnetDeviceObjectPropertyIdentifier`) at which an externally held value is

located; or the *type* (BACnetDataType) of an internally held value. These options (size, LEOaddress, address, type) are mutually exclusive: a branch header node will have a name and size, unless the branch is contained in an external LEO, in which case there is only a pointer to that LEO and no size; an externally held value will have a name and address; and a locally held value will have a name and type. A LEOaddress reference will always point to the Entity_Attributes property of the external LEO.

“Location”, of type BACnetLocation, is used to indicate the geographical location of an entity element. This may be useful for identifying the locations of parts of a distributed system of devices or of rooms in a facility or of buildings on a campus. The definition of BACnetLocation allows for supplying a string as well as GPS (latitude N, longitude E, height) coordinates in decimal degrees notation (lat, long) and meters (height), as well as generic coordinates (x,y,z) relative to some local reference (which may be a building GPS reference stored in the GPS field).

Datatype construct for Entity_Attributes:

```
BACnetLEAttributes ::= SEQUENCE {
    name           [0]  CharacterString,
    reference      [1]  BACnetLEReference,
    location       [2]  BACnetLocation OPTIONAL,
    description    [3]  CharacterString OPTIONAL
}
```

-- new definitions needed:

```
BACnetLEReference ::= CHOICE {
    size           [0]  Unsigned,           -- Number of branches and leaves below
    this structure node
    LEOaddress     [1]  BACnetDeviceObjectPropertyReference, -- location of external LEO
    branch
    address        [2]  BACnetDeviceObjectPropertyReference, -- location of externally held value
    type           [3]  BACnetDatatype     -- data type of value held locally
}
```

```
BACnetLocation ::= SEQUENCE {
    name           [0]  CharacterString,
    GPS position   [1]  BACnet3DPosition OPTIONAL,
    XYZ position   [2]  BACnet3DPosition OPTIONAL
}
```

```
BACnetDatatype ::= ENUMERATED {
    Null           (0),
    Boolean        (1),
    Unsigned       (2), -- Unsigned Integer
    Signed         (3), -- Signed Integer (2's complement notation)
    Real           (4), -- ANSI/IEEE-754 floating point
    Double         (5), -- ANSI/IEEE-754 double precision floating point
    OctetString    (6),
    CharacterString (7),
    BitString      (8),
    Enumerated     (9),
    Date           (10),
}
```

```

Time          (11),
BACnetObjectIdentifier (12),
...?????
}

```

```

BACnet3Dposition ::= SEQUENCE {
X Real,
Y Real,
Z Real
}

```

The elements of Entity_Attributes may be viewed as a table that describes the data structure of the logical entity. An example entity structure is given in Table 12-Y+1. Indentations in the “name” column are added for clarity in this example. In this case a cluster of data is received from a utility with price and schedule information. This cluster is written to the “Tariff” branch of the data structure, and all data values in the Tariff branch are held locally.

Table 12-Y+1. Example of an RTP (Real Time Pricing) entity structure definition

Entity_Attributes				Present_Value
name	reference	location	descrip	value
RTPcontrol	address = BACnetDevObjPropRef			TRUE
Get_RTP	size = 3			<i>null</i>
RTP1	LEOaddress = BACnetDevObjPropRef	Bldg1 OWS		<i>null</i>
RTP2	LEOaddress = BACnetDevObjPropRef	Bldg2 OWS		<i>null</i>
RTP3	LEOaddress = BACnetDevObjPropRef	Bldg3 OWS		<i>null</i>
Tariff	size = 4			<i>null</i>
ID	type = Octet String			1.19.25.789
CurrentBill	type = Real			175.02
Pricing	size = 3			<i>null</i>
Name	type = Char String			“Meter1”
Units	type = Char String			“\$/kWh”
Prices	size = 4			<i>null</i>
Price1	type = Real			0.0240
Price2	type = Real			0.0262
Price3	type = Real			0.0295
Price4	type = Real			0.0258
Schedule	size = 4			<i>null</i>
Schd1	type = Time			0:00:00.00
Schd2	type = Time			8:00:00.00
Schd3	type = Time			12:00:00.00
Schd4	type = Time			16:00:00.00

As a second example, consider a Weather Station LEO that gathers together various sensor data from around the exterior of the building and presents it in the LEO, as shown in Table 12-Y+2. Here the reference column mainly points to externally held values (sensor AV Objects held by the device connected to the respective sensors). However, the “Outdoor Temp” element is calculated locally using some internal algorithm from the input of several sensors.

Table 12-Y+2. Example of a weather station entity structure definition

Entity_Attributes				Present_Value
name	reference	location	descrip	value
Outdoor Temp	type = REAL			85.5
Windspeed	address = BACnetDevObjPropRef	WS, rooftop1		2.5
Wind Dir	address = BACnetDevObjPropRef	WS, rooftop1		282.2
Rel Humid	address = BACnetDevObjPropRef	WS, rooftop1		65
Baro Pressure	address = BACnetDevObjPropRef	WS, rooftop1		29.4

12.X.7 Profile_Name

This optional property, of type CharacterString, is the name of an object profile to which this object conforms. To ensure uniqueness, a profile name must 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 profile name shall indicate the organization that publishes and maintains the profile document named by the remainder of the profile name. This vendor identifier need not have any relationship to the vendor identifier of the device within which the object resides.

