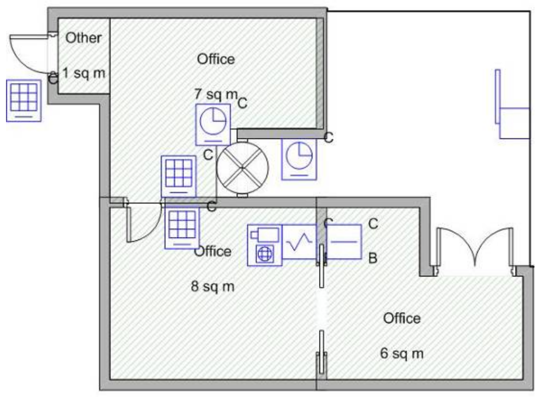
Physical Access Control with BACnet
White Paper



Abstract
This document outlines the use of existing definitions of as well as extensions to ANSI/ASHRAE 135-2004 BACnet to enable the implementation of and interfacing to Physical Access Control Systems using the BACnet data model and services. The extensions shall enable inter-domain interoperability and common user interfaces. They also enable the construction of Physical Access Control Systems containing devices of different product vendors and product generations. The extensions include new object types, alarming and logging.

This document represents the ongoing work of the SSPC135 LSS-Working Group, and the basis for proposed changes to the BACnet standard. The contents are subject to change. It is anticipated that some content of the document will become an informative Annex to the BACnet Standard.

Contributors
Bernhard Isler, Senior system architect, Fire Safety and Security Products, Siemens Building Technologies, Zug, Switzerland.
Hans-Joachim Mundt, Head of standards, Siemens Building Technologies, Karlsruhe, Germany
Rob Zivney, Vice President of Marketing, HIRSCH Electronics, Santa Ana, CA, USA
David Ritter, Senior software developer/technical lead for Access Control Products, Delta Controls, Surrey, BC, Canada.
Stephen Treado, Ph.D., P.E., Mechanical engineer, Building and Fire Research Laboratory with the NIST, Gaithersburg, MD, USA
Attendees of LSS-Working Group meetings since January 2003
Reviewers of HJM-002, the conceptual work document of the LSS-Working Group and base for this white paper

Contents

Figures

1. Introduction

1.1 Purpose

BACnet is extending its scope to include Physical Access Control Systems (PACS) and functionality. This takes BACnet beyond HVAC, Lighting, Energy Consumption Management, Fire, etc..

Physical Access Control Systems (PACS) have requirements to the data model, services, alarming and logging, which are not well covered by BACnet. Although some manufacturers already have implementations based on BACnet or provide BACnet interfaces to such systems by using the current standard and proprietary extensions, a more comprehensive data model and services capability is required to allow:

- Interfacing PACS with systems of other BACnet domains (e.g. HVAC, Fire, Elevators, Lighting, etc.) for inter-domain interoperability,
- Interfacing PACS with systems of non-BACnet domains (e.g., Human Resource Management Systems)
- Unified user interfaces for all systems of a building (e.g. common workstation with common data representation and operation).
- Construction of Physical Access Control Systems containing devices of different product vendors and product generations.
- Harmonization with other industry standards, such as developed by the Security Industry Association (SIA).

This document defines the basic concept of Physical Access Control Systems from a BACnet perspective, and outlines the use of existing definitions and new extensions to ANSI/ASHRAE 135-2004 BACnet to model such systems. This will result in the ability to implement a PACS or a gateway to a PACS by using BACnet as the communication protocol and data model framework.

1.2 Scope

This document covers the extensions to BACnet for PACS and those elements of intrusion detection that are associated with doors. The BACnet data modeling concepts serve as a framework and base for these extensions; and the overall PACS model will use existing definitions as well. However, the existing BACnet definitions are not replicated herein.

Extensions for other security systems such as Closed Circuit TV (CCTV, Digital Video, or video surveillance) are not part of this document.

Application functionality is not required to be exposed through the data model. Such unexposed functionality is addressed as a local matter, according to BACnet convention.

As is becoming industry practice, the term Physical Access Control Systems or PACS is used herein to differentiate from Logical Access Control Systems which provide access to computers, information and networks. However, with “convergence” and one card solutions becoming more common, BACnet has embraced concepts that are expected to work well in a Logical Access implementation.

1.3 Audience

This document is intended for:

- ASHRAE SSPC 135 Life Safety & Security Working Group (LSS-WG)
- ASHRAE SSPC 135
- BACnet users, integrators and manufacturers
- Access Control Industry

1.4 Related BACnet and Working Group Documents

[STD]	ANSI/ASHRAE 135-2004, BACnet	BACnet Standard 2004
[ADF]	Addendum *f* to [STD] (Approved by ASHRAE March 25, 2007)	Addendum *f* to BACnet Standard 2004: *f*-1: Add a new Access Door object type
[ADJ]	Addendum *j* to [STD] (4th Public Review Version, September 2008)	Addendum *j* to BACnet Standard 2004: *j*-1: Add a new Access Point object type *j*-2: Add a new Access Zone object type *j*-3: Add a new Access User object type *j*-4: Add a new Access Rights object type *j*-5: Add a new Access Credential object type *j*-6: Add a new Credential Data Input object type *j*-7: Add a new ACCESS_EVENT event algorithm *j*-8: Add a new Annex X BACnet encoding rules for authentication factor values

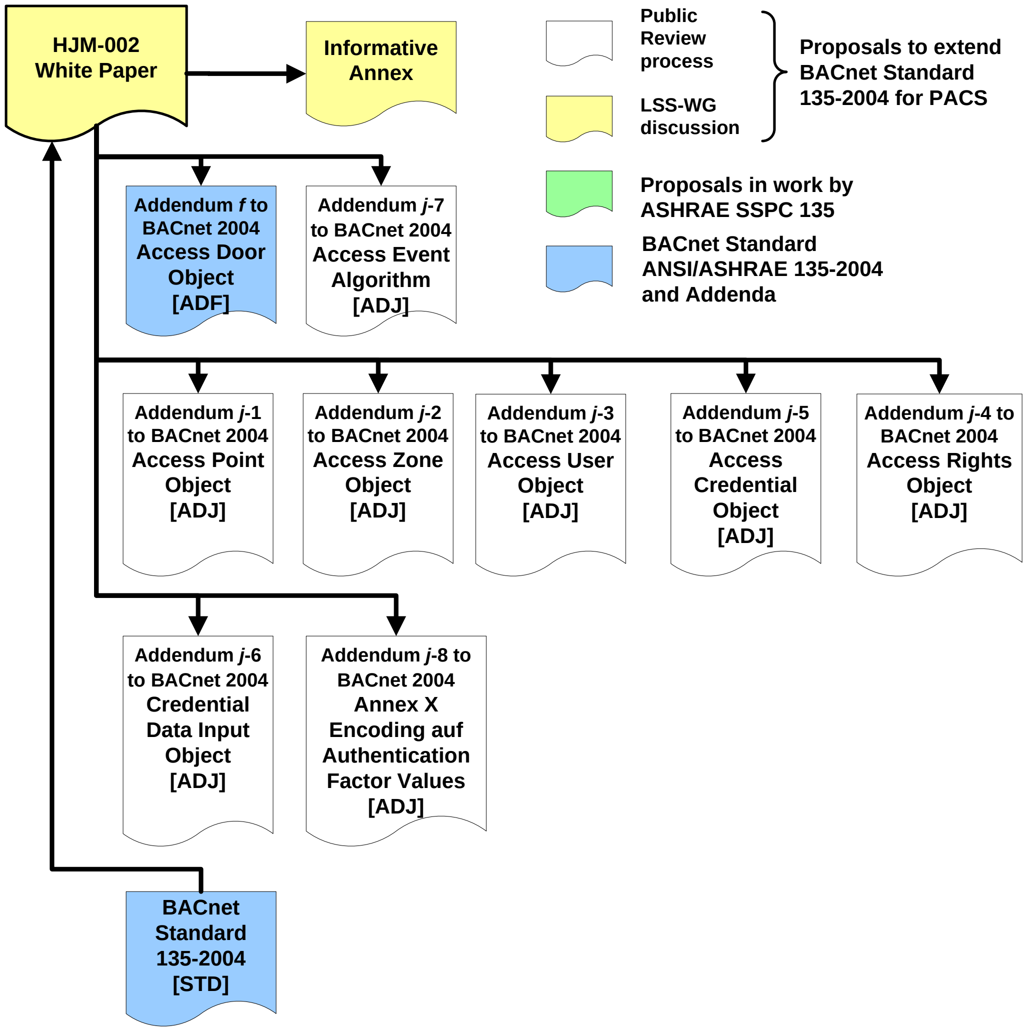

HJM-002 White Paper
Informative Annex
Public Review process
LSS-WG discussion
Proposals to extend BACnet Standard 135-2004 for PACS
Proposals in work by ASHRAE SSPC 135
BACnet Standard ANSI/ASHRAE 135-2004 and Addenda
Addendum f to BACnet 2004 Access Door Object [ADF]
Addendum j-7 to BACnet 2004 Access Event Algorithm [ADJ]
Addendum j-1 to BACnet 2004 Access Point Object [ADJ]
Addendum j-2 to BACnet 2004 Access Zone Object [ADJ]
Addendum j-3 to BACnet 2004 Access User Object [ADJ]
Addendum j-5 to BACnet 2004 Access Credential Object [ADJ]
Addendum j-4 to BACnet 2004 Access Rights Object [ADJ]
Addendum j-6 to BACnet 2004 Credential Data Input Object [ADJ]
Addendum j-8 to BACnet 2004 Annex X Encoding auf Authentication Factor Values [ADJ]
BACnet Standard 135-2004 [STD]

Figure 1–1, Document Structure

1.5 Related Non-BACnet Documents

[FIPS-201]	FIPS Publication 201-1, June 23, 2006	Federal Information Processing Standards Publication, Personal Identity Verification (PIV) of Federal Employees and Contractors. Issued by the NIST Computer Security Division March 2006, and change 1, June 23, 2006. Documents available at http://csrc.nist.gov/publications/PubsFIPS.html
[RBAC]	ANSI/INCITS 359-2004	Information Technology - Role Based Access Control Inter-National Committee for Information Technology Standards (formerly NCITS). Related documentation available at http://csrc.nist.gov/rbac/

1.6 Disclaimer

This document is provided for informational purposes only and the information herein is subject to change without notice. This document does not redefine any standards documents. The content represents one of many possible interpretations of BACnet 2004, its addenda and any related documents. The authors of this document do not provide any warranties covering and specifically disclaims any liability in connection with this document.

2. Conceptual Overview

In order to understand what the BACnet standardization of PACS is about and what it covers, required prerequisites are to understand the application functionality of a PACS and to know the basic BACnet application model.

The PACS application functionality is decomposed into elements which can be mapped to the BACnet application model, and the dependencies and interfaces between these elements are identified. The data and service model of these interfaces is defined by using and extending the current BACnet data model (i.e. object types and properties). No new BACnet application services are added.

The BACnet interfaces defined include the new PACS models and reuse existing BACnet models. The interfaces are not specific to any client, and allow BACnet capable clients to easily be extended to support PACS. They support both manual operations through user interface devices and machine-to-machine interaction.

Manual operation covers tasks carried out manually by operators at a workstation or any other type of user interface.

- Browsing
- Operation
- Configuration
- Event Management
- Device Management
- Etc.

Machine-to-Machine Interaction covers tasks carried out by machines autonomously, without operator intervention. This could be devices of a PACS, or devices of other domains.

- Alarm and event logging
- Log archiving
- Automation
- Commanding
- Scheduling
- Trending
- Etc.

3. Physical Access Control

A Physical Access Control System (PACS) manages passage of people or assets through an opening in a secure perimeter based on a set of rules.

Users requesting passage through a secured perimeter are individual persons, assets or even a group of persons and / or assets. The general term used for this is access user. An access user has credentials assigned. An access user can have more than one physical credential if for instance the user visits different sites that implement different vendor technologies resulting in having to carry multiple cards.

Credentials may be physical credentials, e.g. cards carried by a person or bar code stickers at an asset. Physical credentials contain authentication factors, such as simple numbers or structured data. Physical credentials implicitly identify the user. Access user properties may be used as biometric authentication factors, e.g. fingerprint, hand-shape, or iris. Biometric authentication factors are typically processed and condensed into a biometric template, both for authentication and storage. Biometric authentication factors allow an explicit identification of a user. An access user also may know an authentication factor, typically a personal identification number (PIN). Authentication factors known by an access user allow implicit identification of access users only.

A credential reader can read one or more authentication factors and forwards the data to the authentication and authorization function. A read of an authentication factor typically starts the authentication and authorization process to decide on access. Different authentication factors may be required to be read for multi-factor authentication.

The opening in the secure perimeter is referred to as an access door. An access door can be any controlled opening in the secure perimeter including a door, a window, or a roof hatch. An access door in access control is a collection of physical devices. These physical devices typically include: An electromechanical lock, a door monitor switch (alarm contact), a request to exit device (REX or RQE) such as a pushbutton or motion detector. Special access doors may allow passage in one direction only.

The area or space enclosed by the secure perimeter is called a zone. A zone can have one or multiple doors that provide passage into or out of the zone. A door is always associated with two and only two distinct zones, one on each side. Where there are several zones at a site, a user typically moves from one zone to another.

The rules used to manage access through a secure perimeter are based on the concepts of authentication and authorization. Authentication is a process of proving identity based on comparison of the access user’s current authentication factors with those previously enrolled and maintained in the PACS database. Authorization is a process of checking one or more pre-defined validation rules, such that passing ALL validation checks is a pre-requisite to being granted access. A basic example of authorization validation checks would be the right to pass through a specific door at a specific time. It is not unusual to have numerous additional validation checks that might include: Two person rule, expiration date not yet passed, anti-passback, holiday, use count limits not reached, etc. The order that the elements of authentication and authorization are processed is a local matter.

An access point is a logical construct that binds authentication and authorization for passage to a geographical location. From an access zone viewpoint, an entry (or ingress) access point is the point to enter the access zone, while an exit (or egress) access point is the point to leave the access zone. An entry access point is related to the credential readers outside of the access door, behind which the access zone to enter is. An exit access point is related to the credential reader inside the access door.

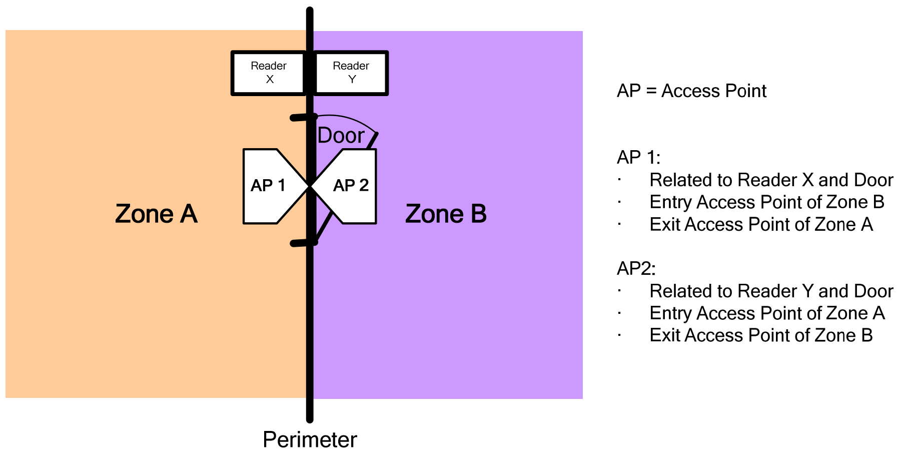

Reader X
Reader Y
Door
AP 1
AP 2
Zone A
Zone B
Perimeter
AP = Access Point
AP 1:
· Related to Reader X and Door
· Entry Access Point of Zone B
· Exit Access Point of Zone A
AP2:
· Related to Reader Y and Door
· Entry Access Point of Zone A
· Exit Access Point of Zone B

Figure 3–1, Access Point Relations

The access point determines which authentication factors are required for the authentication policy in effect. For instance, a card may be required during the day, but both a card and PIN are required after hours.

Access grants and denials, called transactions, are reported as events along with door forced open alarms (actually a shared Intrusion Detection function), and status such as a door being open and/or locked. A fundamental access control function is to temporarily “mask” (not shunt) a door forced open alarm for a predetermined time interval following an access grant. It is important to note that for an access zone to be “Secured”, the door(s) must be closed, locked and alarms unmasked.

4. BACnet Application Model Overview

The BACnet application model is based on processes running in devices. Processes implement and perform application functionality. A process may expose an interface accessible by processes of remote devices. The data model of exposed interfaces is based on BACnet objects. BACnet access services are used by remote processes to access BACnet objects, while the BACnet objects of the interface notify subscribed remote processes about changes of values and events.

Note: The BACnet Application Model is part of the BACnet framework. For further details of this framework, especially the underlying communication protocol stack and networking see section "Functions and Features Inherent in the BACnet Framework", or the BACnet standard [STD].

4.1 BACnet Processes

BACnet processes implement and perform application functionality. BACnet does not specify what the processes perform, or how, as these are considered a "local matter". Multiple processes may run in a physical device.
Seen from outside of a device, processes may take a server or client role or both at the same time. In the server role, they expose a BACnet Process Interface to the network (the lollipop). In the client role, they use an exposed interface.

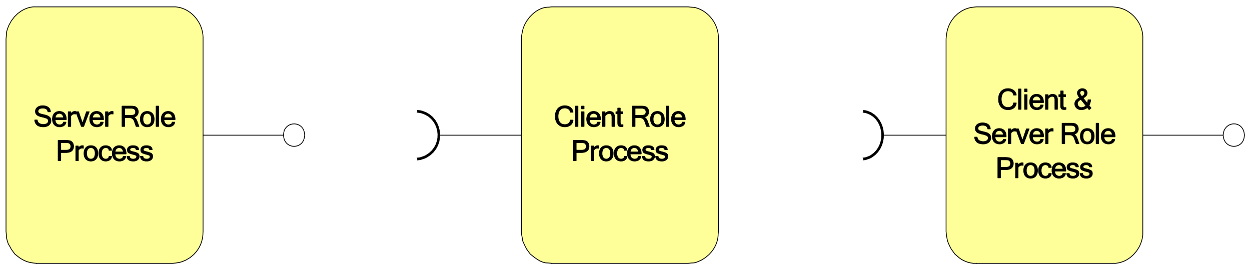


Figure 4–1, Roles of BACnet Processes

4.2 BACnet Process Interface

A BACnet process interface, as exposed to the network by a server role process, is the communication peer for remote client role processes to communicate with over the BACnet network. It provides standardized access to the functionality of a process. Multiple client role processes may communicate with one BACnet process interface. The data exposed in a BACnet process interface is a collection of BACnet objects. BACnet application services are used in the communication between the BACnet objects and a remote client role process.

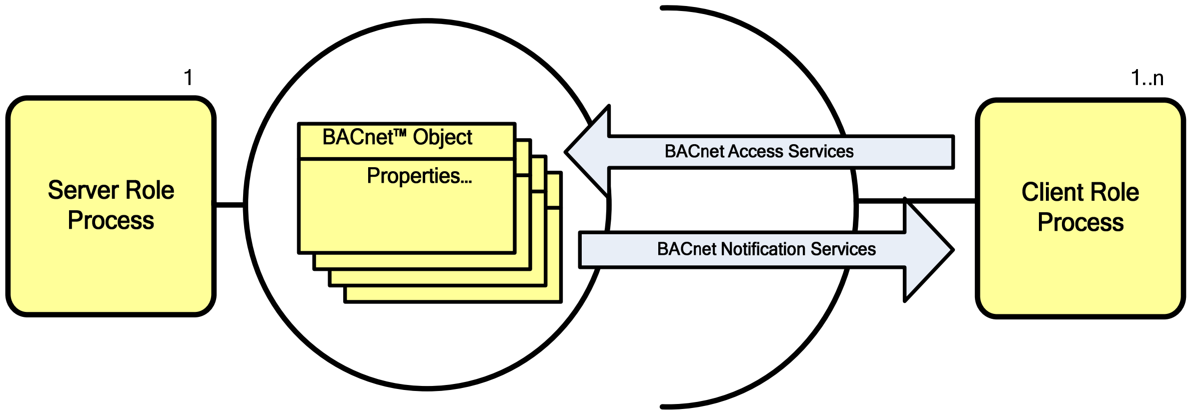
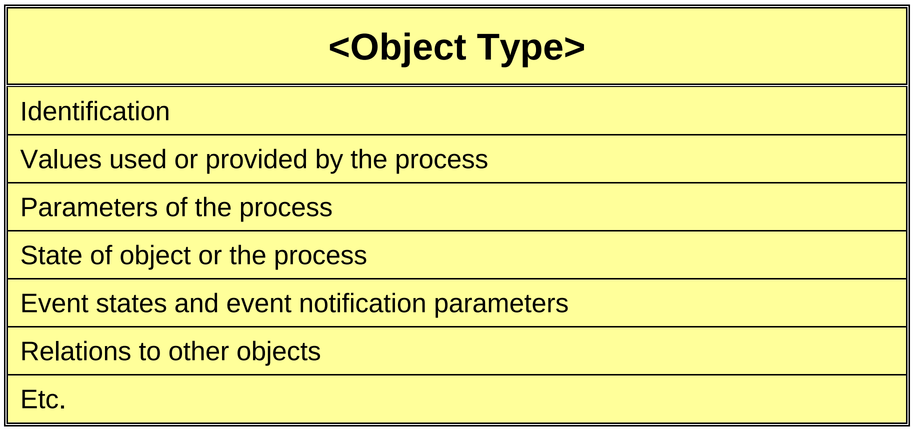


Figure 4–2, BACnet Process Interface

4.3 BACnet Objects

BACnet objects are used to structure and identify data of a BACnet process interface. The BACnet standard defines object types and the properties they shall or may contain. The application functionality performed in a device, and through this the set of object instances in a physical device, is defined individually by vendors. The overall set of objects of the exposed interfaces of a device models the externally visible overall functionality of the device. BACnet mandates that one instance of a Device Object must be present in a device, to represent the device itself.

Every object type defines the required and optional properties an object instance of this type may have. Properties are of a defined BACnet Data Type, each composed of a defined set of primitive data types. They represent a property or feature of the object. Properties are used to represent:



<Object Type>
Identification
Values used or provided by the process
Parameters of the process
State of object or the process
Event states and event notification parameters
Relations to other objects
Etc.

Figure 4–3, Object Type Template

Properties can be read and written by client role processes using various BACnet application services. Commanding an object is done through writing values to properties. Command arbitration is done by prioritizing commands, where the highest priority command wins. Objects are the source of notifications sent to client role processes. Notifications report about changes of values (COV) of properties or events and alarms detected.

4.4 BACnet Application Services

BACnet defines a set of application services for remote communication between processes and objects of networked devices. A service basically consists of a request and optionally a response message. Both the service request and the service response have defined parameters, composed of BACnet Data Types.
There are services defined for object access, change of values (COV) reporting, event reporting, and remote device management. Virtual terminal session services allow a device to provide a VT100 terminal style user interface on remote devices. The set of defined services can be classified in BACnet Access Services and BACnet Notification Services.

4.4.1 BACnet Access Services

BACnet Access Services are used to access properties of objects and commanding of objects. They are initiated by client role processes and addressed to objects for execution and response.

Since objects are used to expose the functionality of the server role process, the server role process effectively performs the execution of the service request and may return a response.