A profile defines a set of additional properties, behavior, and/or requirements for this object beyond those specified here. This standard defines only the format of the names of profiles. The definition of the profiles themselves is outside the scope of this standard.

NBA Services

How does Name-Based Addressing work?

With RP and WP I need to address a property by DevObjPropID, and then array index. This ignores the hierarchical structure contained within the array. NBA addresses the data in the present_value array from the viewpoint of the hierarchy described in Entity_Attributes. I need to know DeviceID and either the LEO ObjectID or the LEO Object_Name. Beyond that I can read or write data in the structure by using a string of names that consist of branch headers starting at the top level and proceeding down to the data I want to read or write, e.g. "topLevelBranch.subBranch[subSubBranch.leaf]". I can read/write a single value (a leaf node) or I can read/write a sub-set of data (a branch). If a subBranch is actually an external LEO, the name string can continue into that LEO's data structure by name and be used to read and write data to the external LEO.

Why do we need Name-Based Addressing services?

First, let it be said that we can access all the information of the LEO without using NBA. However, there are some potentially very useful benefits from using NBA, as follows:

- We may define the LEO in such a way that we can use available RP and WP services to pass a sub-set of the data in Present_Value identified by index (e.g. use WP to pass PV an array of data values AND an array index N, meaning write the array sub-set to PV starting at index N). However, it seems more natural to address this sub-set by a meaningful name as a unit (a branch), rather than by an index number.
- NBA allows me the potential to write to a sub-LEO. I can use a higher level LEO to access all my lower level LEOs and read from or write to them, rather than reading/ writing to them directly. This is an additional layer of indirection that may be useful.
- The utilities want NBA since their communication protocol works that way.
- NBA provides additional indirection when compared to using RP/WP on the LEO: NBA allows addressing a LEO by Object_Name rather than ObjectID, and it also allows retrieving data by branch names rather than array index. Potentially multiple structures could have a branch sub-structure in common but otherwise be different. This could change array index while the naming tree could be more easily held unchanged.
- Names are more suitable for interacting with humans. Somewhere there needs to be a conversion from name to device address. NBA services allow an application to use names to get data rather than first going to a table to get device address for a given name. This also benefits interoperability since an application can be written to access data in a non-proprietary way.

15.X Name Based Addressing Read (NBA-R) Service

The NBA-R service is used by a client to retrieve data from a Logical Entity object (LEO) using the names given in the branch structure described in the LEO's Entity_Attributes property. A single value (leaf) can be read, or some sub-set (branch) of the data values of the data structure, or the entire data structure. In addition, the naming string can include a branch name corresponding to an external LEO and be used in this way to read the present value of a sub-LEO. Continuing further, the NBA Read service can be used to read a sub-set (branch) of the data within a sub-LEO, etc.

The service requires the LEO's Object Identifier or its Object Name. No Property ID is required.

15.X.1 Structure

The structure of the NBA-Read service primitives is shown in Table 15-Y. The terminology and symbology used in this table are explained in 5.6.

Table 15-Y Structure of NBA-Read Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	C	C(=)		
Object Name	C	C(=)		
NBA Address	M	M(=)		
Result (+)			S	S(=)
Object Identifier			C	C(=)
Object Name			C	C(=)
NBA Address			M	M(=)
Array of BACnetLEValue			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.X.1.1 Argument

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

15.X.1.1.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose property is to be read and returned to the requesting client BACnet-user. If this parameter is present, then Object Name shall not be used.

15.X.1.1.2 Object Name

This parameter, of type CharacterString, may be used in place of the Object Identifier to identify the Logical Entity Object containing the data structure.

15.X.1.1.3 NBA Address

This parameter, of type BACnetNBAStrng, shall convey the name of the data value or branch to be read from the data structure.

15.X.1.2 Result (+)

The 'Result (+)' parameter shall indicate that the service request succeeded. A 'Result (+)' response shall be returned if any of the values of the request are successfully read.

15.X.1.2.1 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall identify the LEO object whose data values have been read and are being returned to the client BACnet-user. This parameter shall be returned only if the request was made using the Object Identifier choice.

15.X.1.2.2 Object Name

This parameter, of type CharacterString, shall identify the LEO object whose data values have been read and are being returned to the client BACnet-user. This parameter shall be returned only if the request was made using the Object Name choice.

15.X.1.2.3 NBA Address

This parameter, of type BACnetNBAStrng, shall convey the name of the data value or branch to be read from the data structure.

15.X.1.2.4 Array of BACnetLEValue

If access to the specified LEO was successful, this parameter shall indicate, for each requested array element, the success or failure of the read attempt. If the read attempt was successful for a given element, the BACnetLEValue value at that element shall be the requested value, of datatype appropriate for the requested value. If the read attempt failed, the BACnetLEValue value at that element shall be an error code indicating the reason for the access failure.