If the access is performed internal in a device (i.e. client and server role process in the same device), no service requests and responses are visible on the BACnet network. Such device internal services are not in the scope of BACnet definitions.

4.4.2 BACnet Notification Services

BACnet Notification Services are used to asynchronously notify subscribed client role processes on changes of value or changes of event state of objects. In order to receive notifications, the client role process subscribes to objects of interest using BACnet Access Services. Notification requests are initiated by objects and addressed to subscribed client role processes. There are two sub-classes of BACnet Notification Services:

- Change-Of-Value (COV) notifications when values of properties of an object change.
- Event notifications when the event state of an object changes.

Since objects are used to expose the functionality of the server role process, the server role process is effectively initiating notification service requests. The client role process is executing the service and typically provides a simple response indicating reception of the request.

If notification is performed internal in a device (i.e. client and server role process in the same device), no service requests and responses are visible on the BACnet network. Such device internal services are not in the scope of BACnet definitions.

5. PACS Functional Decomposition

The PACS as defined in this document is segmented into functional units to identify the fundamental logical PACS processes, the associated interfaces, and their interrelationship. Note that this is not a physical decomposition, but a functional one. This serves to identify the interfaces to these functional units. The data and service models of these interfaces can then be defined using BACnet objects.

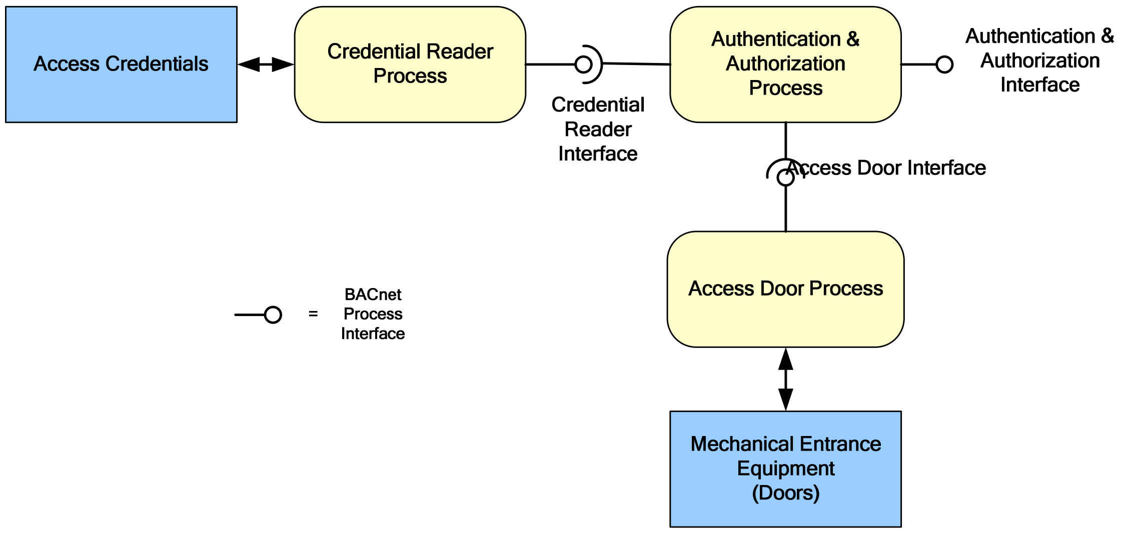

Access Credentials
Credential Reader Process
Credential Reader Interface
Authentication & Authorization Process
Authentication & Authorization Interface
Access Door Interface
Access Door Process
Mechanical Entrance Equipment (Doors)
= BACnet Process Interface

Figure 5–1, PACS Functional Decomposition

The PACS includes the following logical processes, which are described in detail in the following sections:

- Credential Reader Process
- Access Door Process
- Authentication & Authorization Process

Note that, depending in part on the technology or intelligence being employed in the Credentials and Credential Reader as well as the Physical Door and its Electro-Mechanical Control and Monitoring Equipment, it could be argued that this equipment is or is not part of the PACS. Industry practitioners would consider this equipment as part of the PACS. Such determination has no impact to the BACnet interfaces, since hidden to those by the respective processes.

5.1 Credential Reader Process

The Credential Reader Process performs reading and validation of access credentials, control of indicators and / or keys at the reader front plate, may access other data of a credential etc. The exact functionality and implementation of this process is not part of the BACnet definitions; it is a local matter of the device or subsystem which implements it. Any physical structure or deployment of this functionality behind the BACnet Process Interface provided by this process is a local matter, hidden to the BACnet network. This includes non-BACnet reader device interfaces, such as Wiegand type interfaces, or reader to card communication.

The Credential Reader Process represents its functionality at its BACnet process interface, the **Credential Reader Interface**. This interface is subject of the BACnet standardization, and modeled in detail later in this document. Within a PACS, it is used by the Authentication & Authorization Process for access decisions. It may also be used by other processes performing other application functions, such as Time & Attendance, Guard Tour Monitoring, Muster Station, etc.

This process does not take any client role. Its operation is independent of other PACS processes.

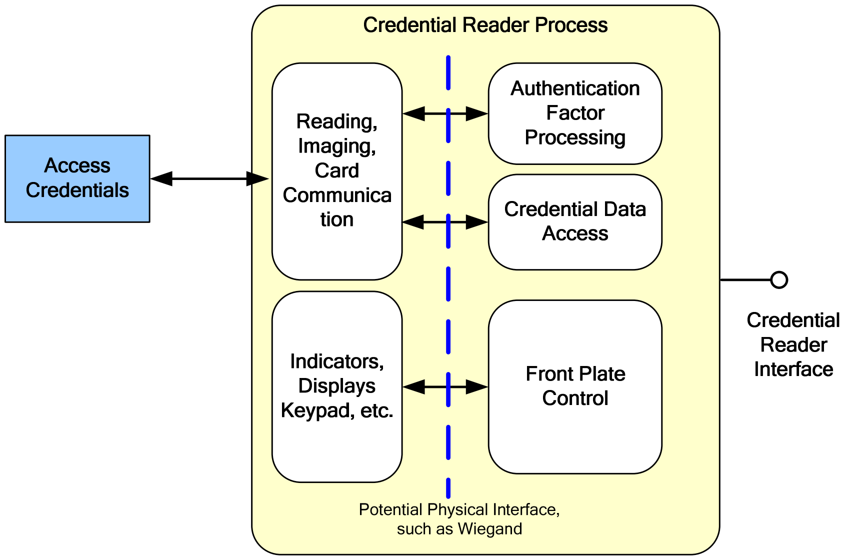

Credential Reader Process
Access Credentials
Reading, Imaging, Card Communica tion
Authentication Factor Processing
Credential Data Access
Indicators, Displays Keypad, etc.
Front Plate Control
Credential Reader Interface
Potential Physical Interface, such as Wiegand

Figure 5–2, Credential Reader Process

5.1.1 Reading, Imaging, Card Communication

This part of the Credential Reader Process performs communication with access credentials or sensing of authentication factors. Depending on the technology, various types of one- or two-way communication may take place:

- Magnetic-stripe card reading
- Barcode reading
- Contact or contact-less smartcard communication
- Finger-print scanning
- Hand-shape scanning
- Iris scanning
- Imaging (e.g. for face recognition)
- Etc.

This functionality may be performed in a dedicated physical device, such as a card reader device, fingerprint scanner, camera, etc. The connection of such dedicated devices to the controller device, which contains the remaining parts of the Credential Reader Process and provides the Credential Reader Interface, is outside the scope of BACnet standardization. Existing and upcoming industry standards are typically used for this connection (e.g. Wiegand).

The data generated by this functionality is considered a raw authentication factor. Some implementations may make this raw authentication factor data available at the Credential Reader Interface, for e.g. diagnostics purposes.

5.1.2 Indicators, Keypad, etc.:

This part of the Credential Reader Process manages front plate elements for interaction with the access user. It includes:

- Signaling LED's
- Display (monochrome, color, text, graphics, etc.)
- Keypad
- Etc.

This functionality may also be performed in a dedicated device, such as a card reader device. The connection of such devices to the controller device, which contains the remaining parts of the Credential Reader Process and provides the Credential Reader Interface, is outside the scope of BACnet standardization. Existing and upcoming industry standards are typically used for this connection (e.g. Wiegand).

5.1.3 Authentication Factor Processing

The raw authentication factor data read from a credential, or e.g. produced by biometrics scanning, is processed and validated before made available at the Credential Reader Interface.

Reading data from a credential or imaging biometrics may include transferring such data from a non-BACnet reader device into a controller through any non-BACnet communication. Such communication is considered a local matter of the Credential Reader Process.

Processing and validation may include parity checks and strip-off of parity bits, structuring of data, authentication by certificates, verification of authenticity by hashing algorithms or a PIN, validation of expiry date information read with the factor, minutiae generation of biometrics etc. Failed processing may be indicated at the Credential Reader Interface. Resulting authentication factor data is presented at the Credential Reader Interface for any client role processes using that interface.

Any information used to validate an Authentication Factor itself is not required to be presented at the Credential Reader Interface, since the PACS does not use it for Authentication & Authorization. Such information is used by the Credential Reader Process itself for preprocessing and validation of Authentication Factors.

Examples:

- The credential reader process is able to read Wiegand numbers from a magnetic stripe card, to check parity bits and strip them off before providing the number at the Credential Reader Interface.
- If a PIV card is read, the entire Card Holder Unique Identifier (CHUID) is typically not used in the PACS, but either the FASC-N part or GUID is used in the PACS. The credential reader process is responsible to read the PIV card, to validate its content against expiration date, and may use the signature to further validate the CHUID. Then, it extracts the FASC-N from the CHUID and provides it as an authentication factor at the Credential Reader Interface.
- The raw scan of a fingerprint is analyzed, and minutiae data of that scan is generated. The minutiae data is provided at the Credential Reader Interface as an authentication factor.

A Credential Reader Process may be capable of providing different subsets or formats of authentication factors, or different authentication factors. This capability is represented at the Credential Reader Interface through according BACnet objects.

5.1.4 Credential Data Access

The Credential Reader Process may make credential data accessible. This is data, available on some credentials, may be for example smart card blocks or credits. Although this is out of scope of PACS, the Credential Reader Process may, at its Credential Reader Interface, provide models for such content and making it accessible for reading and writing it by any client role process.

5.1.5 Front Plate Control

A Credential Reader Process controls front plate elements for input from and indication to the access user. Such elements may be:

- Keypads
- LED indicators
- LCD displays
- Etc.

Access to such elements by client role processes may be provided at the Credential Reader Interface.

5.1.6 Deployment to Physical Structure

BACnet basically does not define how functionality is deployed to physical devices. Any physical structure may be possible behind a BACnet process interface. Sample deployments are:

- The entire process is deployed into an intelligent reader device. This would result in a reader device which has a BACnet network connection.

- In case the functionality is distributed over a reader device and a controller, the Credential Reader Interface is provided by the controller, accessible through its BACnet network connection. Communication between the reader device and the controller is a local matter of the process, not defined by BACnet. Such communication may be based on any access control industry common communication protocol.

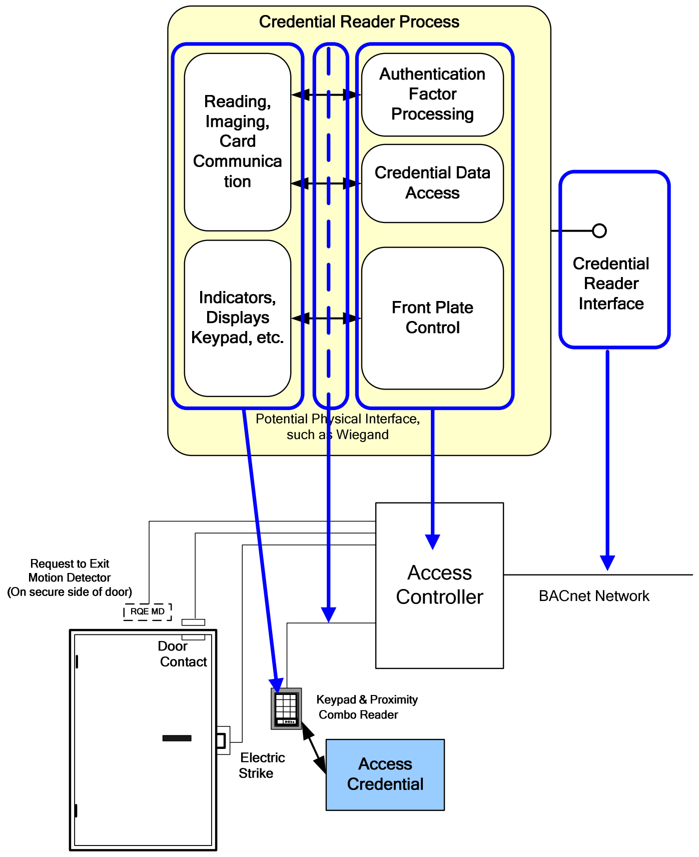

Credential Reader Process
Reading, Imaging, Card Communica tion
Indicators, Displays Keypad, etc.
Authentication Factor Processing
Credential Data Access
Front Plate Control
Credential Reader Interface
Potential Physical Interface, such as Wiegand
Access Controller
BACnet Network
Request to Exit Motion Detector (On secure side of door)
RQE MD
Door Contact
Electric Strike
Keypad & Proximity Combo Reader
Access Credential

Figure 5–3, Example Deployment of the Credential Reader Process

5.2 Access Door Process

The Access Door Process performs the control and monitoring of the mechanical entrance equipment, the access door. The exact functionality and implementation of this process is not part of the BACnet definitions; it is a local matter of the device or subsystem which implements it. Any physical structure or deployment of this functionality behind the BACnet process interface provided by this process is a local matter, hidden to the BACnet network. The means how this process communicates with the mechanical door equipment such as drives, locks etc. is a local matter, outside the scope of BACnet.

This process represents its functionality at its BACnet process interface, the **Access Door Interface**. This interface is subject of the BACnet standardization, and modeled later in this document. Within a PACS, it is used by the Authentication & Authorization Process to control entrance based on access decisions. It may be used by other processes performing other applications, such as Schedulers, Fire Systems, Intrusion Detection Systems, etc.

This process does not take any client role. Its operation is independent of other PACS processes.

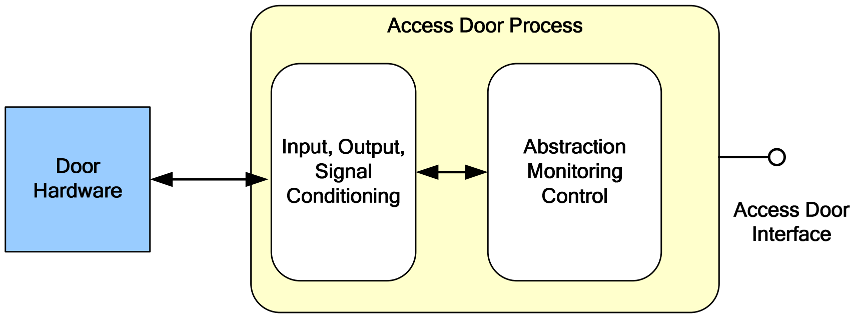

Access Door Process
Door Hardware
Input, Output, Signal Conditioning
Abstraction Monitoring Control
Access Door Interface

Figure 5–4, Access Door Process

5.2.1 Input, Output, Signal Conditioning

Sensors at the door provide input to the process. Such input may be conditioned before further processing. This may include adaptation curves, debouncing etc. Inputs may also be made accessible individually at the Access Door Interface. Sensors may be:

- Lock contacts
- Dead-bolt contacts
- Door closed contacts
- Emergency stop buttons
- Operation continuation buttons or sensors (e.g. inside revolving doors)
- Movement detectors
- Request-to-exit buttons
- Manual open buttons
- Handle switches
- Etc.

Actuators at the door are used to enable control of the door by the process. Such actuators may also be made accessible individually at the Access Door Interface. Actuators may be:

- Dead-bolt coils
- Electric strikes
- Drives (e.g. sliding doors)
- Etc.

5.2.2 Abstraction, Monitoring, Control

The individual elements of a door are abstracted into a uniform model of a door, to simplify and standardize control and monitoring of various door types by external client processes. This includes monitoring of the door state and status based on sensor values, as well as control of individual actuators. Such control may include timing, speed, interlocks etc.

The abstracted door functionality is made accessible at the Access Door Interface.

5.2.3 Deployment to Physical Structure

BACnet basically does not define how functionality is deployed to physical devices. Any physical structure may be possible behind a BACnet process interface. Sample deployments are:

- The entire process is deployed into an access controller. The door sensors and actors are directly connected to the access controller. The Access Door Interface is accessible through the access controller's BACnet connection.

- The door has its own door controller device, unaware of whether it is used by a PACS. The door sensors and actors are connected to that door controller. The door controller makes the Access Door Interface accessible through its BACnet connection. An access controller device would use the BACnet connection to interact with the door.

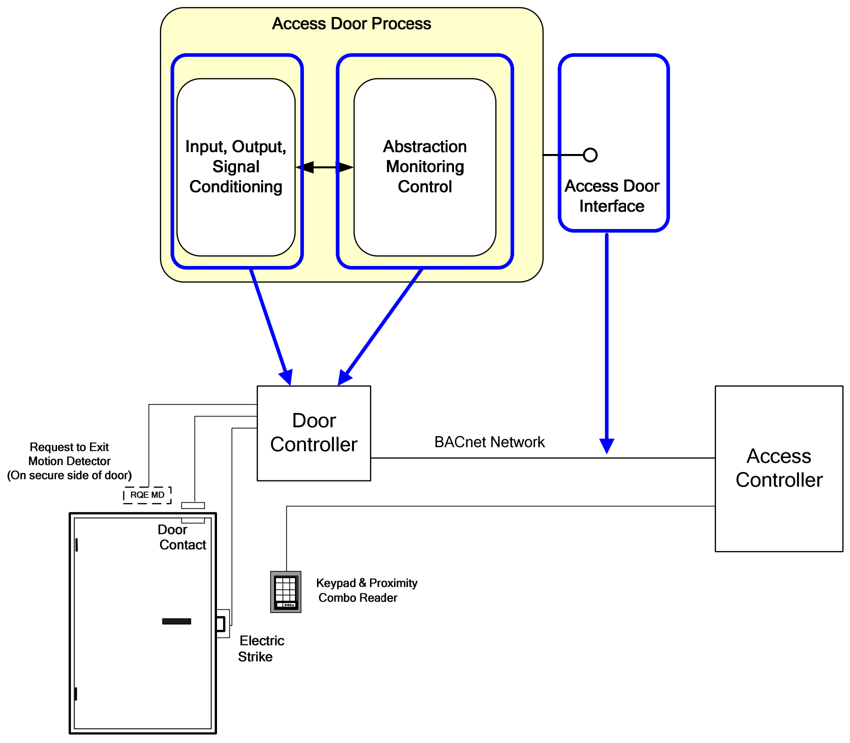

Access Door Process
Input, Output, Signal Conditioning
Abstraction Monitoring Control
Access Door Interface
Door Controller
BACnet Network
Access Controller
Request to Exit Motion Detector (On secure side of door)
RQE MD
Door Contact
Keypad & Proximity Combo Reader
Electric Strike

Figure 5–5, Example Deployment of the Access Door Process

5.3 Authentication & Authorization Process

The Authentication & Authorization process performs the effective access control functionality. It uses information read from access credentials (i.e. authentication factors) as provided at the Credential Reader Interface to authenticate the credential, determines access authorization and controls the door through the Access Door Interface. Authentication and authorization functionality may be summarized as answering the following four “W” questions:

Where is access requested?
The geographical organization of access control structures the geography of a site. This includes secured zones and the points of access to such zones, where authentication and authorization for access takes place.

Who or what requests access?
Authentication of the users which request access to secured zones is required. This includes representation of persons, assets, groups thereof, as well as the credentials they use for authentication.

Why should access be granted?
Access rights define the rights which a user has at an access point or for an access zone. Access authorization may also depend on the credential presented, or other internal or external conditions.

When can access be granted?
Access rights may depend on the time of day, day of week or date.

Notification and logging of access transactions and alarms is an essential function of this process. The exact functionality and implementation of this process is not part of the BACnet definitions; it is a local matter of the device or subsystem which implements it. Any physical structure or deployment as well as partial support of this functionality behind the BACnet Process Interface provided by this process is a local matter, hidden to the BACnet network.

This process represents its functionality at its BACnet process interface, the **Authentication & Authorization Interface**. This interface is subject of the BACnet standardization, and modeled later in this document. Within a PACS, it is typically used by other Authentication & Authorization Processes running in other devices, for data replication and synchronization, or to complement it to full access functionality. Another typical usage of this interface is access user and credential enrollment, performed by some external system such as a Credential Management System (CMS) or Identity Management System (IDMS). An access control management system uses this interface for e.g. rights assignment, credential status setting, etc.

This process takes a client role at Credential Reader Interfaces for authentication factor input, as well as a client role at Access Door Interfaces for control of doors, if the corresponding processes are deployed to other physical devices. If they run internally in the same device, interaction with these processes is not subject to BACnet definitions, and considered a local matter.

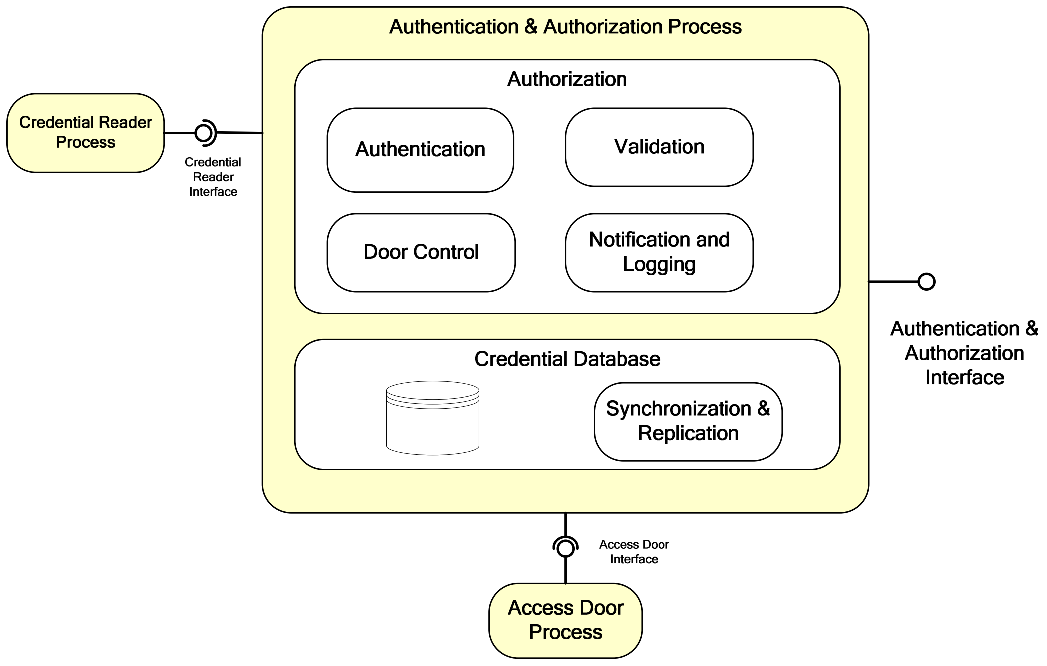

Authentication & Authorization Process
Authorization
Authentication
Validation
Door Control
Notification and Logging
Credential Database
Synchronization & Replication
Credential Reader Process
Credential Reader Interface
Authentication & Authorization Interface
Access Door Interface
Access Door Process

Figure 5–6, Authentication & Authorization Process

5.3.1 Authorization

Authorization is the basic function of a PACS. It includes authentication of credentials, validation of access rights and other conditions. This is a series of checks required to pass successfully before access is granted. Granting access results in controlling a door.

5.3.2 Authentication

Authentication factors are provided at the Credential Reader Interface at the time a user requests access. These factors are compared with information in the credential database, in order to find a matching credential representation in the database. Matching algorithms are a local matter of the process. This may include accessing external systems, typically through other protocols than BACnet. Authentication is one of the checks performed for authorization. Without finding any matching credential representation, access is denied.

The results of authentication may be reported to client role processes through the Authentication & Authorization Interface.

5.3.3 Validation

Validation represents a series of checks to pass successfully before access is granted. The sequence of checks is not mandated. Example checks are:

- Validation of credentials based on its status, expiry time etc.
- Evaluation of external conditions
- Sufficient access rights assigned to the credential for the access point
- External verification by an operator or external system
- Etc.

Some validation checks are related to the geographical location where the respective credential has been read. The geographical location is given through which credential reader has read an authentication factor.

The validation checks may be taken in any order, before or after authentication, or run in parallel. A failed check results in finally denying access, while only if there is a sufficient number of successful validation checks, access is granted.

The device's local Credential Database holds data which is used for some of the validation checks, although in some implementations data is used which is located in external devices, typically access servers. Degraded validation may take place if such external devices are not accessible when validation checks have to be preformed.

Some checks may already be performed by the Credential Reader process, but then the Authentication & Authorization Process does not become active since no valid authentication factor is received from the Credential Reader Interface.

The results of validation may be reported to client role processes through the Authentication & Authorization Interface.

5.3.4 Door Control

Doors are controlled based on the results of authentication and validation. Typically, a door is opened after successful authentication and validation. Access Door Interfaces are used to control respective doors. Which door to control is determined based on which credential reader has read an authentication factor.

5.3.5 Notification and Logging

Notification and logging of access transactions and alarms is an essential function of the Authentication & Authorization Process. Notifications may be directed to other access control devices as well as management systems. Local logging of such notifications is typically supported by an implementation, to allow autonomous operation of the device the process is deployed to.

5.3.6 Credential Database

To perform authentication and validation checks, information is required on credentials and access rights. Such data is typically available in the device the process runs, and stored in some form of database. This is not necessarily a database as known in the IT realm, such as a relational database, but optimized for constrained resources and fast access to essential data.

It typically holds data about:

- Credentials, their authentication factors, and relations with access users
- Access rights and relations with credentials and geographical structure
- Relations among access rights and credentials
- Access user representations and relations

Depending on implementation and supported functionality, more or less of this data may be present.

The storage format of such data within a controller is not necessarily the same as the format visible at the Authentication & Authorization Interface. The Authentication & Authorization Interface enables provisioning of such data into the access control device using BACnet.

5.3.7 Replication & Synchronization

Data in the credential database may be required to be synchronized among different devices, or replicated in multiple devices. This is required since for example state and parameters of a specific credential must be the same in all devices which know that credential. Any state change must be synchronized among all credential databases of these devices.

The amount of data to be replicated and synchronized depends on the nature of data and its usage in a device. For example, it is not required to have access rights present which are valid for some door which is not controlled by the device.

There may be synchronization and replication between access controllers, or via a central access control server, which holds the master data for all controllers of the PACS, and is responsible for replication and synchronization.

The Authentication & Authorization Interface supports this functionality through an appropriate BACnet data model and BACnet application service support. In particular, objects in multiple devices that represent the same entity (e.g. credential, user) contain the same Global Identifier, which allows identifying all these objects which reside in multiple devices.

5.3.8 Deployment to Physical Structure

BACnet basically does not define how functionality is deployed to physical devices. Any physical structure may be possible behind a BACnet process interface. Sample deployments are:

- The entire process and its entire database are deployed into an access controller. Logging is performed by the controller itself, such that fully autonomous operation is possible. Any external functionality such as credential enrollment accesses this controller directly. The Authentication & Authorization Interface is accessible through the access controller’s BACnet connection.

- The process and the database are partially deployed onto access controllers and a central access server. Only the minimum required data is loaded into the controller, while the overall master data is located in a central server. Logging may be centralized in this server, while the controllers only have limited logging capacity to span communication interruptions. The access controller’s Authentication & Authorization Interface is accessible through the controller’s BACnet connection. The central server not necessarily provides such an interface; it may contain client role processes only.

6. PACS Data Model Overview

The overview of the PACS data model reflects the three BACnet process interfaces as identified in the functional decomposition of the PACS.

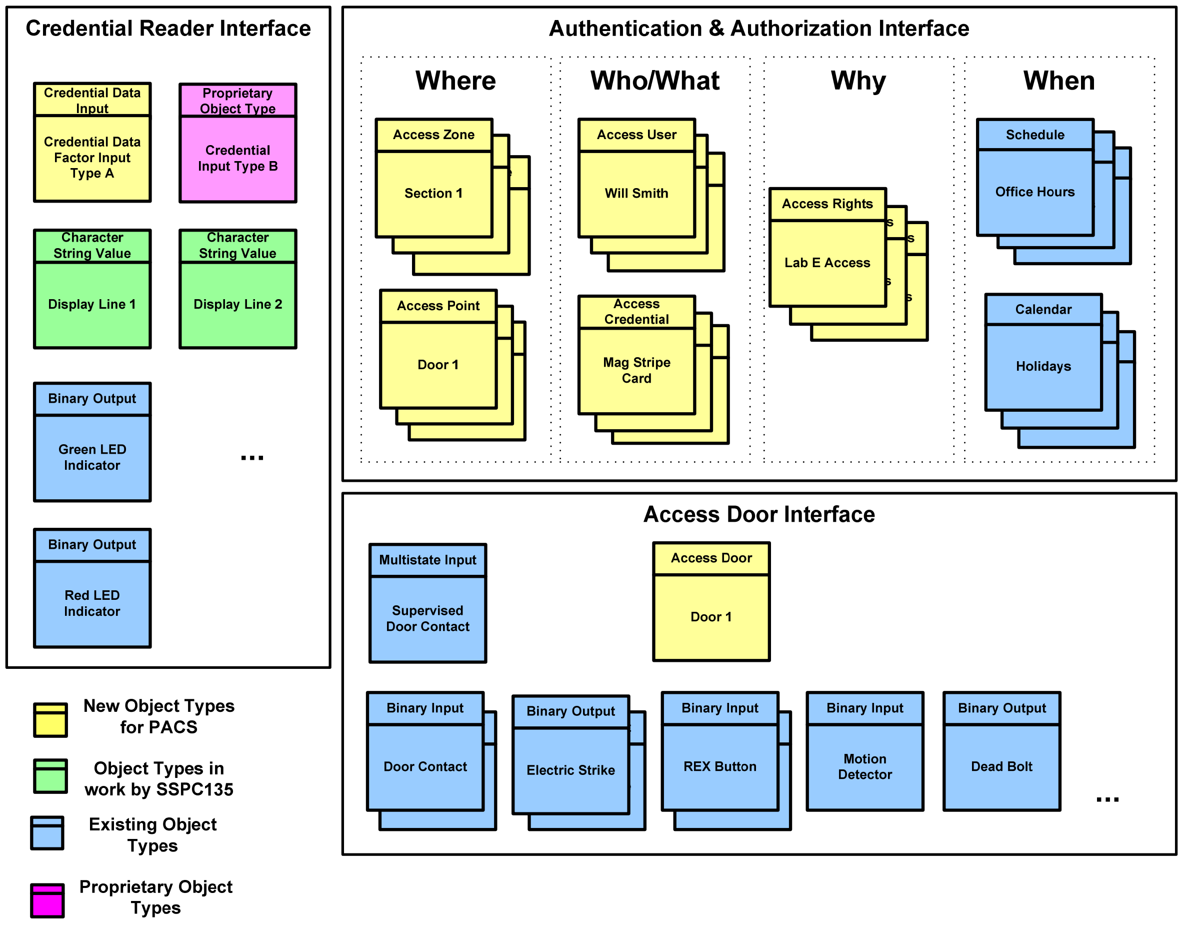


Figure 6–1, Data Model Overview

An overview of the relationship among these objects in a PACS is shown in the figure following. Note that any client role process may have relationships with the objects shown.

The arrows between two object types show the association or relationship between the two object types while the direction of the arrow indicates the direction of this relationship. When an arrow points from one object type to another it means that an instance of the first object type will hold a reference to an instance of the second object type. In the example given, the Access Rights object will hold references to zero or more Access Point or Access Zone objects.

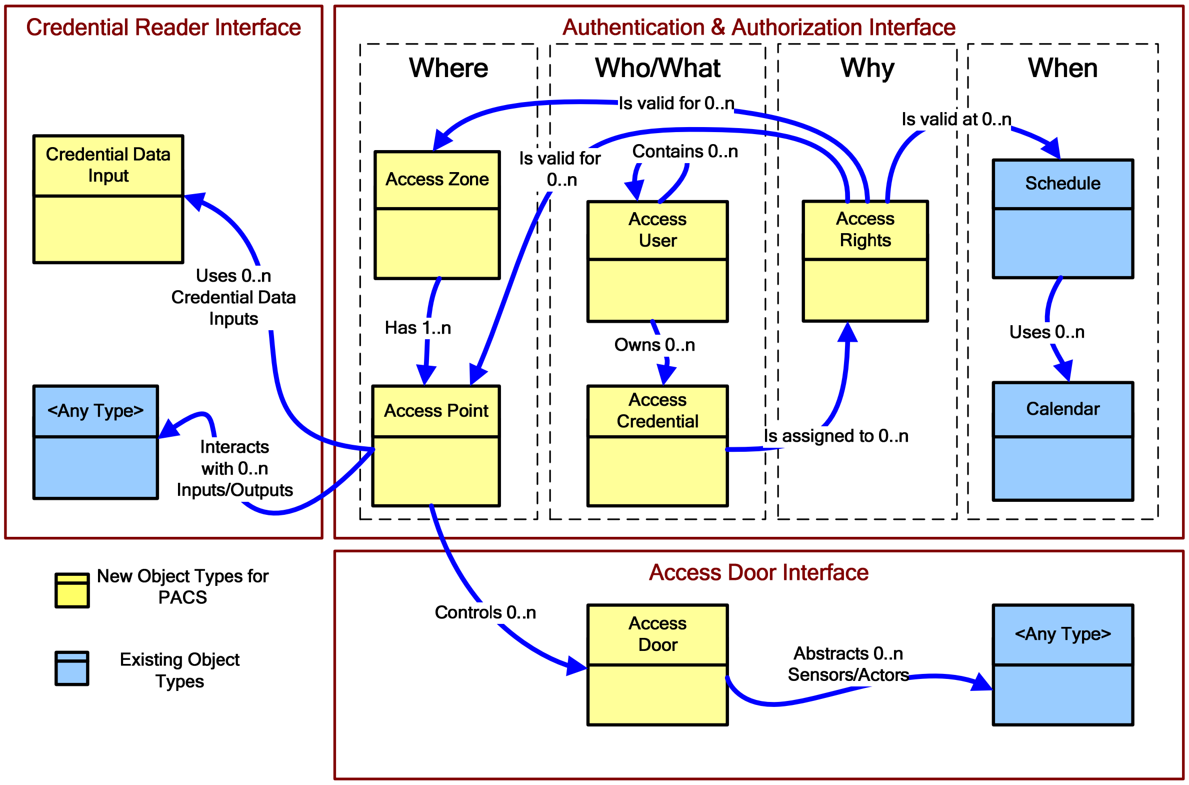


Figure 6–2, Relationship Overview

A more detailed overview of the relations is given by the following figure. The details are discussed in the following sections of this document.

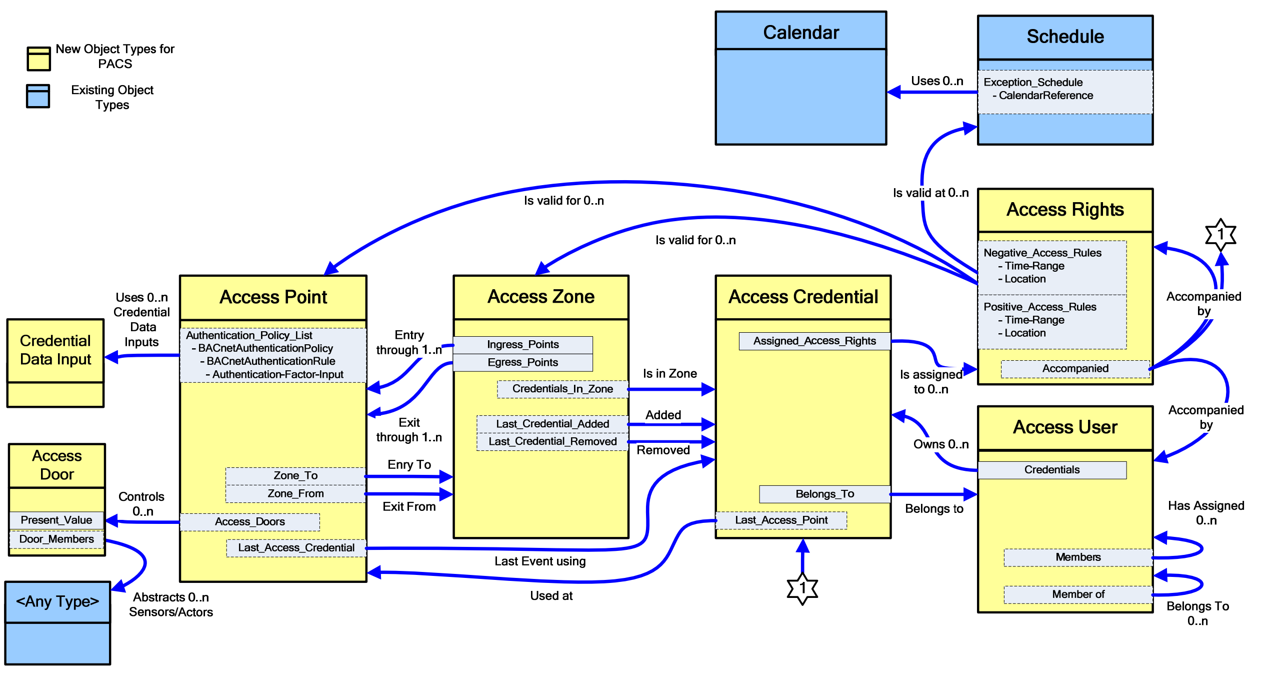


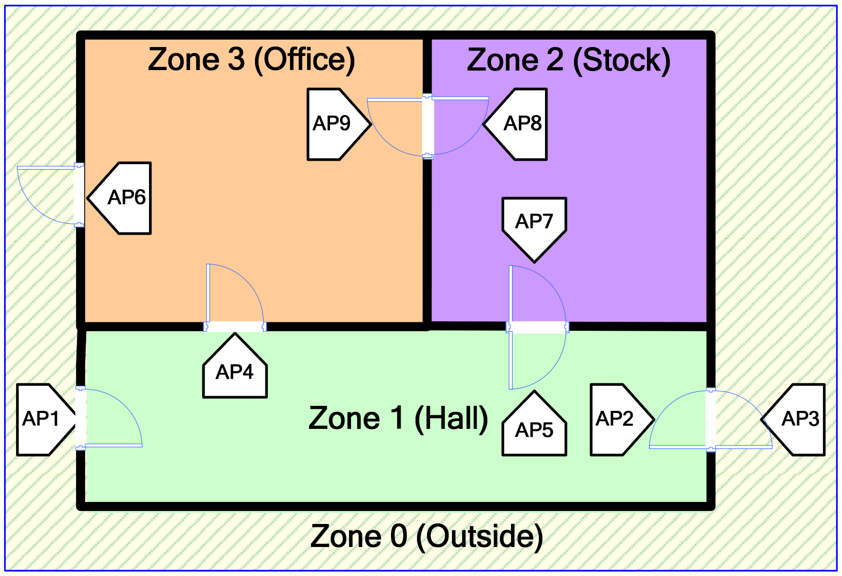
Figure 6–3, Relationship Details

7. Authentication & Authorization Interface

The Authentication & Authorization Interface defines a data model that represents the structure and data required to perform authentication and authorization. The data model also enables to control the functionality of the authorization and authentication process. Access transaction and alarm notifications are issued by the objects of this interface.

7.1 Geographical Organization

The point of authentication and access to secured zones is represented by objects of type **Access Point**. The secured zones may be represented by **Access Zone** objects. Note that the mechanical entrance equipment is subordinate to this concept, and provided by the Access Door Interface.



AP1 (E.g. Parking lot employee entrance):
- Entry to Zone 1 (Hall).
- Exit from Zone 0 (Outside).

AP2:
- Entry to Zone 0 (Outside).
- Exit from Zone 1 (Hall).

AP3 (E.g. Main entrance to Hall):
- Entry to Zone 1 (Hall).
- Exit from Zone 0 (Outside).

AP4:
- Entry to Zone 3 (Office).
- Exit from Zone 1 (Hall).

AP5:
- Entry to Zone 2 (Stock).
- Exit from Zone 1 (Hall)

AP6 (E.g. Fire Exit):
- Entry to Zone 0 (Outside).
- Exit from Zone 3.

AP7:
- Entry to Zone 1 (Hall).
- Exit from Zone 2 (Stock).

AP8:
- Entry to Zone 3 (Office).
- Exit from Zone 2 (Stock).

AP9:
- Entry to Zone 2 (Stock).
- Exit from Zone 3 (Office).

Figure 7–1, Example Geographical Organization

7.1.1 Access Point Object Type

The Access Point object represents the authentication & authorization process at a specific geographic access controlled point (i.e., door, gate, turnstile, etc). Access through this point is directional in that it represents access in one direction only. A door in which access is controlled in both directions leads to two separate Access Point objects.

In the most simple access control systems the Access Point controls access through a secured door. In more sophisticated systems the Access Point controls entry into a zone (Entry Access Point) or exit from a zone (Exit Access Point). In the case of adjacent zones, the Access Point is an Entry Access Point for one zone and an Exit Access Point for the neighbor zone at the same time.

Access Point	
Identification:	Name, ID, Type, Description, Profile
General Health:	Status Flags, Event state, Reliability, Out of service
Authentication Status:	Status of the authentication process
Authentication Policies:	Select one of a predefined set of policies
Authorization Mode:	Defines mode of Authorization operation
Lockout:	Failed Access Attempts and Lockout
Threat Level:	Minimum Requirement to Credential’s Threat Authority
Occupancy:	Enforcement of occupancy limits, Count adjustment
Accompaniment Time:	Defines the timing of accompaniment authentication
Access Event Reporting:	Alarm and Transaction Events
COV Reporting:	Access event changes for remote monitoring
Access Doors:	The Access Doors controlled and supervised
Muster Station Support:	Flag to indicate a Muster Point
Access Door Commanding:	Doors commanded with what command priority
Access Zone Relationship:	Entry and Exit Access Zone

Figure 7–2, Access Point Object Type

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 1.

7.1.1.1 Identification

These are the standard properties of BACnet objects for object identification, type identification, description, etc.

7.1.1.2 General Health and Out of Service

Properties are defined which indicate the general health of the object. Reliability indication and out of service is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which provide more details.

- The IN_ALARM flag is 1 if the object has an event state other than NORMAL.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDDEN flag is 1 if the object is overridden by some mechanism local to the BACnet device.
- The OUT_OF_SERVICE flag is 1 if the property Out_Of_Service is TRUE.

The required property ***Event_State***, of type *BACnetEventState*, indicates the event state associated with the object. Since access event reporting is considered state-less, this property has a value of NORMAL, except if Reliability is something else than NO_FAULT_DETECTED. In this case it has a value of FAULT.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of the authentication & authorization process for the access point this object represents. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
PROCESS_ERROR	The process is unreliable.
CONFIGURATION_ERROR	The configuration of the Access Point object has an error preventing reliable processing.
COMMUNICATION_FAILURE	Proper operation of the object is dependant on communication with a remote sensor or device and communication with the remote sensor or device has been lost.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

If the Reliability property has a value other than NO_FAULT_DETECTED then no authentication or authorization is performed. No access events are generated in this case.

The required property ***Out_Of_Service***, of type *BOOLEAN*, indicates whether the authentication & authorization process at the access controlled point this object represents is out of service or not. If it is TRUE neither authentication nor authorization takes place at the access controlled point. When this property changes from FALSE to TRUE then the Access_Event property is set to OUT_OF_SERVICE. When this property changes from TRUE to FALSE the Access_Event property is set to OUT_OF_SERVICE_RELINQUISHED.

7.1.1.3 Authentication_Status

The required property ***Authentication_Status***, of type *BACnetAuthenticationStatus*, indicates the current status of the authentication process. This is an enumeration with the following status values:

NOT_READY	The authentication process is not ready to perform a new authentication. This indicates a temporary condition due to processing of the current authentication factor, initialization during startup or other internal processing.
READY	The authentication process is ready to start a new authentication.
DISABLED	The authentication process has been disabled. The property shall take on this status when the Out_Of_Service property is TRUE.
WAITING_FOR_AUTHENTICATION_FACTOR	The authentication process is waiting for an additional authentication factor for a multi-factor authentication.
WAITING_FOR_ACCOMPANIMENT	The authentication process is waiting for the authentication of the accompanying credential.
WAITING_FOR_VERIFICATION	The authentication process is waiting for the verification of the credential by an external process or human operator.
IN_PROGRESS	The authentication process is in progress.

An Access Point object is not required to implement all these states. Proprietary extensions are not foreseen for this enumeration.

7.1.1.4 Authentication Policies

Authentication may utilize multiple authentication factors such as card and PIN. An Authentication Policy determines which authentication factors are used and how they are used to reach an authentication decision. Each Access Point may have its own Authentication Policy and may allow selecting the active policy from a set of policies.
An Access Point uses Credential Data Input objects to get authentication factors. The used Credential Data Input objects are exposed through the defined Authentication Policies if present.

If the Access Point object supports to select the active Authentication Policy, it exposes the number of supported Authentication Policies, and allows selecting the active Authentication Policy from these. The explicit definition of each supported Authentication Policy may be exposed optionally, which includes individual maximum times to present the authentication factors. Optionally, the names of the supported Authentication Policies may be exposed. The figure following provides an overview and example Authentication Policy support.