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.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) and 'Error Code'. See Clause 18.

15.X.2 Service Procedure

After verifying the validity of the request, the responding BACnet user shall attempt to access the requested value(s) of the LEO. If access to any of the values is successful, a 'Result (+)' primitive shall be issued, which returns for each element either the requested value or the reason the attempt failed. If the read attempt fails in entirety, a 'Result (-)' primitive shall be issued indicating the reason for the failure.

Clause 21 Encoding

NBACRead-Request ::= SEQUENCE {
 objectIdentifier [0] BACnetObjectIdentifier OPTIONAL,
 objectName [1] CharacterString OPTIONAL,
 NBAAAddress [2] BACnetNBAStrng,
 }

NBACRead-ACK ::= SEQUENCE {
 objectIdentifier [0] BACnetObjectIdentifier OPTIONAL,
 objectName [1] CharacterString OPTIONAL,
 NBAAAddress [2] BACnetNBAStrng,
 arrayOfBACnetLEValue [3] SEQUENCE OF BACnetLEValue
 }

BACnetNBAStrng ::= SEQUENCE OF CharacterString {
 hierarchicalName [0] CharacterString
 }

BACnetLEValue ::= CHOICE {
 value [0] ABSTRACT-SYNTAX.&Type,
 error [1] Error
 }
 (This is the datatype for the LEO Present_Value)

15.X Name Based Addressing Write (NBA-W) Service

The NBA-W service is used by a client to write data to a Logical Entity object (LEO) using the names given in the branch structure described in the LEO's Entity_Attributes property. A single value (leaf) can be written, or some sub-set (branch) of the data values of the data structure, or the entire data structure. In addition, the naming string can include a branch name corresponding to an external LEO and be used in this way to write to the present value of a sub-LEO. Continuing further, the NBA Write service can be used to write to a sub-set (branch) of the data within a sub-LEO, etc.

The service requires the LEO's Object Identifier or its Object Name. No Property ID is required.

15.X.1 Structure

The structure of the NBA-Write service primitives is shown in Table 15-X. The terminology and symbology used in this table are explained in 5.6

Table 15-X Structure of NBAWrite Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Object Identifier	C	C(=)		
Object Name	C	C(=)		
NBA Address	M	M(=)		
Data Value	M	M(=)		
Result (+)			S	S(=)
Array of BACnetLEValue			M	M(=)
Result (-)			S	S(=)
Error Type			M	M(=)

15.X.1.1 Argument

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

15.X.1.1.2 Object Identifier

This parameter, of type BACnetObjectIdentifier, shall provide the means of identifying the object whose property is to be modified. If this parameter is present, then Object Name shall not be used.

15.X.1.1.3 Object Name

This parameter, of type BACnetCharacterString, may be used in place of the Object Identifier to identify the Logical Entity Object containing the data structure.

15.X.1.1.4 NBA Address

This parameter, of type BACnetNBAStrng, shall convey the name of the data value(s) to be written to the Logical Entity Object.

15.X.1.1.5 Data Value

The Data Value parameter is an array of datatype BACnetLEValue. If access to the specified values of the specified object(s) is available, the value(s) contained in this parameter shall be used to replace the specified value(s) held in the LEO's Present_Value property. For each element of the array, the BACnetLEValue choice 'value' shall be taken, with the datatype appropriate for the element of the data structure to be written.

15.X.1.2 Result (+)

The 'Result (+)' parameter shall indicate that the service request succeeded. A 'Result (+)' response shall be returned if any of the values of the request are successfully written.

15.X.1.2.1 Array of BACnetLEValue

This parameter, an array of datatype BACnetLEValue, is returned with a 'Result (+)' response. If the write for a given element was successful, then choice 'value' is taken with the value set to 'null'. If there is an error writing a value of a given element, then the error type is returned in the array at that element index.

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.1.3.1 Error Type

This parameter consists of two component parameters: (1) an 'Error Class' and (2) and 'Error Code'. See Clause 18.

15.9.2 Service Procedure

After verifying the validity of the request, the responding BACnet user shall attempt to write the data value(s) given in 'Data Value' to the locations specified in the Entity_Attributes property of the specified LEO. If the write attempt for any of the elements is successful, a 'Result (+)' primitive shall be issued. If the modification attempt fails in its entirety, a 'Result (-)' primitive shall be issued indicating the reason for the failure.

Clause 21 Encoding

```
NBWrite-Request ::= SEQUENCE {
    objectIdentifier          [0] BACnetObjectIdentifier OPTIONAL,
    objectName                [1] CharacterString OPTIONAL,
    NBAddress                 [2] BACnetNBAStrng,
    DataValue                 [3] array of BACnetLEValue
}
```

```
NBWrite-ACK ::= SEQUENCE {
    arrayOfBACnetLEValue     [3] SEQUENCE OF BACnetLEValue
}
```