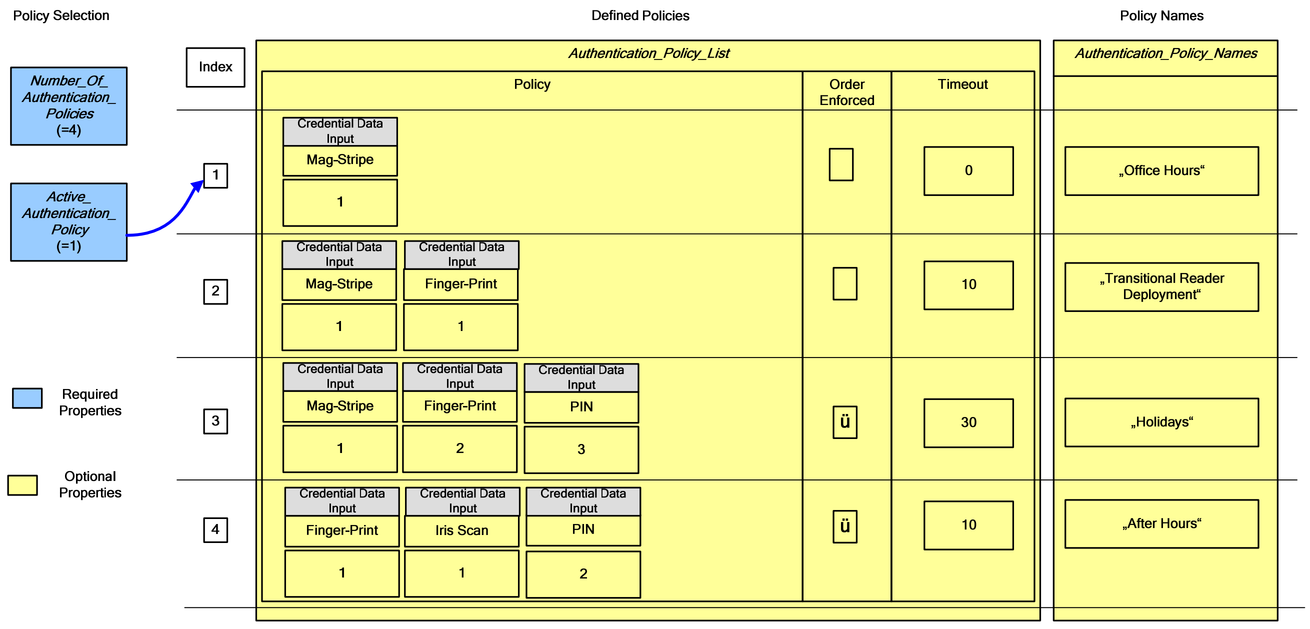


Figure 7–3, Authentication Policy Overview and Example

The required property ***Active_Authentication_Policy***, of type *Unsigned*, selects the currently active Authentication Policy. This is a value in the range 0...N, where N is defined by the total number of supported Authentication Policies, as specified in the Number_Of_Authentication_Policies property. If the Authentication Policies are explicitly specified in the property Authentication_Policy_List, or names are specified in the property Authentication_Policy_Names, the value of the Active_Authentication_Policy property also specifies the index into these arrays, determining the currently active explicit Authentication Policy specification and its name.

A value of zero in this property disables the Access Point by setting it to unreliable with CONFIGURATION_ERROR in the Reliability property. There is no authentication policy in effect, and through this, no authentication and no authorization. In case the active Authentication Policy becomes invalid through modification, this property takes a value of zero, disabling the Access Point.

The required property ***Number_Of_Authentication_Policies***, of type *Unsigned*, specifies the total number of supported Authentication Policies. The value of this property is always greater than zero.

The optional property ***Authentication_Policy_List***, of type *BACnetArray of BACnetAuthenticationPolicy*, exposes the explicit configuration of the defined authentication policies. It is the vendor's choice whether this property may be written for configuration, or is present for visibility purpose only. If this property is not present, then the configured authentication policies are a local matter, not exposed through BACnet.

An Authentication Policy is basically a set of required or optional authentication factors to be presented for successful authentication. The definition of what authentication factor type is to be used is made by a reference to a Credential Data Input object, where the respective authentication factor is to be read from. This implicitly defines the type of Authentication Factor. Since a Credential Data Input object may support different Authentication Factor types, it is essential that the set of types supported by that Credential Data Input object are functionally equivalent and provide equivalent security.

The following elements define an Authentication Policy:

- **Required authentication factor types** of a policy. This defines what authentication factor types are required to be presented at the Access Point for successful authentication.
- **Choices of optional authentication factor types** of a policy. One of the authentication factor types of a choice is required to be presented at the Access Point for successful authentication.
- **Sequence enforcement flag** of a policy. This flag indicates whether the sequence of authentication factors as defined by the index value within the policy is to be kept when presenting the factors (TRUE), or the factors may be presented in any order (FALSE.
- **Policy timeout** of a policy. This timeout may specify the maximum time available to present all required authentication factor types.

The structured data type of an Authentication Policy is of type BACnetAuthenticationPolicy and contains:

Policy	This field is a list of elements that specifies the Credential Data Inputs which are used for this authentication policy. Each element of the list has the following fields:	
	Credential-Data-Input	This field, of type BACnetDeviceObjectReference, contains a reference to an object of type Credential Data Input where the authentication factor value is read from the physical device.
	Index	This field, of type Unsigned, indicates the order in which the authentication factors will be evaluated. The value starts with the value 1 and continues in increasing sequence. If two or more entries of the Policy list have the same index value this indicates that there is a choice between any of the authentication factors supported by the Credential Data Input objects referenced by these entries. In this case the user may present any one of these authentication factors.
Order-Enforced	If TRUE, then the ordering sequence to present authentication factors, as specified by the Index fields of the Policy list, is enforced. If FALSE, then the order is not enforced.	
Timeout	This field, of type Unsigned, specifies the maximum time in seconds for which all authentication factors, as defined by this policy, must be presented. A value of zero indicates an unlimited time to present all authentication factors. If not all authentication factors are presented in the allotted time, then a timeout occurs and authentication fails.	

An Authentication_Policy_List array element is considered invalid if the Policy field is empty or if it is not well formed., e.g. the Index is not in increasing order. If an invalid entry is present in the property, however it got there, then the Reliability property takes on the value CONFIGURATION_ERROR.

If this property is not present, then the authentication policies are a local matter.

Authentication factors are read at a Credential Reader front plate, preprocessed and presented by the Credential Reader Process at the Credential Reader Interface as a value of type BACnetAuthenticationFactor. The Authentication & Authorization Process reads these BACnetAuthenticationFactor values and determines if the active Authentication Policy is met.

The Authentication_Policy_List property, if present, contains at least one valid Authentication Policy. The i^{th} element of this array corresponds to the i^{th} element of the Authentication_Policy_Names array. The size of the Authentication_Policy_List array must equal the value of the property Number_Of_Authentication_Policies.

Attempts to write malformed policies are denied by the Access Point object. Since the elements in an Authentication Policy element of the Authentication_Policy_List array is embedded in a structured data type, BACnet services allow reading or writing entire policies only. So a verification of the correct form of a policy when written is always possible.

If the size of the Authentication_Policy_List array is increased without initial values being provided, then the new array elements for which no initial value is provided are initialized with an empty Policy list, Order-Enforced is FALSE, and Timeout has the value zero.

The optional property ***Authentication_Policy_Names***, of type *BACnetArray of CharacterString*, exposes the names of the Authentication Policies. This property may be present independent of the presence of the Authentication_Policy_List property.

The i^{th} element of this array corresponds to the i^{th} element of the Authentication_Policy_List array, if present. The size of the Authentication_Policy_Names array must equal the value of the property Number_Of_Authentication_Policies.

7.1.1.5 Reading Authentication Factors

The Authentication Factors to be read are determined by the current authentication policy in effect. If the Authentication_Policy_List property is present the authentication policy is explicitly defined and specifies which Credential Data Input objects the authentication factors are read from. When any authentication factor is read the Access_Event property is set to AUTHENTICATION_FACTOR_READ.

If the authentication factor read does not match any known authentication factor or the authentication factor read has an error (i.e., has a format type of ERROR) then authentication fails and access is denied. In the case where the authentication factor is unknown the Access Event property is set to DENIED_UNKNOWN_CREDENTIAL. In the case where the authentication factor has an error the Access_Event property is set to DENIED_AUTHENTICATION_FACTOR_ERROR.

It may be possible with certain credential readers to signal a duress code when reading an Authentication Factor. Determining when a duress code has been read is a local matter. In this case the Access_Event property is set to DURESS.

7.1.1.5.1 Single Factor Authentication

In single factor authentication only one authentication factor is required to identify and authenticate the access credential. Depending on the current authentication policy the access user may have a choice of multiple credentials to use.

7.1.1.5.2 Multi-Factor Authentication

In multi-factor authentication two or more authentication factors are used for authentication. Typically when multiple factors are used the first authentication factor is used to identify the credential while the subsequent authentication factors are used to validate the identity. All authentication factors of a multi-factor authentication are expected to be configured in the same Access Credential object.

If a timeout is specified for the current authentication policy and not all authentication factors are read within that time then authentication fails and access is denied. In this case the Access_Event property is set to DENIED_AUTHENTICATION_FACTOR_TIMEOUT.

If one of the authentication factors presented is not the value expected or is presented in an incorrect order then it is a local matter as to whether authentication fails immediately or whether the access user is given subsequent chances to present the correct authentication factor. If authentication fails immediately then the Access_Event property is set to DENIED_INCORRECT_AUTHENTICATION_FACTOR.

If the access user is allowed subsequent attempts but fails to present the correct authentication factor within a certain number of attempts then the Access_Event property is set to DENIED_MAX_ATTEMPTS. The maximum number of attempts the access user is allowed is a local matter.

7.1.1.5.3 External Authentication

If the authentication decision is made by an external process, such as a remote server or human operator, it may be possible that the authentication process becomes unavailable. When this occurs and when there is no secondary authentication process available, the authentication fails and the Access_Event property is set to DENIED_AUTHENTICATION_UNAVAILABLE.

7.1.1.6 Authorization Mode

The Authorization Mode determines how authorization is performed at the Access Point. An Access Point object is not required to support all of these authorization modes but is required to support at least AUTHORIZE.

The required property ***Authorization_Mode,*** of type *BACnetAuthorizationMode*, specifies the authorization mode in effect.

The enumeration *BACnetAuthorizationMode* has the values:

AUTHORIZE	The access rights of an active credential are evaluated aside from other authorization checks.
GRANT_ACTIVE	An active credential is granted access without evaluating the access rights assigned to the credential. Other authorization checks may still lead to denying access.
DENY_ALL	Any credential is denied access, except an active credential that has master exemption. When denied the Access_Event property is set to DENIED_DENY_ALL
VERIFICATION_REQUIRED	The access rights of an active credential are evaluated, in addition to other possible authorization checks. Granting access requires external verification. In this case the Access_Event property is set to VERIFICATION_REQUIRED and the access point waits for the external verification. The external verification process and the mechanism by which the verification requested and the result is provided to the access point is a local matter. If the external verification process denies access then the Access_Event property is set to DENIED_VERIFICATION_FAILED. If there is no external verification result within the time specified by the Verification_Time property then the Access_Event property is set to DENIED_VERIFICATION_TIMEOUT.
AUTHORIZATION_DELAYED	The access rights of an active credential are evaluated, in addition to other possible authorization checks. Granting access is delayed by the time specified by the Verification_Time property. This provides an external verification process the opportunity to deny access. In this case the Access_Event property is set to AUTHORIZATION_DELAYED and the access point waits for the external verification. The external verification process and the mechanism by which the verification result is provided to the access point is a local matter. If the external verification process denies access within the time specified in the Verification_Time property then the Access_Event property is set to DENIED_VERIFICATION_FAILED. If there is no external verification result within the time specified by the Verification_Time property then this authorization check succeeded.

NONE	No authorization functionality takes place at this access point and no authorization events (e.g., grant or any deny events) are generated. This may be used to implement special access point functionality, such as a guard tour or muster point, where authorization checks are not required.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to allow proprietary authorization modes other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

Note that locking down a door requires the Access Door to be commanded accordingly. Setting the authorization mode to DENY_ALL is insufficient to achieve this. Credentials with master exemption may still be granted access.

7.1.1.7 Authorization Decision

Authorization is the process of determining whether or not the credential, which has been used to request access at an access point, will be permitted access. The authorization process completes when the access decision to grant or deny has been reached. Typically, there are multiple authorization criteria used to determine if access will be granted. Examples of authorization criteria are checking access rights, checking passback violations, checking threat level, checking occupancy limits, etc. It is a local matter as to what order the authorization criteria are checked. The authorization criteria, used to determine whether access is allowed, may change according to the time of day, location and the credential used.

If all authorization criteria are successful then the credential is granted access at the access point. In this case, the Access_Event property is set to GRANTED.

If even one authorization check fails, then access is denied and the Access_Event property is set to the appropriate deny event. If there is no access event, either defined or proprietary, which is specific to the actual reason why access was denied then the Access_Event property is set to DENIED_OTHER.

Access may be denied if the authorization process detects an inconsistency with the access request such as when an access user requests access to a zone but there is no record of that access user being in the building. How the inconsistency is determined is a local matter. In this case access is denied and the Access_Event property is set to DENIED_UNEXPECTED_LOCATION_USAGE.

The optional property **Verification_Time**, of type *Unsigned*, specifies the time, in seconds, to wait for external verification when the Authorization_Mode property has a value of AUTHORIZATION_DELAYED or VERIFICATION_REQUIRED.

7.1.1.8 Access Attempts and Lockout

An Access Point may restrict the number of consecutive failed access attempts within a given time. What events are considered a failed access attempt may be exposed and configured, or is a local matter. If the number of failed access attempts exceeds the limit within a specific time window, the Access Point is in lockout state,, denying any access request, except if the Access Credential has Master Exemption.

A maximum failed access attempts value specifies how many failed access attempts are allowed before the Access Point is in lockout state. The failed access attempts are counted during a defined duration of time, and the Access Point lockout may be relinquished after a specified timeout.

The optional property ***Lockout***, of type *BOOLEAN*, is TRUE if the Access Point object is in a lockout state. When the Access Point is in a lockout state any access request will fail except for Access Credentials that have master exemption. For each denied access request the Access_Event will be set to DENIED_LOCKOUT. An Access Point object may be set to a lockout state due to too many failed access attempts, as defined in the Max_Failed_Attempts property, or by writing TRUE to this property.

When the property Lockout becomes TRUE due to too many failed access attempts, the Access_Event property is set to LOCKOUT_MAX_ATTEMPTS. If TRUE is written to this property for any other reason, the Access_Event property is set to LOCKOUT_OTHER. When the Lockout property becomes FALSE, the Access_Event property is set to LOCKOUT_RELINQUISHED.

The optional property ***Lockout_Relinquish_Time***, of type *Unsigned*, is used to specify the time, in seconds, to delay, after the Lockout property has taken on the value TRUE, before automatically relinquishing the lockout state. The lockout state is relinquished by setting the Lockout property to FALSE. A value of zero indicates that the lockout state will not automatically be relinquished.

If this property is present, then the Lockout property is required to be present.

The optional property ***Failed_Attempts***, of type *Unsigned*, indicates the actual number of failed access attempts within the actual Failed_Attempts_Time period. Writing to this property may be allowed for resetting this counter.

This property is set to zero when a successful access attempt occurs or when the property Lockout becomes FALSE.

The optional property ***Failed_Attempt_Events***, of type *List of BACnetAccessEvent*, specifies those access events that are counted as failed attempts. It is the vendor's option if this property is writable, enabling the configuration of the failed attempt events.

If this property is not present, then it is a local mater as to which access events are considered a failed attempt.

The optional property ***Max_Failed_Attempts***, of type *Unsigned*, specifies the maximum number of failed access attempts before Lockout is set to TRUE. If the Failed_Attempts property becomes greater than or equal to the value of this property and this property is not zero, the Lockout property is set to TRUE. Zero indicates that the Lockout property is not set to TRUE as the result of failed access attempts.

If the Max_Failed_Attempts property is present the Failed_Attempts property is required to be present.

The optional property ***Failed_Attempts_Time***, of type *Unsigned*, specifies the time, in seconds, to delay before setting the Failed_Attempts property to zero, after the last failed access attempt.

If the Failed_Attempts_Time property is present the Failed_Attempts property is required to be present.

7.1.1.9 Threat Level

The Access Point exposes its actual Threat Level by the optional property ***Threat_Level***, of type *BACnetAccessThreatLevel*. This is an unsigned value in the range 0...100, with zero is the lowest threat level, effectively switching off the threat level authorization check at this Access Point. For passing the threat level authorization check, the Access Point, authenticated Access Credentials are required to have the same or higher threat authority. Otherwise the authorization fails. In this case the Access_Event property is set to DENIED_THREAT_LEVEL.

The Threat Level, if supported, can be set from other processes, in order to adjust security, locking out Access Credentials with insufficient threat authority. This enables to increase the security level without any modification of access rights.

7.1.1.10 Occupancy Enforcement

The Access Point supports configuration of enforcement of occupancy limits and occupancy counting. Access Credentials may be exempted from occupancy limit enforcement.

The optional property ***Occupancy_Upper_Limit_Enforced***, of type *BOOLEAN*, indicates whether the upper occupancy limit of the Access Zone for which the Access Point is an Entry Access Point is enforced. If enforced, authorization fails if the access controlled zone's occupancy is greater than or equal to its upper occupancy limit, unless the credential is exempted from this authorization check. When this authorization check fails, the Access_Event property is set to DENIED_UPPER_OCCUPANCY_LIMIT.

The optional property ***Occupancy_Lower_Limit_Enforced***, of type *BOOLEAN*, indicates whether the lower occupancy limit of the Access Zone for which the Access Point is an Exit Access Point is enforced. If enforced, authorization fails if the access controlled zone's occupancy is lower than or equal to its lower occupancy limit, unless the credential is exempted from this authorization check. When this authorization check fails, the Access_Event property is set to DENIED_LOWER_OCCUPANCY_LIMIT.

The optional property ***Occupancy_Count_Adjust***,, of type BOOLEAN, indicates whether (TRUE) this object will adjust the occupancy count of the zones for which it controls access, or not (FALSE). The occupancy count is decremented for the zone for which this Access Point is an Exit Access Point and incremented for the zone for which this Access Point is an Entry Access Point.

Occupancy count is adjusted if the credential holder passes through the access point. How this is determined is a local matter. The occupancy count of the zones is adjusted by writing a negative amount to the Adjust_Value property of the exit Access Zone object and the corresponding positive amount to the Adjust_Value property of the entry Access Zone object.

If this property is not supported the Access Point object behaves as if the value is FALSE

7.1.1.11 Accompaniment

The optional property ***Accompaniment_Time***, of type Unsigned, specifies the time, in seconds, to wait for a second credential to be presented at this Access Point when the original credential requires accompaniment. If an accompanying credential is not presented within this time the authorization of the original credential fails and the Access_Event property is set to DENIED_NO_ACCOMPANIMENT.

7.1.1.12 Access Event Reporting

The Access Point object may support intrinsic reporting of access alarm events and access transaction events. The mechanism is based on the new ACCESS_EVENT event algorithm. This algorithm may also be applied on Access Point objects by Event Enrollment objects. For details of the event algorithm and the use of Event Enrollment objects see main section Event Reporting and Logging below.

The basic value representing access events is an enumeration of possible authentication and authorization decisions and maybe subsequent actions taken by a user. This enumeration is defined by the new data type *BACnetAccessEvent.*

NONE	The Access Point did not yet determine any access event. This is not a reported event. It is required to enable algorithmic ACCESS_EVENT reporting. It is also used when the Access Point object is not generating access events.
GRANTED	Access granted to the presented credential.

MUSTER	If the Access Point is a muster point a muster event is generated when an Access Credential is presented.
PASSBACK_DETECTED	A passback violation for the presented Access Credential has been detected.
DURESS	A duress incident was detected at this Access Point.
TRACE	A traced credential has been presented.
LOCKOUT_MAX_ATTEMPTS	The Access Point is a lockout state due to maximum failed authentication attempts.
LOCKOUT_OTHER	The Access Point is in a lockout state due to any reason other than maximum failed authentication attempts.
LOCKOUT_RELINQUISHED	The Access Point has relinquished the lockout state.
LOCKED_BY_HIGHER_PRIORITY	The controlled Access Door is commanded at a higher priority.
OUT_OF_SERVICE	The Out_Of_Service flag of the Access Point has been set to TRUE.
OUT_OF_SERVICE_RELINQUISHED	The Out_Of_Service flag of the Access Point has been set to FALSE.
ACCOMPANIMENT_BY	The credential presented accompanies the previous credential.
AUTHENTICATION_FACTOR_READ	An authentication factor has been read. This event indicates a successful read of an authentication factor in single-factor or multi-factor authentication.
AUTHORIZATION_DELAYED	Authorization of a credential is delayed to allow time for an external process to deny access.
VERIFICATION_REQUIRED	Authorization of a credential requires verification from an external process.
NO_ENTRY_AFTER_GRANTED	Access was granted to the presented credential but the physical door was not opened.
DENIED_DENY_ALL	Access denied because the authorization mode of the Access Point is set to DENY_ALL.
DENIED_UNKNOWN_CREDENTIAL	Access denied due to unknown credential. The authentication factor presented did not match any known authentication factor.
DENIED_AUTHENTICATION_UNAVAILABLE	Access denied because the authentication and authorization decision is unavailable.
DENIED_AUTHENTICATION_FACTOR_TIMEOUT	Access denied due to required authentication factor for multi-factor authentication not presented within time.

DENIED_INCORRECT_AUTHENTICATION_FACTOR	Access denied due to the authentication factor presented for a multi-factor-authentication not being the one expected.
DENIED_ZONE_NO_ACCESS_RIGHTS	Access denied due to evaluation of TRUE of a negative access rule for the access zone.
DENIED_POINT_NO_ACCESS_RIGHTS	Access denied due to evaluation of TRUE of a negative access rule for the access point.
DENIED_NO_ACCESS_RIGHTS	Access denied due to no positive access rule found for the access zone or access point.
DENIED_OUT_OF_TIME_RANGE	Access denied due to the presented Access Credential not being valid at this Access Point or Access Zone at this time.
DENIED_THREAT_LEVEL	Access denied due to insufficient threat authority for the presented Access Credential.
DENIED_PASSBACK	Access denied due to a passback violation for the Access Credential.
DENIED_UNEXPECTED_LOCATION_USAGE	Access denied due to the Access Credential used at a location which violates local consistency rules.
DENIED_MAX_ATTEMPTS	Access denied due to too many failed access attempts at the access point..
DENIED_LOWER_OCCUPANCY_LIMIT	Exit from a zone for which this Access Point is an Exit Access Point is denied due to zone occupancy below or at the minimum limit.
DENIED_UPPER_OCCUPANCY_LIMIT	Access to a zone for which this Access Point is an Entry Access Point is denied due to zone occupancy at or above the maximum limit.
DENIED_AUTHENTICATION_FACTOR_LOST	Access denied due to the authentication factor used being reported as lost.
DENIED_AUTHENTICATION_FACTOR_STOLEN	Access denied due to the authentication factor used being reported as stolen.
DENIED_AUTHENTICATION_FACTOR_DAMAGED	Access denied due to the authentication factor used being reported as damaged.
DENIED_AUTHENTICATION_FACTOR_DESTROYED	Access denied due to the authentication factor used being reported as destroyed.
DENIED_AUTHENTICATION_FACTOR_DISABLED	Access denied due to the authentication factor used is disabled for unspecified or unknown reasons.
DENIED_AUTHENTICATION_FACTOR_ERROR	Access denied due to the authentication factor used had a read error.
DENIED_CREDENTIAL_UNASSIGNED	Access denied due to the Access Credential used has not yet been assigned to an Access User.
DENIED_CREDENTIAL_NOT_PROVISONED	Access denied due to the Access Credential used is not yet provisioned.

DENIED_CREDENTIAL_NOT_YET_ACTIVE	Access denied due to the Access Credential used is not yet active.
DENIED_CREDENTIAL_EXPIRED	Access denied due to the Access Credential used is expired.
DENIED_CREDENTIAL_MANUAL_DISABLE	Access denied due to the Access Credential used is manually disabled.
DENIED_CREDENTIAL_LOCKOUT	Access denied due to the Access Credential used is locked out.
DENIED_CREDENTIAL_MAX_DAYS	Access denied due to the number of days the credential may be used has been exceeded.
DENIED_CREDENTIAL_MAX_USES	Access denied due to the number of allowed uses of the Access Credential used has been exceeded.
DENIED_CREDENTIAL_INACTIVITY	Access denied due to the Access Credential used being disabled after a period of inactivity.
DENIED_CREDENTIAL_DISABLED	Access denied due to the Access Credential used is disabled for unspecified or unknown reasons.
DENIED_NO_ACCOMPANIMENT	Access denied due to the expected accompanying Access Credential not being presented.
DENIED_INCORRECT_ACCOMPANIMENT	Access denied due to the accompanying Access Credential presented was incorrect
DENIED_LOCKOUT	Access denied due to the Access Point being in lockout state.
DENIED_VERIFICATION_FAILED	Access denied due to an external process denying access when verification was required.
DENIED_VERIFICATION_TIMEOUT	Access denied due to an external process failed to send a response, in the allotted time, when verification was required.
DENIED_OTHER	Access is denied for unspecified reasons.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to indicate Access Events other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

The required property ***Access_Event***, of type *BACnetAccessEvent*, indicates the last access event that has occurred at the access point. An Access Point object is not required to support all BACnetAccessEvent enumeration values.

For COV reporting and remote event enrollment using COV subscription, COV Notifications are issued whenever the value of this property is updated, even if the new value is the same as the old value. To support this using standard BACnet mechanisms and services, the following series of operations are performed atomically by the Access Point object when updating the Access_Event property:

(1) The new access event is set in the Access_Event property,
(2) If this event is the start of a new access transaction the value of the Access_Event_Tag property is incremented.

(3) The current date and time is stored in the Access_Event_Time property.
(4) The reference to the Access Credential object which is associated with this event is stored in the Access_Event_Credential property.
(5) The value of the authentication factor which is associated with this event is stored in the Access_Event_Authentication_Factor property.

A COV notification is initiated to all COV subscribers after updating Access_Event. COV notifications are also initiated if the Status_Flags property changes.

The required property ***Access_Event_Tag,*** of type *Unsigned*, is a numeric value which identifies the access transaction to which the current access event belongs. Multiple access events may be generated in a single access transaction.

The value of this property increases monotonically for each new access transaction. It may be implemented using modulo arithmetic. If the value is at its range limit and needs to be incremented, it is wrapped back to zero.

The required property ***Access_Event_Time***, of type *BACnetTimeStamp* indicates the most recent update time of the Access_Event property. This property changes its value on each update of Access_Event. This property is required to be present if the Access_Event property is present. Update times of type Time or Date use "wildcard" (X'FF') in each octet, and Sequence number update times use the value 0 if no update has yet occurred.

The required property ***Access_Event_Credential***, of type *BACnetDeviceObjectReference*, holds the Access Credential object that corresponds to the last access event specified in Access_Event, if applicable. This property contains 4194303 for the instance part of the object identifier and for the device instance part of the device identifier, if present, under the following conditions:

(a) there is no credential recognized up to now, or
(b) there is no credential that is associated to the current access event, or
(c) the credential of the authentication factor that is associated to the current event is unknown, or
(d) the device chooses not to expose the credential.

The optional property ***Access_Event_Authentication_Factor***, of type *BACnetAuthenticationFactor* holds the last Authentication Factor that corresponds to the last access event. This property contains a value of format type UNDEFINED, with no content (no octets) in the Value field, under the following conditions:

(a) there was no authentication factor read up to now, or
(b) there is no authentication factor that is associated to the current access event, or
(c) the device chooses not to expose the authentication factor.

For the definition of BACnetAuthenticationFactor see the Credential Reader Interface section.

The Access Point supports the intrinsic reporting of two different levels of Access Events:

- **Access Alarm Events**: These are events requiring operator attention and handling.
- **Access Transaction Events**: These are events which are typically logged only, not requiring operator attention.

To support these two levels, the Access Point applies the ACCESS_EVENT algorithm twice in its intrinsic reporting. A specific Access Event may be reported as an Access Alarm Event and as an Access Transaction Event at the same time. To parameterize these two algorithms, the Access Point has distinct properties if needed, other properties are shared among both algorithms. Sharing is possible since either the same value is used, or the value is irrelevant or ignored for one of the two levels.

Access events are stateless (i.e., NORMAL to NORMAL transitions only). A single access transaction, such as a request to enter or an operator action, can result in one or more access events. All access events that belong to the same access transaction have the same access event tag.

The Access Point properties are used as follows:

Access Point Properties	**Access Alarm Events**	**Access Transaction Events**
Event_State	Shared, NORMAL always[1]	Shared, NORMAL always[1]
Access_Event	Monitored Value	Monitored Value
Access_Event_Time	Time of update	Time of update
Access_Alarm_Events	List of Access Events to be reported as Access Alarm Events. Acts as List_Of_Access_Events for algorithm.	-
Access_Transaction_Events	-	List of Access Events to be reported as Access Transaction Events. Acts as List_Of_Access_Events for algorithm.
Notification_Class	Notification Class Properties: Priority[TO-NORMAL][2], Ack_Required(to-normal) flag and Recipient_List for Recipients of Access Alarm Event notifications.	-
Transaction_Notification_Class	-	Notification Class Properties: Priority[TO-NORMAL][2] and Recipient_List for Recipients of Access Transaction Event notifications. Ack_Required(to-normal) flag is not used, always FALSE in notification.
Event_Enable	TO-NORMAL bit applies to enable Access Alarm Event notifications	TO-NORMAL bit applies to enable Access Transaction Event notifications
Notify_Type	Used for notification.	Not used, always EVENT in event notification.
Acked_Transitions	Indicates acknowledgement of Access Alarm Events	Not affected
Event_Time_Stamps	Time stamp [TO-NORMAL][2] holds time of last Access Alarm Event	Not affected

[1] For Access Events, the Event_State always transitions from NORMAL to NORMAL, since all access events are stateless. The BACnet framework requires conveying From_State and To_State in event notifications, as well as having the Event_State property.
[2] Array element [TO-NORMAL] is the array's element with index 3.

The Notification Class object properties are used as follows. Note that for Access Alarm Events the standard Notification Class object is used as referenced by the Access Point’s Notification_Class property, while for Access Transaction Events the Notification Class object referenced by the Access Point’s Transaction_Notification_Class property is used. If the Transaction_Notification_Class property is not present, then the Notification Class object referenced by the Notification_Class property is used.

Notification Class Properties	**Access Alarm Events**	**Access Transaction Events**
Notification Class Instance	(referenced by Access Point’s Notification_Class)	(referenced by Access Point’s Transaction_Notification_Class if present, otherwise referenced by Notification_Class)
Priority	Used in event notification	Used in event notification
Ack_Required	Used in event notification and Access Point’s Acked_Transitions regular handling.	Not used, always FALSE in notification. Access Point’s Acked_Transitions are not affected.

Event notification parameters differ for Access Alarm Events and Access Transaction Events.

Event Notification Parameters	**Access Alarm Events**	**Access Transaction Events**
timeStamp	Current time. Is also stored in Access Point’s Event_Time_Stamps [TO-NORMAL]	Current time. Access Point’s Event_Time_Stamps [TO-NORMAL] is not affected!
notificationClass	Notification Class referenced by Access Point’s Notification_Class property	Notification Class referenced by Access Point’s Transaction_Notification_Class property if present, otherwise Notification Class referenced by Notification_Class property
priority	Notification Class Priority	Notification Class Priority
eventType	ACCESS_EVENT	ACCESS_EVENT
messageText (OPTIONAL)	<any>	<any>
notifyType	Access Point’s Notify_Type	EVENT
ackRequired	Notification Class’ Ack_Required	FALSE
fromState	NORMAL	NORMAL
toState	NORMAL	NORMAL
eventValues	CHOICE = ACCESS_EVENT	CHOICE = ACCESS_EVENT
Access_Event	Access Point’s new Access_Event value	Access Point’s new Access_Event value
Status_Flags	Access Point’s Status_Flags value	Access Point’s Status_Flags value
Access_Event_Tag	Access Point’s Access_Event_Tag value	Access Point’s Access_Event_Tag value
Access_Event_Time	Access Point’s new Access_Event_Time value	Access Point’s new Access_Event_Time value
Access_Credential	Access Point’s Access_Event_Credential value	Access Point’s Access_Event_Credential value
Authentication_Factor (OPTIONAL)	Access Point’s Access_Event_Authentication_Factor value if present	Access Point’s Access_Event_Authentication_Factor value if present

The optional property ***Notification_Class***, of type *Unsigned*, refers to the notification class for Access Alarm Events at the Access Point.

This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Transaction_Notification_Class***, of type *Unsigned*, refers to the notification class for Access Transaction Events at the Access Point. If this property is not present, the Access Transaction Events are distributed through the standard Notification Class, referenced by the property Notification_Class, but still ignoring Ack_Required of that Notification Class.

The optional property ***Access_Alarm_Events***, of type *List of BACnetAccessEvent*, holds the Access Events to be reported as Access Alarm Events.
An Access Alarm Event is reported when Access_Event is updated and the new value is equal to one of the values of this property.
This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Access_Transaction_Events***, of type *List of BACnetAccessEvent*, holds the Access Events to be reported as Access Transaction Events.
An Access Transaction Event is reported when Access_Event is updated and the new value is equal to one of the values of this property.
This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Event_Enable***, of type *BACnetEventTransitionBits*, allows specifying which event state transitions are reported. Deactivating the TO-NORMAL bit of this property effectively switches off reporting of any access event. Deactivating the TO-FAULT bit of this property switches off reporting of Reliability related fault events.
This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Acked_Transitions***, of type *BACnetEventTransitionBits*, indicates acknowledgement of transitions of the Event_State property. The TO-NORMAL bit is related to Access Alarm Events that require acknowledgement. Access Transaction Events do not affect this property. The TO-FAULT bit is related to fault events that require acknowledgement.
This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Notify_Type***, of type *BACnetNotifyType*, specifies whether the Access Alarm Events are notified as ALARM or EVENT. Access Transaction Events are always reported as EVENT, ignoring the value of this property. The notify type is a classification of notifications typically used in client role processes. It has no effect on the server role process behavior.
This property is required to be present if the Access Point object supports intrinsic reporting.

The optional property ***Event_Time_Stamps***, a *BACnetArray of BACnetTimeStamp*, holds the time stamp as conveyed in the most recent notifications for the individual event state transitions. The array element associated with the TO-NORMAL transition (i.e. array element 3) holds the time stamp of the most recent Access Alarm Event. Access Transaction Events do not modify this property.
This property is required to be present if the Access Point object supports intrinsic reporting.

7.1.1.13 COV Reporting

The Access Point object supports COV reporting on object level. Property level COV reporting support is at the vendor's discretion.

Although the Access_Event property is the key value of COV reporting, it may not change its value on consecutive access events, Access_Event may be updated with the same value it already has. Therefore object level COV reporting is triggered when Access_Event_Time changes, or Status_Flags changes.

The following properties are reported in the COV notification:

- Access_Event
- Status_Flags
- Access_Event_Tag
- Access_Event_Time
- Access_Event_Credential
- Access_Event_Authentication_Factor (if present)

In COV reporting as defined by BACnet, a notification is sent out on subscription or on re-subscription without a change of the Access_Event_Time property. On the other hand, COV notifications are sent on change of this property.

The COV subscriber's requirements are as follows:

- The subscriber needs to be able to distinguish whether it received a COV notification due to an update of Access_Event and Access_Event_Time, or COV (re-)subscription.
- The subscriber receives a COV notification whenever Access_Event_Time is updated.

By basing COV reporting on the update of Access_Event_Time, the client becomes able to verify if it got an update of Access_Event, or the COV notification was sent due to (re-)subscription.

The subscriber's behavior can be sketched as follows:

(Re-)Subscription:

1: Read and cache Access_Event_Time from the object in question.
2: Subscribe for COV notifications from the object

Operation:

3: Receive COV notifications. If the COV notification conveys the same Access_Event_Time time stamp as read from the object or cached from last COV notification, a COV notification caused by the subscription or Status_Flags change was received.
4: If the COV notification conveys another Access_Event_Time time stamp, the COV notification was caused by a real update of Access_Event. Act on it and cache the Access_Event_Time time stamp.
5: Continue at step 3

The COV context in the Active_COV_Subscriptions property of the Device object contains the Access_Event property for those contexts that are related to an object level COV subscription for an Access Point object.

7.1.1.14 Access Door Commanding

The Access Point exposes which Access Door objects are commanded by the Authentication & Authorization Process for the Access Point. Commanding of the Access Door objects referred to by the Access Point object takes place after successful authorization at this Access Point.

The required property ***Access_Doors***, a *BACnetArray of BACnetDeviceObjectReference*, specifies the Access Doors the Access Point commands to unlock when access is granted.

The required property ***Priority_For_Writing***, of type *Unsigned* in the range 1..16, specifies the command priority to be used when commanding the Access Doors.

After successful authorization the following actions occur:

(1) The Access_Event property is set to GRANTED, and
(2) The physical doors, as specified in the Access_Doors property, are commanded with either a PULSE_UNLOCK or EXTENDED_PULSE_UNLOCK door command, at the priority specified by the Priority_For_Writing property.

Commanding the doors may fail due to a higher priority command in effect in the Access Door object. In this case the Access _Event property is set to LOCKED_BY_HIGHER_PRIORITY.

The respective Access Doors may be monitored to verify that they are opened and access takes place. If no access takes place, the Access_Event property is set to NO_ENTRY_AFTER_GRANTED.

7.1.1.15 Muster Station Support

An Access Point object can act as a muster point for muster applications. If it acts as a muster point, every Access Credential recognized is reported by a MUSTER access event.

The optional property ***Muster_Point***, of type *BOOLEAN*, indicates whether the Access Point generates muster access events. A muster event is generated by setting the Access_Event property to MUSTER after an access credential has been presented at the access point. It is a local matter as to whether a muster event is generated for unknown credentials.

7.1.1.16 Access Zone Relationship

Access Points are the entry or exit points of Access Zones. Access Point objects may therefore expose such relationships. The use of Access Zones is optional, the relationship may not exist.

7.1.1.16.1 Entry (“Zone To”)

The Access Point may refer to the Access Zone for which the Access Point acts as an entry point. The Access Point allows entering the Access Zone referred to.

The Access Zone that this Access Point allows entry to is exposed by the optional property ***Zone_To***, of type *BACnetDeviceObjectReference*, which references the respective Access Zone object. This property shall not reference the same Access Zone object as the Zone_From property. If the Access Point is not an entry point of an Access Zone then this property shall contain 4194303 in the instance part of the object identifier, and in the instance part of the device identifier, if present.

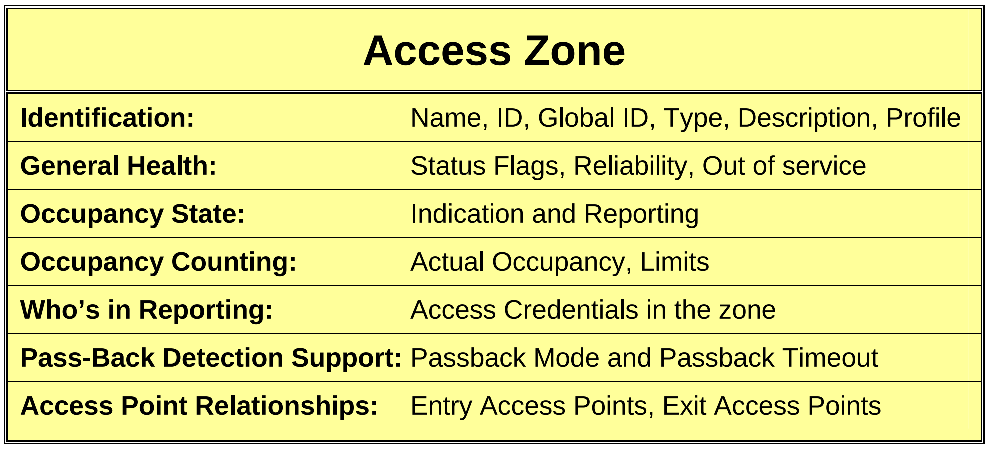
7.1.1.16.2 Exit (“Zone From”)

The Access Point may refer to an Access Zone for which the Access Point acts as an exit point. The Access Point allows leaving the Access Zone referred to.

The Access Zone that this Access Point allows exit from is exposed by the optional property ***Zone_From***, of type *BACnetDeviceObjectReference*, which references the respective Access Zone object. This property shall not reference the same Access Zone object as the Zone_To property. If the Access Point is not an exit point of an Access Zone then this property shall contain 4194303 in the instance part of the object identifier, and in the instance part of the device identifier, if present.

7.1.2 Access Zone Object Type

A secured zone is represented by an Access Zone object. Simple systems may model the Authorization & Authentication Interface without using this object, if the functionality this object represents is not exposed at the interface.
The Access Zone object type provides the following features:



Access Zone	
Identification:	Name, ID, Global ID, Type, Description, Profile
General Health:	Status Flags, Reliability, Out of service
Occupancy State:	Indication and Reporting
Occupancy Counting:	Actual Occupancy, Limits
Who's in Reporting:	Access Credentials in the zone
Pass-Back Detection Support:	Passback Mode and Passback Timeout
Access Point Relationships:	Entry Access Points, Exit Access Points

Figure 7–4, Access Zone Object Type

The concept of commanding an Access Zone was considered not to be appropriate, since securing a zone requires much more than what is in the scope of the Access Zone object, e.g. arming intrusion detectors, switching lighting, etc. The Access Zone does not support any commanding. The existing Command Object type may be used for this purpose.

The Access Zone's Occupancy_State property may indicate the current occupancy state. Intrinsic alarming is on Occupancy_State using the existing CHANGE_OF_STATE algorithm. Event Enrollment objects may be configured to watch Access Zone properties.

Object-level COV reporting is not foreseen. But property-level COV may be supported at a vendor's discretion.

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 2.

7.1.2.1 Identification

These are the standard properties of BACnet objects for object identification, type identification, description, etc.

Control of a secured zone may be shared by multiple controller devices; each involved device will have a representation of this secured zone in the form of an Access Zone object.

When a secured zone is represented in multiple devices, the representing Access Zone objects may not have the same Object_Identifier in each device; however, they may be identified using the Global_Identifier property. It is a local matter as to how these objects are synchronized.

The required property ***Global_Identifier***, of type *Unsigned*, is the global identifier of the secured zone this Access Zone object represents. The global identifier is an internetwork-wide non-zero value which is unique for each secured zone. If a global identifier is assigned to a secured zone, then all Access Zone objects in all devices which represent this zone have the same global identifier value in this property. A value of zero in this property indicates that no global identifier is assigned.

The synchronization of content is a local matter, but the model supports this requirement through the use of standard BACnet services. How the synchronization between devices occurs is considered a local matter. However, this may be done through COV notifications.

7.1.2.2 General Health

Properties are defined which indicate the general health of the object. Reliability indication and out of service is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which may provide more details.

- The IN_ALARM flag is 1 if the property Event_State has a value other than NORMAL.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDDEN flag is always 0.
- The OUT_OF_SERVICE flag is 1 if the property Out_Of_Service is TRUE.

The required property ***Event_State***, of type *BACnetEventState*, indicates the event state associated with the object. The following event states are supported:

NORMAL	Occupancy is considered in allowed range. Occupancy_State has a value that is not listed in Alarm_Values.
OFFNORMAL	Occupancy is outside allowed range. Occupancy_State has a value that is listed in Alarm_Values.
FAULT	Processing or the object is unreliable. Reliability is not NO_FAULT_DETECTED.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of the Authentication & Authorization Process for the Access Zone this object represents. In particular this is the reliability of the values of the properties Occupancy_State, Occupancy_Count and/or Credentials_In_Zone. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

If the Reliability property is present and has any value other than NO_FAULT_DETECTED, then the Event_State property has a value of FAULT. The Reliability property is writable when Out_Of_Service is TRUE.

The required property ***Out_Of_Service***, of type *BOOLEAN*, indicates whether (TRUE) or not (FALSE) the object is out of service.

When the object is out of service, the Reliability property and the corresponding state of the FAULT flag of the Status_Flags property are decoupled and the Reliability property 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 Reliability property respond to changes made to this property while Out_Of_Service is TRUE.

If occupancy counting is supported and the Access Zone object is out of service, then the Occupancy_Count property is decoupled from the processing of occupancy counting. In addition, writing to the Adjust_Value property does not modify the Occupancy_Count. The Occupancy_Count property 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 Occupancy_State property respond to changes made to this property while Out_Of_Service is TRUE.

7.1.2.3 Occupancy State and Reporting

The Access Zone supports indication and reporting of the occupancy state. For intrinsic reporting, the Occupancy_State property is monitored and the standard Change-Of-State event algorithm is applied. The value of Occupancy_State is conveyed as the "New State" parameter of event notifications, along with the Status_Flags. For details of the Change-Of-State algorithm see clause 13 of the BACnet standard [STD].

The required property ***Occupancy_State***, of type *BACnetAccessZoneOccupancyState*, reflects the occupancy state of the Access Zone. It uses Occupancy_Count and the occupancy limit properties to evaluate its state. Proprietary evaluations and states are possible.

BACnetAccessZoneOccupancyState is an enumeration of possible occupancy states:

NORMAL	This is the default occupancy state when no other standard or proprietary states are applicable or occupancy counting is disabled
BELOW_LOWER_LIMIT	If the Occupancy_Lower_Limit property is present and the Occupancy_Count property is lower than this value
AT_LOWER_LIMIT	If the Occupancy_Lower_Limit property is present and the Occupancy_Count property is equal to this value.
AT_UPPER_LIMIT	If the Occupancy_Upper_Limit property is present and the Occupancy_Count property is equal to this value
ABOVE_UPPER_LIMIT	If the Occupancy_Upper_Limit property is present and the Occupancy_Count property is greater than this value
DISABLED	This is the occupancy state when occupancy counting is disabled for this object. Occupancy counting is disabled when the Occupancy_Count_Enable property is FALSE.
NOT_SUPPORTED	This is the occupancy state when occupancy counting is not supported by this object.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to indicate proprietary occupancy states other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD]..

For intrinsic reporting of occupancy state events, the existing Change-Of-State algorithm is used. This requires the presence of the following properties.

The optional property ***Alarm_Values***, a *List of BACnetAccessZoneOccupancyState*, defines which occupancy states of the Occupancy_State property are reported as alarms (Event_State becomes OFFNORMAL). This property is required if intrinsic reporting is supported by this object.

The optional property ***Time_Delay***, of type *Unsigned*, specifies the minimum period of time in seconds that the Occupancy_State property value must remain equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated; or not equal to any of the values in the Alarm_Values property before a TO-NORMAL event is generated. This property is required if intrinsic reporting is supported by this object.

The optional property ***Notification_Class***, of type *Unsigned*, specifies the notification class to be used when handling and generating event notifications for this object. This property is required if intrinsic reporting is supported by this object.

The optional property ***Event_Enable***, of type *BACnetEventTransitionBits*, has 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.

The optional property ***Acked_Transitions***, of type *BACnetEventTransitionBits*, has three flags that separately indicate the receipt of acknowledgments for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events. This property is required if intrinsic reporting is supported by this object.

The optional property ***Notify_Type***, of type *BACnetNotifyType*, indicates whether the notifications generated by the object should be EVENTS or ALARMS. This classification is typically used by client role processes receiving notifications. It has no impact on the behavior of the authentication and authorization process. This property is required if intrinsic reporting is supported by this object.

The optional property ***Event_Time_Stamps***, a *BACnetArray of BACnetTimeStamp*, holds the times of the last event notifications for TO-OFFNORMAL, TO-FAULT, and TO-NORMAL events, respectively. This property is required if intrinsic reporting is supported by this object.

7.1.2.4 Occupancy Counting and Limits

The Access Zone may support counting occupancy if appropriate entry and exit hardware is present. The precision of counting depends on the entry and exit hardware used. The actual occupancy count is incremented if access takes place at an Entry Access Point, and decremented if exit takes place at an Exit Access Point. Incrementing or decrementing may be accomplished through other means, which are a local matter.

The actual occupancy count is exposed by the optional property ***Occupancy_Count,*** of type *Unsigned*. If Occupany_Count_Enable is FALSE, this property has a value of zero. The value of the Occupancy_Count property may be adjusted by writing to the Adjust_Value property. The Occupancy_Count property is writable when Out_Of_Service is TRUE. When Out_Of_Service becomes FALSE it is a local matter as to what the value the occupancy count is set to. This property is required to be present if the Access Zone object supports occupancy counting.

The optional property ***Occupancy_Count_Enable***, of type *BOOLEAN*, indicates whether occupancy counting is in effect (TRUE) or not (FALSE). This property is required to be present if the Access Zone object supports occupancy counting.

If occupancy counting is not supported by the Access Zone object this property always has a value of FALSE, if present, and the Occupancy_State property has a value of NOT_SUPPORTED. If occupancy counting is supported by the Access Zone object and this property has a value of FALSE then the Occupancy_State property has a value of DISABLED.

When this property changes from FALSE to TRUE, it is a local matter as to what value the Occupancy_Count property is set to.

The optional property ***Adjust_Value***, of type *INTEGER*, adjusts the Occupancy_Count property when written. This property is required to be present and writable if the Access Zone object supports occupancy counting. It enables the Access Zone object to avoid race conditions when changing the occupancy count.

The following series of operations are performed atomically when this property is written and the value of the Occupancy_Count_Enable property is TRUE:

(1) The value written to Adjust_Value is stored in the Adjust_Value property.
(2) If the value written is non-zero, then this value is added to the value of the Occupancy_Count property. If the value written is negative and the resulting value of the Occupancy_Count property would be less than zero, then the Occupancy_Count property is set to zero. If the value written is zero, then the value of the Occupancy_Count property is set to zero.

When this property is written and the value of the Occupancy_Count_Enable property is FALSE, then the Adjust_Value property is set to zero. If Adjust_Value has never been written, it has a value of zero.

The optional property ***Occupancy_Upper_Limit,*** of type *Unsigned*, specifies the occupancy upper limit of the zone. If this property has a value of zero then there is no upper limit. If this value is not zero, it shall be greater than the value of the Occupancy_Lower_Limit, if present.

The optional property ***Occupancy_Lower_Limit,*** of type *Unsigned*, specifies the occupancy lower limit of the zone. If this property has a value of zero then there is no lower limit.

If limits are present, an Entry Access Point of the Access Zone may generate corresponding access events if a limit is reached or exceeded. Whether or not events are generated is defined by adding the corresponding events to the respective access event lists of the Access Point where the current access transaction takes place.

Whether the occupancy limits are enforced or not may be enabled or disabled individually at each Access Point of an Access Zone. Enforcement means to deny access if the limit would become exceeded if access takes place. Access Credentials may be exempted from occupancy enforcement checks.

7.1.2.5 "Who Is In" Reporting

The Access Zone may support live view reporting of who is currently in the Access Zone if appropriate entry and exit hardware is present. This information is used by passback detection, or for "Who-Is-In" live-view reporting. If a live-view needs to display persons and person related data instead of Access Credentials, the Access Credential information allows retrieving which Access Users are in the Access Zone.

The list of present Access Credentials is exposed by the optional property ***Credentials_In_Zone***, of type *List of BACnetDeviceObjectReference*, referring to Access Credential objects.

For synchronization purposes among the Access Zone objects present in different access control devices and representing the same secured zone, the Access Zone may be required to expose which Access Credential is added to Credentials_In_Zone, and which one is removed. The reason for adding or removing is not relevant, and may be, among others:

- Regular access
- Regular exit
- Manual removal
- Timed removal
- Etc.

The availability of this information as properties is required if standard BACnet application services are to be used for synchronization among access control devices participating in controlling the same secured zone. Most typically, a property-level COV is applied on the following properties, to keep Credentials_In_Zone synchronized in all access control devices controlling the same secured zone.

The optional property ***Last_Credential_Added***, of type *BACnetDeviceObjectReference*, holds the reference to the Access Credential object which has last been added to the Credentials_In_Zone property. When no Access Credential has been added yet, then this reference contains 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present. If property level COV subscriptions are in place on this property, any update, even with the same value, is reported by a COV notification.

The optional property ***Last_Credential_Added_Time***, of type *BACnetDateTime*, indicates the date and time when a reference to an Access Credential object has last been added to the Credentials_In_Zone property. If this property is present, but no credential has yet been added, then this property does not convey an actual time and has "wildcard" (X'FF') in all octets.

The optional property ***Last_Credential_Removed***, of type *BACnetDeviceObjectReference*, holds the reference to the Access Credential object which has last been removed from the Credentials_In_Zone property. When no Access Credential has been removed yet, then this reference contains 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present. If property level COV subscriptions are in place on this property, any update, even with the same value, is reported by a COV notification.

The optional property ***Last_Credential_Removed_Time***, of type *BACnetDateTime,* indicates the date and time when a reference to an Access Credential object has last been removed from the Credentials_In_Zone property. If this property is present, but no credential has yet been removed, then this property does not convey an actual time and has "wildcard" (X'FF') in all octets.

7.1.2.6 Passback Violation Detection

A Passback violation occurs when two successive entry transactions into the Access Zone occur using the same Access Credential without an intermediate exit transaction or the passback check timeout for the Access Credential did not yet expire.

The validation of passback is part of the authorization. The validation uses Credentials_In_Zone information to verify if the Access Credential is not already in the zone. If it is already listed in the Credentials_In_Zone list, and its timeout has not yet expired, a passback violation is detected.

The Access Zone object specifies the actual passback validation mode for all Entry Access Points of this zone in the optional property ***Passback_Mode***, of type *BACnetAccessPassbackMode*.

The enumeration *BACnetAccessPassbackMode* has the values:

PASSBACK_OFF	Passback violations are not checked.
HARD_PASSBACK	Passback violations are checked, enforced and reported. When a passback violation is detected the Access_Event Property of the corresponding Access Point object is set to DENIED_PASSBACK and the authorization for the credential fails.
SOFT_PASSBACK	Passback violations are checked and reported but not enforced. When a passback violation is detected the Access_Event Property of the corresponding Access Point object is set to PASSBACK_DETECTED.

The optional property ***Passback_Timeout***, of type *Unsigned*, specifies the passback timeout in minutes. The timeout is evaluated individually for every credential used to enter the zone. The timeout period for a particular credential begins at the time of successful access to the zone. After the timeout has expired for a particular credential a passback violation of this credential will no longer be detected. A value of zero or absence of this property indicates passback violations will never time out.

If this property is present, then the property Passback_Mode is required to be present.

7.1.2.6.1 Shared control of a secured zone

In case not all Access Points of a secured zone are controlled by the same access control device, shared control of a secured zone is in place. If passback violation detection is to be achieved for this shared secured zone, some requirements arise:

1. Each access control device involved in securing a certain zone needs to have an Access Zone object for representing the same secured zone.
2. Each Access Zone object representing the same secured zone must have the same global identifier in the Global_Identifier property.
3. The information in the different representations of the same secured zone, i.e. in the Access Zone objects in the different access control devices, needs to be synchronized. How this is achieved is a local matter.

7.1.2.7 Access Point Relationship

The Access Zone object refers to all Access Points to enter or exit the Access Zone. These are the Entry and Exit Access Points respectively.

The Access Zone exposes these Entry and Exit Access Point relations by the two required properties ***Entry_Points*** and ***Exit_Points***, both of type *List of BACnetDeviceObjectReference* to Access Point objects.

The Exit Access Point for one Access Zone is typically the Entry Access Point of a neighboring Access Zone.

7.2 Authentication

Authentication is the process of validating an Access Credential and is considered as a validation check required for authorization. If an authentication fails due to any reason, access is denied.

The data model for authentication consists of two basic representations:

- **Access Credential**
  An Access Credential represents a container of Authentication Factors and assignment of Access Rights. This supports multi-factor cards as well as multi-factor authentication. For multi-factor authentication, all Authentication Factors required at an Access Point need to be contained in a single Access Credential. Access rights are assigned to an Access Credential.

- **Access User**
  An Access User represents an individual person, an asset or a group. A group is any combination of persons and assets. An Access User owns one or multiple Access Credentials, of which the Authentication Factors are used to authenticate the Access User. The model allows operating PACS without Access User objects. This may be due to resource constraints, security reasons or any other design considerations.

An Authentication Factor read is processed and validated by the Credential Reader Process and formatted into a BACnetAuthenticationFactor before made available at the Credential Reader Interface.
The Authentication & Authorization Process of the PACS gets this formatted Authentication Factor at the Credential Reader Interface. It scans through its internal credential database (i.e. Access Credentials representation) to see whether the Authentication Factor is known, i.e. listed as one of the Authentication Factors of an Access Credential.

If the matching Access Credential (contains the Authentication Factor!) representation is found, and single factor authentication is in place, this Access Credential is authenticated.
If the Authentication Policy indicates multi-factor authentication, then only when all Authentication Factors are read as required by the active authentication policy of the Access Point, and match with those of a specific Access Credential object, the Access Credential is authenticated.

If the authenticated Access Credential is assigned to an Access User, this Access User is authenticated too, but implicitly.

7.2.1 Access Credential Object Type

There are many types of Access Credentials today, and various new types may come up in the future. Some are a single Authentication Factor; others are typically physical containers of multiple Authentication Factors.

- Media Conveyed Authentication Factors:
  - Numbers from magnetic stripes
  - Structure elements of magnetic stripes
  - Numbers from contact and contact-less cards
  - Structure elements of contact and contact-less cards
  - Bar codes of various shape and type
  - Codes stored in smart cards
  - Others
- Biometric Authentication Factors
  - Finger print
  - Iris image or pattern
  - Hand shape or pattern
  - Face geometry
  - Face image
  - Others
- Remembered Authentication Factors
  - PIN
  - Daily-Code
  - Others

To achieve interoperability among different Access Control devices, and because of the lack of a general standard covering any type of such Authentication Factors, the data model defines well known Authentication Factors while allowing for vendor or organization specific definitions. For details see the Credential Reader Interface section below.

The Access Credential object type holds a single or multiple Authentication Factors.

The Access Credential object represents a container of related Authentication Factors such as card, PIN, biometric, etc. Authentication Factors are grouped in a single container when each factor has the identical access rights and when one or more authentication factors are used in multi-factor authentication (i.e., card and biometric) .The authentication factors in a credential may physically exist on the same media, such as a smart card (including FIPS-201 card), or may exist as physically separate entities, such as a card and PIN. Access Rights are assigned to an Access Credential object. The Access Credential object may be assigned to an Access User.

Access Credential	
Identification:	Name, ID, Global ID, Type, Description, Profile
General Health:	Status Flags, Reliability
Status:	Access Credential Status
Authentication:	Authentication Factors
Validity:	Time Window
Disabling:	Disabling of the Access Credential
Use Counting:	Absolute, remaining Uses
Relationship:	Access User Relationship
Access Rights:	Assigning and enabling Access Rights
Threat Handling:	Threat Authority
Trace & Search:	Tracing and Searching Support
Credential Special Support:	Extended Door Unlock Time, Exemptions

Figure 7–5, Access Credential Object Type

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 5.

7.2.1.1 Identification

These are the standard properties of BACnet objects for object identification, type identification, description, etc.

Since the same Access Credentials have to be represented in multiple devices, it is recommended that all Access Credential objects representing the same Access Credential have the same relevant content. Some content may not be required in all devices, e.g. a finger print Authentication Factor is not required when no finger print reader is present at or used by the device.

When an access credential is represented in multiple devices, the representing Access Credential objects may not have the same Object_Identifier in each device; however, they may be identified using the Global_Identifier property. It is a local matter as to how these objects are synchronized.

The required property ***Global_Identifier***, of type *Unsigned*, is the global identifier of the access credential this object represents. The global identifier is an internetwork-wide non-zero value which is unique for each access credential. If a global identifier is assigned to an access credential, then all Access Credential objects in all devices which represent this credential have the same global identifier value in this property. A value of zero in this property indicates that no global identifier is assigned.

The synchronization of content is a local matter, but the model supports this requirement through the use of standard BACnet services.

7.2.1.2 General Health

Properties are defined which indicate the general health of the object. Reliability indication is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which may provide more details.

- The IN_ALARM flag is always 0. There is no Event_State property in this object.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDDEN flag is always 0.
- The OUT_OF_SERVICE flag is always 0.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of whether this object is "reliable" as far as the BACnet Device can determine and, if not, why. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

7.2.1.3 Access Credential Status

The Access Credential status indicates the validity of the Access Credential object for authentication, and provides reasons for the Access Credential being inactive.

The required property ***Credential_Status***, of type *BACnetBinaryPV*, indicates whether the Access Credential is active or inactive. Only the value ACTIVE enables the Access Credential to be used for authentication. While the list in property Reason_For_Disable is non-empty the status of the Access Credential is INACTIVE, otherwise it is ACTIVE.

When an inactive credential is used to request access, the authentication of this credential fails and access to the access point is denied. In this case, the Access_Event property of the Access Point object where the credential has attempted access is set to the value which corresponds to the reason this credential is disabled, as specified in the Reason_For_Disable property.

The required property ***Reason_For_Disable***, of type *List of BACnetAccessCredentialDisableReason*, contains a list of reasons why the credential has been disabled. The Access Credential can be disabled for multiple reasons at the same time. While the Credential_Status property has a value INACTIVE this list shall not be empty. When an entry is removed from this list which results in the list becoming empty the Credential_Status is set to ACTIVE.

The enumeration *BACnetAccessCredentialDisableReason* of reasons for which the credential can be disabled is as follows:

DISABLED	The credential is disabled for unspecified reasons.
DISABLED_NEEDS_PROVISIONING	The credential needs further provisioning which can include vendor proprietary data.
DISABLED_UNASSIGNED	The credential is not currently assigned to any access user.
	This status is assigned only if the property Belongs_To is present and contains instance 4194303 in the object identifier.
DISABLED_NOT_YET_ACTIVE	The credential is not yet valid at this time. The current time is before the Activation_Time.

DISABLED_EXPIRED	The credential is not valid any more at this time. The current time is after the Expiry_Time.
DISABLED_LOCKOUT	Too many retries in multi-factor authentications have been performed.
DISABLED_MAX_DAYS	The maximum number of days for which this credential is valid for has been reached.
DISABLED_MAX_USES	The maximum number of uses for which this credential is valid for has been reached.
DISABLED_INACTIVITY	The credential has exceeded the allowed period of inactivity.
DISABLED_MANUAL	The credential is commanded to be disabled by a human operator.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to indicate disable reasons other than those defined by the BACnet standard [STD].. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

A PACS is not required to support all these reasons for disable.

There is no intrinsic reporting foreseen in the Access Credential object. The Access Credential object does not support object-level COV reporting.

When access is requested using a credential that is inactive, access is denied. In this case the Access Point object, representing the access point where access was requested, sets its Access_Event property as defined in the following table:

Credential Disable Reason	**Applicable Access Event**
DISABLED_UNASSIGNED	DENIED_CREDENTIAL_UNASSIGNED
DISABLED_NEEDS_PROVISIONING	DENIED_CREDENTIAL_NOT_PROVISONED
DISABLED_NOT_YET_ACTIVE	DENIED_CREDENTIAL_NOT_YET_ACTIVE
DISABLED_LOCKOUT	DENIED_CREDENTIAL_LOCKOUT
DISABLED_MAX_DAYS	DENIED_CREDENTIAL_MAX_DAYS
DISABLED_MAX_USES	DENIED_CREDENTIAL_MAX_USES
DISABLED_INACTIVITY	DENIED_CREDENTIAL_INACTIVITY
DISABLED_MANUAL	DENIED_CREDENTIAL_MANUAL_DISABLE
DISABLED	DENIED_CREDENTIAL_DISABLED

If Reason_For_Disable contains multiple values, it is a local matter to which corresponding access event the Access_Event property is set to.

7.2.1.4 Authentication Factors

Authentication Factors are values of type BACnetCredentialAuthenticationFactor which are used to identify the Access Credential. The Authentication & Authorization Process uses these values to compare with a BACnetAuthenticationFactor value read from the Credential Reader Interface, in order to find the according Access Credential object. Note that this is not the BACnet object identifier. For details about the BACnetAuthenticationFactor data type see the Credential Reader Interface section.

The Access Credential object exposes these values in the required property ***Authentication_Factors***, of type *BACnetArray of BACnetCredentialAuthenticationFactor* elements.

Each element of the array has two fields:

Disable	This field, of type BACnetAccessAuthenticationFactorDisable, specifies whether the corresponding authentication factor is disabled or not. Any value other than NONE indicates that the authentication factor is not valid for authentication.

The following authentication factor disable values are defined:

DISABLED	The physical authentication factor is disabled for unspecified reasons.
DISABLED_LOST	The physical authentication factor is reported to be lost.
DISABLED_STOLEN	The physical authentication factor is reported to be stolen.
DISABLED_DAMAGED	The physical authentication factor is reported to be damaged.
DISABLED_DESTROYED	The physical authentication factor is reported to be destroyed.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to specify disable values other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see Clause 23.1 of the BACnet standard [STD].

Authentication-Factor	This field, of type BACnetAuthenticationFactor, specifies the authentication factor that belongs to this credential.

Any access attempt using an authentication factor which is disabled fails. In this case, the Access_Event property of the Access Point object where this authentication factor was used is set to the value corresponding to the reason why it was disabled. See Table 12-X+2.

Authentication Factor Disable	**Applicable Access Event**
DISABLED	DENIED_AUTHENTICATION_FACTOR_DISABLED
DISABLED_LOST	DENIED_AUTHENTICATION_FACTOR_LOST
DISABLED_STOLEN	DENIED_AUTHENTICATION_FACTOR_STOLEN
DISABLED_DAMAGED	DENIED_AUTHENTICATION_FACTOR_DAMAGED
DISABLED_DESTROYED	DENIED_AUTHENTICATION_FACTOR_DESTROYED

For multi-factor authentication, all Authentication Factors required need to be in the same Access Credential object. If a particular factor of a multi-factor authentication can also be used independently of the other factors and has access rights which differ from the rest of the factors, then this factor must also be instantiated in a separate Access Credential object.

7.2.1.5 Validity Time Window

An Access Credential may optionally be valid only during a predefined time window. The time window for which the credential is valid is bounded by the activation time, which defines the start of the validity period, and the expiry time, which defines the end of the validity period.

The required property ***Activation_Time***, of type *BACnetDateTime*, indicates the date and time at or after which the Access Credential may be valid. If the current time is before the activation time, the Access Credential is disabled and the value DISABLED_NOT_YET_ACTIVE is added to the Reason_For_Disable list. The value DISABLED_NOT_YET_ACTIVE is removed from the Reason_For_Disable list when this condition no longer applies. If any of the fields of the BACnetDateTime contain "wildcard" values, or this property is not present, then the credential can be used from ‘start of time’.

The required property ***Expiry_Time***, of type *BACnetDateTime*, indicates the date and time from when the Access Credential is expired. This defines the end of the validity period of the Access Credential. If the current time is after the expiry time, the Access Credential is disabled and the value DISABLED_EXPIRED is added to the Reason_For_Disable list. The value DISABLED_EXPIRED is removed from the list when this condition no longer applies. If any of the fields of the BACnetDateTime contain "wildcard" values, or this property is not present, then the credential can be used until ‘end of time’.

7.2.1.6 Disabling an Access Credential

An Access Credential object can be disabled by an operator or by any process. If disabled, the Reason_For_Disable contains a corresponding reason, and Credential_Status becomes INACTIVE. If no disable is set (NONE), the status of the Access Credential is determined according other properties of the object.

The required property ***Credential_Disable***, of type *BACnetAccessCredentialDisable*, allows an operator or external process to disable the Access Credential. When this property takes on any value other than NONE the Access Credential is disabled and the corresponding disable reason is added to the Reason_For_Disable list. When this property is changed any value previously added to the Reason_For_Disable list, as a result of changing this property, is removed from that list.

The following disable enumeration values are known and defined in the new BACnet enumeration *BACnetAccessCredentialDisable*:

NONE	The credential has not been disabled by an operator or external process.
DISABLE	The credential has been disabled for unspecified reasons. The disable-reason value DISABLED shall be added to the Reason_For_Disable property.
DISABLE_MANUAL	The credential has been disabled by a human operator. The disable-reason value DISABLED_MANUAL shall be added to the Reason_For_Disable property.
DISABLE_LOCKOUT	The credential is disabled because has been locked out by an external process. The disable-reason value DISABLED_LOCKOUT shall be added to the Reason_For_Disable property.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values for disabling a credential other than those defined by the BACnet standard [STD]. A disable-reason value shall be added to the Reason_For_Disable property. It is a local matter which disable reason is added. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

7.2.1.7 Use Counting

Access Credentials may be limited in the number of their use for authentication. The uses may be counted based on different criteria.

7.2.1.7.1 Number of days used

An Access Credential can be limited in the number of days it may be used. The days do not need to be contiguous days.

The optional property ***Days_Remaining***, of type *INTEGER*, indicates the number of remaining days for which the credential can be used. If this property has a value greater than zero, its value is decremented by one when the credential this object represents is granted access at an access controlled point, and the current date is more recent than the date indicated in the property Last_Use_Time. If this property becomes zero, the Access Credential is disabled and the value DISABLED_MAX_DAYS is added to the Reason_For_Disable list. The value DISABLED_MAX_DAYS is removed from the Reason_For_Disable property when this property is set to a value greater than zero.

If this property is present, and the credential this object represents is not limited in the days it can be used, the value of this property is -1 and DISABLED_MAX_USES is never added to the Reason_For_Disable property.

If the Days_Remaining property is present, Last_Use_Time is required to be present.

7.2.1.7.2 Number of uses

An Access Credential can be limited in the number of uses for access. Access granted at an Access Point counts as one use. Entry or Exit is both counted as one use of the Access Credential.

The optional property ***Uses_Remaining***, of type *INTEGER*, indicates the number of remaining uses which the credential can be used for authentication. If this property has a value greater than zero, and access is granted at an access controlled point, the value of this property is decremented by one. If the value becomes zero, the Access Credential is disabled and the value DISABLED_MAX_USES is added to the Reason_For_Disable list. The value DISABLED_MAX_USES is removed from the Reason_For_Disable property when this property is set to a value greater than zero.

If this property is present, and the credential this object represents is not limited in the number of uses, the value of this property is -1 and DISABLED_MAX_USES is never added to the Reason_For_Disable property.

7.2.1.7.3 Number of days not used (Inactivity Counter)

An Access Credential may become invalid after a number of days of not being used. This is independent of the Expiry_Time, and is reset each time the Access Credential is used for access.

The optional property ***Absentee_Limit***, of type *Unsigned*, specifies the maximum number of consecutive days for which the credential can remain inactive (i.e. unused) before it becomes disabled. The calculation of inactivity duration is based on the time of last use as indicated by the property Last_Use_Time. If Last_Use_Time does not have a valid time and date then the absentee limit is considered not being exceeded. When the absentee limit is exceeded the Access Credential is disabled and the value DISABLED_INACTIVITY is added to the Reason_For_Disable list. The value DISABLED_INACTIVITY is removed from the list when this condition no longer applies.

7.2.1.8 Access User Relationship

The Access Credential may be assigned to an Access User. This is used to provide the reference to the owning Access User.

The optional property ***Belongs_To***, of type *BACnetDeviceObjectReference*, references an Access User object which represents the owning access user (i.e. person, group, or asset). If this property is present and the credential is not assigned to an access user, this property contains an instance number of 4194303 in the object identifier field and in the device instance part of the device identifier, if present. The determination of whether the credential is valid for authentication, based on the value of this property, is a local matter. If the credential has not been assigned to an access user and the policy of the site requires that it be assigned, then the credential is disabled and the value DISABLED_UNASSIGNED is added to the Reason_For_Disable list. The value DISABLED_UNASSIGNED is removed from the list when this condition no longer applies.

Note that there is a single reference to an Access User. This is by intent, enforcing the use of Access User objects to represent groups owning the same Access Credential.

7.2.1.9 Assigned Access Rights

An Access Credential object refers to the Access Rights assigned. Once the Access Credential is authenticated, the Authentication & Authorization Process uses this information to find the Access Rights assigned to the Access Credential.

The required property ***Assigned_Access_Rights***, of type *BACnetArray of BACnetAssignedAccessRights*, refers to Access Rights objects which define the access rights assigned to this credential.

BACnetAssignedAccessRights is a structure with the following fields:

Assigned-Access-Rights	This field, of type BACnetDeviceObjectReference, refers to the Access Rights objects that define the access rights assigned to this credential. Each object referenced in this field is an Access Rights object. Any entry which references to a non-existent Access Rights object is ignored. If no access rights are specified then this reference contains 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present.
Enable	This field, of type BOOLEAN, specifies whether the access rights specified in the assigned-access-rights field is enabled (TRUE) or not (FALSE) for the credential this object represents.

When this array is increased in size without providing content, new elements are initialized to contain in the Assigned-Access-Rights field, 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present. The Enable field is initialized to FALSE.

7.2.1.10 Tracing and Searching Support

The Access Credential object may refer to the last Access Point where this Access Credential has been recognized. An Access Credential is recognized when one of its Authentication Factors matches the Authentication Factor read from the Credential Reader Interface.

The optional property ***Last_Access_Point*** of type *BACnetDeviceObjectReference* refers to the last Access Point object where at least one of the Authentication Factors of the Access Credential has been recognized. A COV subscription to this property enables monitoring applications to trace and search this Access Credential. These applications, when subscribed, will receive all information required conveyed in the COV notifications. If property level COV is in effect for this property, any update of this property causes a COV notification to be issued, regardless of whether the value of this property changes. If the Access Credential is not yet used, this property contains an instance number of 4194303 in the object identifier and in the device instance part of the device identifier, if present.

To trace on access events created by the Access Credential, the optional property ***Last_Access_Event***, of type *BACnetAccessEvent*, indicates the last access event created on use of the Access Credential. If multiple access events are generated by the use it is a local matter which access event is the Last_Access_Event. If no access event is created yet, this property has a value of NONE.

The optional property ***Last_Use_Time***, of type *BACnetDateTime*, indicates the date and time of the last use of the Access Credential at an Access Point. If the Access Credential is not yet used, this property contains "wildcard" in all fields.

The optional property ***Trace_Flag***, of type *BOOLEAN*, is set to TRUE if the access credential is to be traced. This instructs an Access Point object to set its Access_Event property to TRACE when the Access Credential is recognized.

7.2.1.11 Threat Authority

The Access Credential may be valid for authorization depending on the threat level of an Access Point. If the threat level of the Access Point is higher than the Access Credential’s threat authority, the Access Credential will be denied access at the Access Point.

The optional property ***Threat_Authority***, of type *BACnetAccessThreatLevel*, indicates the maximum threat level for which this credential is valid. If this value is less than the Threat_Level of the Access Point where the Access Credential is used, access is denied. If this property is not present it is assumed that the threat authority of the Access Credential is zero.

7.2.1.12 Credential Special Support

The Access User using this Access Credential may need special support of the PACS to have more time to pass an Access Door. If this is the case, EXTENDED_PULSE_UNLOCK is used to command an Access Door, delaying the activation of a DOOR_OPEN_TOO_LONG alarm for longer time according the Access Door configuration.

The need for special support is indicated by the optional property ***Extended_Time_Enable*** of type *BOOLEAN*. If Extended_Time_Enable is TRUE, EXTENDED_PULSE_UNLOCK is used to command the Access Door, otherwise PULSE_UNLOCK.

It is required that Access Credentials can be exempted from all authorization checks, sometimes known as master exemption.

The master exemption from authorization checks is indicated by the optional property ***Master_Exemption***, of type *BOOLEAN*. Once authenticated, the Access Credential is exempted from all standard authorization checks if Credential_Status is ACTIVE. It is a local matter if the Access Credential is exempted from proprietary authorization checks.

It is required that Access Credentials can be exempted from passback detection.

The exclusion from passback detection is indicated by the optional property ***Passback_Exemption*** of type *BOOLEAN*. If this property is TRUE, the Access Credential is excluded from passback enforcement. The Access Credential is not denied access due to passback violations.

It is required that Access Credentials can be exempted from occupancy enforcements.

The exclusion from occupancy enforcement is indicated by the optional property ***Occupancy_Exemption*** of type *BOOLEAN*. If this property is TRUE, the Access Credential is excluded from occupancy limit enforcement. The occupancy count in the Access Zone object is updated as normal however, the Access Credential is not denied access due to occupancy limit enforcement.

7.2.2 Access User Object Type

The object type Access User is used to represent a single person, an organizational entity or an asset. Relationships among Access Users are supported, to model hierarchical organizations comprising e.g. companies, departments, or groups of any kind, or to model ownership of assets.
The Access User object is not directly involved in Authentication & Authorization. It is used for informational purpose and may hold references to other systems. PACS implementations are not generally required to support this object type, although for some applications it may become useful.

Access User	
Identification:	Name, ID, Global ID, Type, Description, Profile
General Health:	Status Flags, Reliability
Type Indication:	Represented Type
Name:	User Name
User Reference:	User Reference Number
User Information:	User Information Reference
Hierarchy:	Hierarchical Structures
Ownership:	Credential Ownership

Figure 7–6, Access User Object Type

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 3.

7.2.2.1 Identification

These are the standard properties of BACnet objects for object identification, type identification, description, etc.

Since the same Access Users have to be represented in multiple devices, it is recommended that all Access User objects representing the same Access User have the same relevant content. Some content may not be required in all devices.

When a user is represented in multiple devices, the representing Access User objects may not have the same Object_Identifier in each device; however, they may be identified using the Global_Identifier property.

The required property ***Global_Identifier***, of type *Unsigned*, is the global identifier of the user of an access control system this object represents. The global identifier is an internetwork-wide non-zero value which is unique for each user. If a global identifier is assigned to a user, then all Access User objects in all devices which represent this user have the value of this global identifier in this property. A value of zero indicates that no global identifier is assigned.

The synchronization of content is a local matter, but the model supports this requirement through the use of standard BACnet services.

7.2.2.2 General Health

Properties are defined which indicate the general health of the object. Reliability indication is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which may provide more details.

- The IN_ALARM flag is always 0. There is no Event_State property in this object.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDDEN flag is always 0.
- The OUT_OF_SERVICE flag is always 0.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of whether this object is "reliable" as far as the BACnet Device can determine and, if not, why. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

7.2.2.3 Type Indication

The Access User object exposes an enumeration value which indicates what the object represents. The following values are known and defined in the *BACnetAccessUserType* enumeration:

ASSET	The Access User object represents a physical item which may enter and exit a secured zone. Examples are: • Trucks • Cars • Containers • Computers • Baggage • Etc.
GROUP	The Access User object represents a group of Access Users. Various types of groupings of assets or persons are possible. This covers: • Organization • Department • Etc.
PERSON	The Access User object represents an individual person.
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to indicate proprietary user types other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

The represented type is exposed in the required property ***User_Type*** of type *BACnetAccessUserType*, which indicates what the Access User object represents.

7.2.2.4 User Name

The Access User object exposes the clear text name of the person, group or asset it represents. Note that this information is not restricted by uniqueness or other constraints. Content is not restricted, and may even contain multiple lines of printable characters.

The optional property ***User_Name*** of type *CharacterString*, which is the clear text name of the person, asset or group the object represents.

7.2.2.5 User External Identifier

This indicates an external identifier such as a reference number for the person, asset or group the Access User object represents. This identifier is not explicitly pointing to any further information. Example uses are:

- FIPS-201 mandated person reference number
- Social Security Number
- Employee Number
- Inventory Item Number
- Etc.

The user external identifier is exposed by the optional property ***User_External_Identifier***, of type *CharacterString*. While the content is typically unique, interpretation of the content is a local matter.

7.2.2.6 User Information Reference

The Access User object may indicate a user information reference in the form of a Character String. This is a reference such as an URL to information located in other systems, such as an HRMS, CMS or IDMS.

The optional property ***User_Information_Reference***, of type *CharacterString*, holds this reference. Interpretation of the content is a local matter.

7.2.2.7 Hierarchical Structures

The Access User objects may be hierarchically structured, to adapt to organizational structures and assign assets to persons or groups.

An Access User object exposes what other Access Users it contains by an optional property ***Members*** of type *List of BACnetDeviceObjectReference*, referring to Access User objects.

An Access User object exposes to what Access User it belongs to by an optional property ***Member_Of*** of type *List of BACnetDeviceObjectReference*, referring to Access User objects.

7.2.2.8 Credential Ownership

To represent ownership of Access Credentials, the Access User object may have Access Credentials assigned. This information is typically used for operation purposes (e.g. disabling all Access Credentials of a Person).

The credential ownership is exposed by a property ***Credentials***, of type of *List of BACnetDeviceObjectReference*, referring to Access Credential objects. If there are no credentials owned, the list is empty.

7.3 Authorization

Authorization is a series of validation steps before access is granted. The sequence of steps is not mandated. Example validation steps are:

- Authentication and validation of Access Credentials
- Evaluation of external conditions
- Verification by an external system
- Sufficient Access Rights assigned to the Access Credential
- Others

The validation steps may be taken in any order, or run in parallel. A single failed validation step finally denies access, while only if there is a sufficient number of a successful validation steps, access is granted. The process of authorization is to pass all authorization checks without finding a denial reason.

The validation steps are typically performed by the Authentication & Authorization Process.

Some steps may already be performed by the Credential Reader process, but then the Authentication & Authorization Process does not become active since no Authentication Factor is received from the Credential Reader Interface.

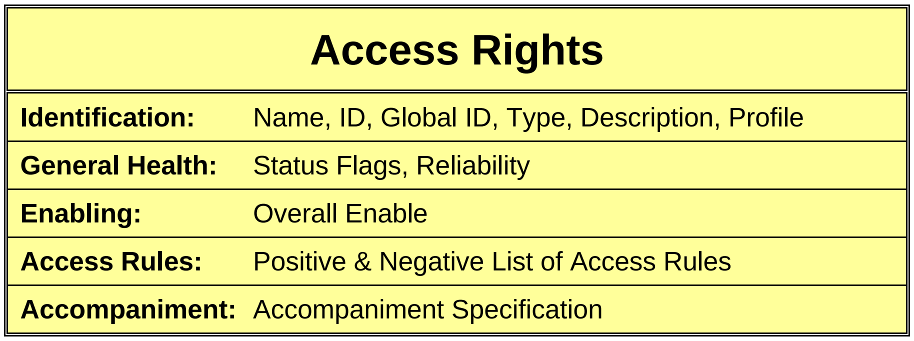
For clarity of the model, the step of authenticating an Access Credential is detailed in section Authentication above. Other validation steps are detailed in the following sections, although proprietary validations may take place and are a local matter of the Authentication & Authorization Process.

7.3.1 Access Rights Object Type

The Access Rights object defines a set of negative and positive access rules. All access rules of all Access Rights objects assigned to an Access Credential are evaluated for authorization. Negative access rules take precedence over positive access rules.
Each access rule specifies when or why access is to be denied or possible at a geographical location. If a negative access rule is found which is enabled and matches location and current time of the request to access, then the access request is finally denied. If an access rule is found which is enabled and matches location and current time of the request to access, then the validation step of checking access rights is successful.

An Access Rights object is referenced from Access Credential objects, which is an assignment of access rights to this Access Credential. Multiple Access Credential objects may refer to the same Access Rights object. Access Rights objects facilitate to model the access rights of "roles", which Access Users may take within an organization.

Access Rights objects may also be used as profiles, e.g. Office Area at office times, Server Room at nightshift, etc.



Access Rights	
Identification:	Name, ID, Global ID, Type, Description, Profile
General Health:	Status Flags, Reliability
Enabling:	Overall Enable
Access Rules:	Positive & Negative List of Access Rules
Accompaniment:	Accompaniment Specification

Figure 7–7, Access Rights Object Type

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 4.

7.3.1.1 Identification

These are the standard properties of BACnet objects for object identification, type identification, description, etc.

Since the same Access Rights have to be represented in multiple devices, and it is recommended that all Access Rights objects representing the same access rights have the same identification and content, the Access Rights objects representing the same set of access rights shall have the same relevant content. Some content may not be required in all devices, e.g. Access Rights for Access Zones and Access Points not controlled by a particular device.

When a specific access rights collection is represented in multiple devices, the representing Access Rights objects may not have the same Object_Identifier in each device, however, they may be identified using the Global_Identifier property.

The required property ***Global_Identifier***, of type *Unsigned*, is the global identifier of the access rights collection this object represents. The global identifier is an internetwork-wide non-zero value which is unique for each access rights collection. If a global identifier is assigned to an access rights collection, then all Access Rights objects in all devices which represent this access rights collection have the value of this global identifier in this property. A value of zero indicates that no global identifier is assigned.

The synchronization of content is a local matter, but the model supports this requirement through the use of standard BACnet services.

7.3.1.2 General Health

Properties are defined which indicate the general health of the object. Reliability indication is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which may provide more details.

- The IN_ALARM flag is always 0. There is no Event_State property in this object.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDDEN flag is always 0.
- The OUT_OF_SERVICE flag is always 0.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of whether this object is "reliable" as far as the BACnet Device can determine and, if not, why. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

7.3.1.3 Overall Enable

The Access Rights object supports the enabling and disabling of all the access rules it contains.

The required property ***Enable***, of type *BOOLEAN*, indicates whether the negative and positive access rules at all specified by the object are valid (TRUE) or not (FALSE). Even if Enable is TRUE, individual access rules may be disabled individually by their individual enable field.

7.3.1.4 Access Rules

The access rules specify access rights (i.e. pass right or “positive” right).

A single access rule contains the following fields:

- The Time-Range-Specifier is used to specify the evaluation of the Time-Range field
- The Time-Range is used to validate the access rule against current time through a Schedule or Calendar object or any other external condition that can be evaluated to TRUE or FALSE.
- The Location-Specifier is used to specify the evaluation of the Location field.
- The Location is used to validate the access rule against the location of authentication, i.e. the Access Point or Access Zone.
- The Enable flag allows individual enabling of the access rule.

The structured type *BACnetAccessRule* is defined as follows:

Time-Range-Specifier	This field is an enumeration that specifies the evaluation of the Time-Range field:

SPECIFIED	Time-Range references a property that will be evaluated to TRUE or FALSE as defined for the Time-Range field.
ALWAYS	The value of the Time-Range field is ignored and always evaluates to TRUE.

Time-Range

This optional field, of type BACnetDeviceObjectPropertyReference, references a property that can be evaluated to TRUE or FALSE, which defines if the rule is valid (TRUE) or not (FALSE).

This field is required to be present if Time-Range-Specifier is SPECIFIED. If the Time-Range-Specifier is ALWAYS, and this field is present, then this field contains 4194303 in the instance part of the object identifier and the device identifier, if present.

If Time-Range-Specifier is SPECIFIED and this field references a property of type other than BOOLEAN, then the following evaluations apply:

If the value of the referenced property is of type Unsigned, a value of zero shall evaluate to FALSE, while any other value shall evaluate to TRUE.

If the value of the referenced property is of type INTEGER, a value of less than or equal to zero shall evaluate to FALSE, while any value greater than zero shall evaluate to TRUE.

If the value of the referenced property is of type BACnetBinaryPV, then INACTIVE shall evaluate to FALSE, while ACTIVE evaluates to TRUE.

If the referenced property does not exist or its value cannot be retrieved, or the value is of type NULL, the Time-Range evaluates to FALSE. This also covers the "uninitialized" case *).

If the reference property is of any other type, then the evaluation is a local matter.

Note: This field can reference a Schedule object Present_Value property for the specification of time ranges.

Location-Specifier

This field is an enumeration that specifies how the Location field is evaluated:

SPECIFIED	Location references a specific Access Point or Access Zone object and is evaluated as specified for the Location field.
ALL	The value of the Location field is ignored and matches any access controlled point or zone.

Location

This optional field, of type BACnetDeviceObjectIdentifier, refers to the Access Point or Access Zone that this access rule is valid for.

This field shall be present if Location-Specifier is SPECIFIED. If Location-Specifier is ALL and this field is present, then this reference contains 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present.

If Location-Specifier is SPECIFIED, then the following evaluations apply:

When Location refers to an Access Point object, this access controlled point is required to be the location where the credential used to request access has been authenticated.

When Location refers to an Access Zone object, the access controlled point where the credential used to request access has been authenticated is required to be an entry point to this zone.

When Location refers to an object that does not exist, Location evaluates to FALSE. This also covers the "uninitialized" case. *)

Enable

This field, of type BOOLEAN, specifies whether this rule is enabled (TRUE) or not (FALSE).

The access rules authorization check is performed on all the access rules assigned to a credential, i.e. on all access rules specified by the Access Rights objects the respective Access Credential object references.

An individual access rule evaluates to TRUE if Time-Range evaluates to TRUE and Location evaluates to TRUE. If the Enable flag is FALSE, the access rule is not considered at all in this authorization check.

All the negative access rules of all the Access Rights objects referenced by the respective Access Credential object are evaluated before the positive access rules. If any enabled negative rule evaluates to TRUE, then this authorization check fails and access is denied. In this case, the Access_Event property of the Access Point object is set to DENIED_POINT_NO_ACCESS_RIGHTS, if the negative rule prohibits access through the access point, or DENIED_ZONE_NO_ACCESS_RIGHTS if the negative rule prohibits access to the zone.

If no negative access rule evaluates to TRUE, then the positive access rules of all the Access Rights objects referenced by the respective Access Credential object are evaluated. When the first enabled positive access rule is found that evaluates to TRUE, then this authorization check succeeds. Access may subsequently be denied or granted based on other authorization checks, such as accompaniment requirement etc. If all positive access rules of all the Access Rights objects referenced by the access credential evaluate to FALSE, then this authorization check fails. In this case, if the credential has access through this access point or to the access zone at a different time, then the Access_Event property of the Access Point object is set to DENIED_OUT_OF_TIME_RANGE. Otherwise, the Access_Event property is to DENIED_NO_ACCESS_RIGHTS.

If the respective Access Credential object has a master exemption, then this authorization check is not performed and always considered successful.

The negative access rules are exposed by the required property ***Negative_Access_Rules***, of type *BACnetArray of BACnetAccessRule*.

If the size of the Negative_Access_Rules array is increased without entry values being provided, then the new array entries are initialized as follows *):

- Time-Range-Specifier is initialized to SPECIFIED,
- Time-Range reference is initialized to contain 4194303 in the instance part of the object identifier and the device identifier, if present,
- Location-Specifier is initialized to contain SPECIFIED,
- Location is initialized to contain 4194303 in the instance part of the object identifier and the device identifier, if present,
- Enable is initialized to TRUE.

The positive access rules are specified by the required property ***Positive_Access_Rules***, of type *BACnetArray of BACnetAccessRule*.

If the size of the Positive_Access_Rules array is increased without entry values being provided, then the new array entries are initialized as follows *):

- Time-Range-Specifier is initialized to SPECIFIED,
- Time-Range reference is initialized to contain 4194303 in the instance part of the object identifier and the device identifier, if present,
- Location-Specifier is initialized to contain SPECIFIED,
- Location is initialized to contain 4194303 in the instance part of the object identifier and the device identifier, if present,
- Enable is initialized to TRUE.

This initialization makes the evaluation of the rule to result in FALSE in both the positive and negative rules list.

*) Note that Addendum j [ADJ] has an error in the respective section. It refers to specifier enumerations that have been removed. The above approach for uninitialized entries was considered reasonable by the author of this document, but will be subject of public review comments and according resolution.

7.3.1.5 Accompaniment Specification

The Access Rules of an Access Rights object may be evaluated successfully only if the Access Credential in question is accompanied by another Access Credential that is presented at the same Access Point and has valid access rights for this Access Point. If the access rights evaluate to FALSE anyway for the credential that requires accompaniment, access is denied. This is configurable by a reference to either another Access Rights object, to an Access Credential object or to an Access User object.

If the accompanying Access Credential is not presented within the amount of time, specified by the Accompaniment_Time property of the Access Point, then the authorization of the original credential will fail. If this time is not specified then the amount of time to wait for the accompanying credential is a local matter. When the expected accompaniment is not received within the timeout set by the Access Point property Accompaniment_Time, the Access_Event property of the Access Point object is set to DENIED_NO_ACCOMPANIMENT.

The optional property ***Accompaniment***, of type *BACnetDeviceObjectReference*, refers to the object the accompanying Access Credential needs to be related to as follows.

The accompaniment criteria are specified as follows:

(a) If this property refers to an Access Rights object then the accompanying Access Credential is required to refer to that Access Rights object.
(b) If this property refers to an Access Credential object then this object is required to represent the accompanying Access Credential.
(c) If this property refers to an Access User object then this object is required to represent the Access User which owns the accompanying Access Credential.

If the property is present and contains 4194303 in the instance part of the object identifier and in the device instance part of the device identifier, if present, or the property is absent, the Access Rules of this Access Rights object are evaluated without any accompanying Access Credential.

If an accompanying credential is presented, whether valid or not, the Access_Event property of the Access Point object is set to ACCOMPANIMENT_BY. If an invalid accompaniment is provided then the Access_Event property of the Access Point object is subsequently set to DENIED_INCORRECT_ACCOMPANIMENT.

7.3.2 Configuration and Validation of Access Rights

The validation of Access Rights is one of the validation steps in authorization. Access Rights allow specifying, for an Access Credential:

- Where is access possible
- When (or under what condition) is access possible
- Does this credential need to be accompanied

The evaluation of access rules is performed by the Authentication & Authorization Process. Although this is a local matter, the interface's data model reflects some behavior and allows configuring Access Rights.

The data model for Access Rights facilitates configurations according the Role Based Access Control (RBAC) model introduced by NIST (see [RBAC]) and used widely by the IT industry.

For the sake of simplicity, no dedicated role representation is introduced. The new Access Rights object may represent roles in the sense of RBAC, and privileges are assigned to the role by specifying Access Rules in these Access Rights objects. Multiple Access Credential objects may have the same Access Rights object assigned. By this these Access Credentials have the role assigned which is represented by the Access Rights object.

7.3.2.1 Access Rights Configuration Overview

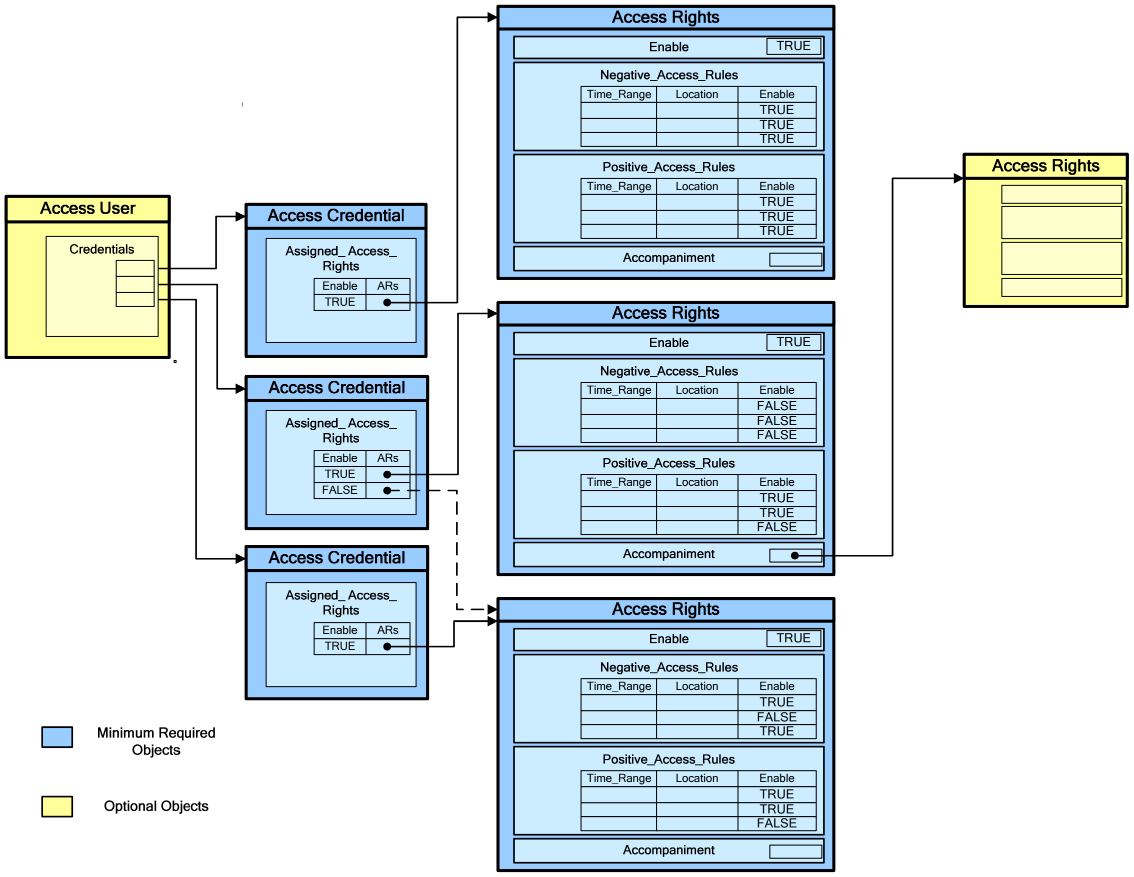


Figure 7–8, Access Rights Configuration Data Structure Overview and Example

- An Access User object refers to the Access Credential objects it owns (Property *Credentials*).
- An Access Credential object refers to Access Rights objects (Property *Assigned_Access_Rights*).
- Access Rights objects contain a list of negative access rules where each defines a time range and a location (Property *Negative_Access_Rules*)
- Access Rights objects contain a list of positive access rules where each defines a time range and a location (Property *Positive_Access_Rules*)
- Access Rights objects may specify an accompaniment requirement (Property *Accompaniment*)

For details of the Access Rights object see the Access Rights Object Type definition above.

7.3.2.2 Access Rights Validation

The validation of Access Rights starts with the Access Credential(s) authenticated at an Access Point. The Access Credential object refers to the Access Rights objects assigned, specifying the access rules for this Access Credential. The validation of accompaniment, if specified, requires that the Authentication & Authorization Process collects other Authentication Factors to authenticate the accompanying Access User's Access Credential.

The validation check is done by first iterating through the Negative_Access_Rules properties of all the Access Rights objects assigned to the Access Credential. Each negative access rule is evaluated. If any negative access rule evaluates to TRUE, access is denied.

The second step is to iterate through the Positive_Access_Rules properties of all the Access Rights objects assigned to the Access Credential. Each positive access rule is evaluated. The first successful validation results, if Accompaniment requirements are satisfied, in a successful validation check of Access Rights.

For accompaniment requirements see the Accompaniment property.

Access may still be denied due to other reasons (e. g. passback violation, exhausted use count, occupancy limits etc.). If there is no successful validation of Access Rights at all, access is finally denied.

7.3.3 Time Ranges

Access rules of Access Rights objects may refer to a time range specification, specifying when in time the rule is valid. The standard BACnet object types Schedule and Calendar are reused for this functionality. The Time-Range field of an access rule refers to a Schedule object's Present_Value, which indicates TRUE (i.e. valid) or FALSE (i.e. invalid). If the Schedule object did not yet set the Present_Value by a scheduled action, the Present_Value contains NULL, which evaluates to FALSE as well.

Schedules and Calendar objects allow defining sophisticated time ranges consisting of

- Weekly schedules
- Exception schedules
- Effective periods
- Calendar for exception schedules

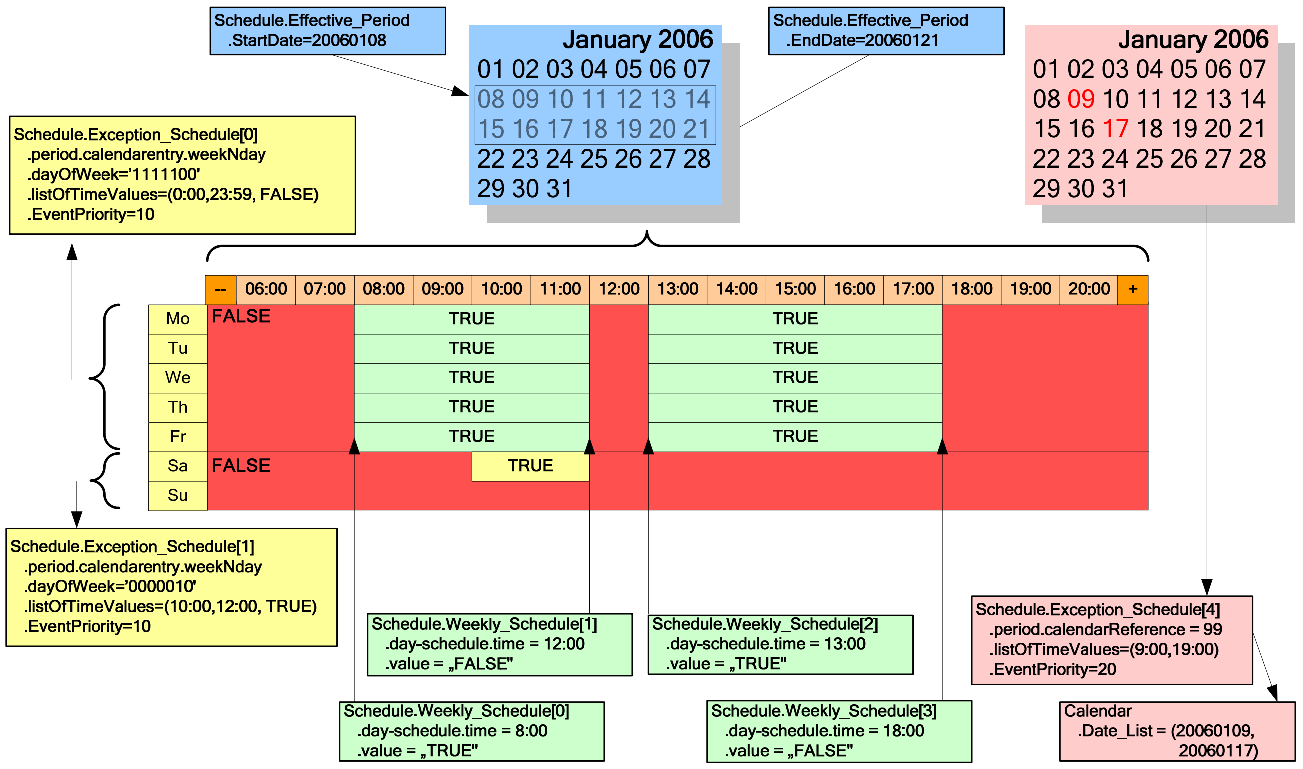

Schedule.Effective_Period .StartDate=20060108
January 2006
01 02 03 04 05 06 07
08 09 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
Schedule.Effective_Period .EndDate=20060121
January 2006
01 02 03 04 05 06 07
08 09 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
Schedule.Exception_Schedule[0] .period.calendarentry.weekNday .dayOfWeek='1111100' .listOfTimeValues=(0:00,23:59, FALSE) .EventPriority=10
-- 06:00 07:00 08:00 09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 +
Mo Tu We Th Fr Sa Su
FALSE
TRUE
FALSE
TRUE
Schedule.Exception_Schedule[1] .period.calendarentry.weekNday .dayOfWeek='0000010' .listOfTimeValues=(10:00,12:00, TRUE) .EventPriority=10
Schedule.Weekly_Schedule[1] .day-schedule.time = 12:00 .value = „FALSE"
Schedule.Weekly_Schedule[2] .day-schedule.time = 13:00 .value = „TRUE"
Schedule.Exception_Schedule[4] .period.calendarReference = 99 .listOfTimeValues=(9:00,19:00) .EventPriority=20
Schedule.Weekly_Schedule[0] .day-schedule.time = 8:00 .value = „TRUE"
Schedule.Weekly_Schedule[3] .day-schedule.time = 18:00 .value = „FALSE"
Calendar .Date_List = (20060109, 20060117)

Figure 7–9, Time Range Example

The Schedule object writes the BOOLEAN values FALSE or TRUE into its own Present_Value. No property references are required in the Schedule object. The Schedule object is not required to perform write operations to other objects.

For an access rule, a Schedule object's Present_Value of TRUE evaluates an access rule's Time-Range to TRUE, a value of FALSE (or NULL) evaluates to FALSE.

For Time Ranges not related to time of day, but for entire special days of a year, the Time-Range field of an access rule may point to a Calendar object's Present_Value, a BOOLEAN value, directly.

7.3.4 External Conditions

Access rules of Access Rights objects may refer to an external condition, specifying if the access rule is valid. The Time-Range field of an access rule refers to a numeric discrete value property of a defined subset of primitive data types. If the referenced property evaluates to TRUE, the Time-Range field evaluates to TRUE, otherwise Time-Range evaluates to FALSE.

7.3.5 Two-Person-Rules

"Two-Person-Rule" as a term covers a number of special authorizations which involve multiple Access Users, or Access Credentials. Supported rules are:

Minimum Occupancy: This is supported by the Access Zone object. The Authentication & Authorization Process may read and evaluate further Authentication Factors before granting access.

Accompanied persons or assets: The Access Rights object specifies what other Access Credential or Access User must be authenticated before access is granted. See the Accompaniment property of the Access Rights object above.

Group rules: Since Access Rights may be shared among different Access Credentials, and by this forming some kind of group, the Accompaniment property of the Access Rights object may point to shared Access Rights, indicating what other group member is required.

Other rules may be supported by the model as defined, but its application by the Authentication & Authorization Process is a local matter.

More sophisticated rules are considered a local matter of the Authentication & Authorization Process and proprietary extensions of the data model. If such rules turn out to be common, the standard model may be extended.

8. Credential Reader Interface

This interface provides access to authentication factors read, validated and processed. It enables control over indication hardware at the credential reader's front plate. It may provide access to data stored in credentials, such as smart card blocks.
The data model and services provided by this interface are BACnet based. The interface is provided and maintained by the Credential Reader Process.

The Credential Reader Interface provides the following:

- Authentication Factors read, processed and validated by the Credential Reader Process
- False reading indications
- Access to Credential contents, such as files or blocks on smart card
- Access to Credential Reader front plate elements, such as LEDs, beepers etc.
- Credential Reader States

The Credential Reader Interface is modeled using existing and new BACnet objects. This creates the freedom to model simple single factor credential readers up to very sophisticated multi-factor readers with sophisticated front plate elements. The new Credential Data Input object type is defined to represent input of Authentication Factors. This object basically provides processed and validated Authentication Factors to any interested client role process.

The Authentication & Authorization Process of the PACS uses this interface to get processed and validated authentication factors presented by the access user at the reader's front plate, and to control the indication elements of the reader's front plate. Other processes may also use this interface, e.g. Time & Attendance applications to get timestamps.

8.1 Authentication Factors

The Credential Reader Interface enables access to Authentication Factors read at the Credential Reader front plate and preprocessed by the Credential Reader Process.

8.1.1 Structured Authentication Factors

Some authentication factors are structured, of which single elements or the entire structure may be used as Authentication Factors. Examples for such structures are magnetic stripe codes containing a site code and a number, or the US government Personal Identity Verification (PIV) initiative based Card Holder Unique Identifier (CHUID) structure, which contains the Federal Agency Smart Card Number (FASC-N) data structure.

To use such structured information in any way, the Credential Reader Process is required to know about such structures, to be able to extract elements and to provide them as a structured or simple authentication factor at the Credential Reader Interface. Any subset is considered a unique format.

If different subsets or formats of an authentication factor are supported, parallel access to the different formats is supported at the Credential Reader Interface by providing the according set of Credential Data Input objects.

8.1.2 Authentication Factor Data Model

The data model used for Authentication Factors is based on structured data of type *BACnetAuthenticationFactor*. Any kind of Authentication Factor is represented as a BACnetAuthenticationFactor structure. This includes, for example, Wiegand numbers, FASC-N numbers, Hash Codes from biometric data etc. The model needs to support the extremely broad range of different Authentication Factors today and increasing variety in the future. Vendors may define custom authentication factor formats.

Up to now there is no standard available which would define how any Authentication Factors looks like in bits & bytes, useful for BACnet. But for interoperability, it is essential that the content and encoding of Authentication Factors is defined. Since no standard is available, BACnet defines the structure and encoding of well known Authentication Factors, while allowing vendor or organization proprietary definitions and encodings.

The concept of site specific Format Class identifiers is added for differentiation of authentication factors conveyed by different Access Credential types, where equal authentication factor values may be possible, but need to be differentiated in authentication and authorization.

Note: The same data type for Authentication Factors (*BACnetAuthenticationFactor*) is used to hold the Authentication Factors in the Access Credential object, introduced for the Authentication & Authorization Interface.

The defined data type for Authentication Factors is *BACnetAuthenticationFactor*, a sequence with the following fields:

Format-Type	This field, of type BACnetAuthenticationFactorType, specifies the internal representation of the authentication factor value in the Value field. If there is no current value available, this field takes on the value UNDEFINED, and the Value field is empty. If an authentication factor value is present that contains errors or that cannot be interpreted as the specified type, this field takes on the value ERROR. The possible enumeration values are described below. The Value field contains error details.
Format-Class	This field, of type Unsigned, is a site specific value that identifies the class of the authentication factor. This field is used in sites where different physical credentials are used that have the same authentication factor format type. The format class value is used to differentiate between the different credentials which may have equal authentication factor values. A value of zero is used as the default where no differentiation of credentials is required. When Format-Type is UNDEFINED, Format_Class has a value of zero.
Value	This field, of type OCTET STRING, holds the authentication factor value data. The encoding of this value is specified in the Format-Type field and described below.

Due to the wide variety of authentication factor formats, a specific BACnet ASN.1 encoding of the Value field for each format is not practical. In addition, because of the vast variety in the size and structure of the authentication factor formats, a single common structure that defines all different formats is considered too complex and therefore not feasible. Devices would be required to know of any possible format, and would not be capable of generically processing unknown formats. Therefore, the Value field is defined to be an OCTET STRING, with encoded content as defined as follows below. Note that the Authentication Factor Type is used to determine the respective encoding. All clause references within the below table refer clauses in the BACnet standard [STD].

The encoding of the authentication factor Value field is defined in Addendum *j* to BACnet 2004 [ADJ] part 8 (proposed normative Annex X to the BACnet Standard [STD]).

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]
UNDEFINED	Undefined – no authentication factor value is specified	Octet String Size = 0

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]
ERROR	Error – this is used when the authentication factor value is not the value expected, or could not be interpreted as expected.	Octet String Size = n Octet [1] = error reason, as follows: 0 = Unspecific error 1 = Parity failure 2 = Too few data 3 = Too much data 4 = Incomplete read 128..255 = Any proprietary error reason Octet[2..3] = authentication factor format type expected (if unknown or cannot be determined use UNDEFINED) Octet[4..n] = data array that holds erroneous data

CUSTOM	Custom (proprietary, or industry standard) format – each format specified is identified by the vendor ID and the proprietary format ID	Octet String Size = n
		Octet[1..2] = BACnet vendor-id (i.e., unsigned 16)
		Octet[3..4] = proprietary type id (i.e., unsigned 16)
		Octet[5..n] = data array that holds proprietary format

SIMPLE_NUMBER16	Simple unsigned number with range [0 .. 65535]	Octet String Size = 2, Octet[1..2] = number (i.e., unsigned 16 bit number)
SIMPLE_NUMBER32	Simple unsigned number with range [0 .. 4294967295]	Octet String Size = 4, Octet[1..4] = number (i.e., unsigned 32 bit number)
SIMPLE_NUMBER56	Simple unsigned number with range [0 .. 72057594037927935] Typically used for DESFire card Serial Numbers	Octet String Size = 7, Octet[1..7] = number (i.e., unsigned 56 bit number)
SIMPLE_ALPHA_ NUMERIC	Simple alpha numeric string	Octet String Size = n, Octet[1] = length of character string in octets including character set specifier (max 255) Octet[2] = character set specifier (as specified in 20.2.9 excluding DBCS, i.e. a value of X'01') Octet[3..n] = string of characters (encoded as specified in 20.2.9)
ABA_TRACK2	Magnetic stripe card format (BCD[2] format) as developed by the banking industry (ABA).	Octet String Size = 15, Octet[1.. 10 (MS nibble)] = primary account number (19 digits) Octet[10 (LS nibble) – 12(MS nibble)] = 4 digit expiration date in form "MMYY" Octet[12 (LS nibble)..13] = 3 digit service code Octet[14.. 15] = discretionary data (4 digits)

WIEGAND26	Standard 26 bit Wiegand format as defined by SIA standard (SIA AC-01).It is separated into facility code and card number.	Octet String Size = 3
		Octet[1] = facility-code (i.e., unsigned 8 bit number)
		Octet[2..3] = card-number (i.e., unsigned 16 bit number)

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]
WIEGAND37	37 bit Wiegand format with a 35 bit card number. (HID 37 bit format. – H10302)	Octet String Size = 5 Octet[1..5] = card-number (i.e., unsigned 40 bit number with range (0..34359738367))
WIEGAND37_FACILITY	37 bit Wiegand format with a 16 bit facility code and 19 bit card number. (HID 37 bit format with facility code. – H10304)	Octet String Size = 5 Octet[1..2] = facility-code (i.e., unsigned 16 bit number) Octet[3..5] = card-number (i.e., unsigned 24 bit number with range (0..524287))

FACILITY16_CARD32	Non-standard Wiegand variants that have 32 bit card number and 16 bit facility code formats.	Octet String Size = 6
		Octet[1..2] = facility-code (i.e., unsigned 16 bit number)
		Octet[3..6] = card-number (i.e., unsigned 32 bit number)

FACILITY32_CARD32	Non-standard Wiegand variants that have 32 bit card number and 32 bit facility code formats.	Octet String Size = 8
		Octet[1..4] = facility-code (i.e., unsigned 32 bit number)
		Octet[5..8] = card-number (i.e., unsigned 32 bit number)

FASC_N	Federal Agency Smart Credential – number.
	Includes only agency code, system code and credential number.
		Octet String Size = 8
		Octet[1..2] = agency-code (i.e., unsigned 16 bit number)
		Octet[3..4] = system-site code (i.e., unsigned 16 bit number)
		Octet[5..8] = credential number (i.e., unsigned 32 bit number)
		-- refer to NIST technical implementation Guidance document for more details

FASC_N_BCD	Federal Agency Smart Credential – number (BCD[2] format)
	Includes only agency code, system code and credential number.
		Octet String Size = 7
		Octet[1..2] = agency-code (4 digit BCD number)
		Octet[3..4] = system-site code (4 digit BCD number)
		Octet[5..7] = credential number (6 digit BCD number)
		-- refer to NIST technical implementation Guidance document for more details

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]

FASC_N_LARGE	Federal Agency Smart Credential – number.
	Includes all FASC-N data fields excluding start sentinel, end sentinel, field separators and LRC.
		Octet String Size = 19
		Octet[1..2] = agency code (i.e., unsigned 16 bit number)
		Octet[3..4] = system/site code (i.e., unsigned 16 bit number)
		Octet[5..8] = credential number (i.e., unsigned 32 bit number)
		Octet[9] = series code (i.e., unsigned 8 bit number)
		Octet[10] = credential code (i.e., unsigned 8 bit number)
		Octet[11..15] = person identifier (i.e., Unsigned 40 bit number)
		Octet[16] = organizational category (i.e., unsigned 8 bit number)
		Octet[17..18] = organizational identifier (i.e., unsigned 16 bit number)
		Octet[19] = association category (i.e., unsigned 8 bit number)
		-- refer to NIST technical implementation Guidance document for more details

FASC_N_LARGE_BCD	Federal Agency Smart Credential – number. (BCD[2] format) Includes all FASC-N data fields excluding start sentinel, end sentinel, field separators and LRC.	Octet String Size = 16 Octet[1..2] = agency-code (4 digit BCD number) Octet[3..4] = system-site code (4 digit BCD number) Octet[5..7] = credential number (6 digit BCD number) Octet[8 (MS nibble)] = series code (1 digit BCD number) Octet[8 (LS nibble)] = credential code (1 digit BCD number) Octet[9..13] = credential number (10 digit BCD number) Octet[14 (MS nibble)] = organizational category (1 digit BCD number) Octet[14 (LS nibble)..16(MS nibble)] = organizational identifier (4 digit BCD number) Octet[16 (LS nibble)] = association category (1 digit BCD number) -- refer to NIST technical implementation Guidance document for more details

GSA75	GSA 75 bit (FASC-N plus expiry date)	Octet String Size = 12
		Octet[1..2] = agency-code (i.e., unsigned 16 bit number)
		Octet[3..4] = system-site code (i.e., unsigned 16 bit number)
		Octet[5..8] = credential number (i.e., unsigned 32 bit number)
		Octet[9..12] = expiry date (4 octets encoded as specified in Clause 20.2.12)

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]
GUID	Global unique identifier represented as IPv6 address	Octet String Size = 16 -- Refer to RFC 2373 for format description and encoding
CHUID	Card Holder Unique Identifier (CHUID), without Asymmetric Key and without Authentication Key MAP. See SP 800-73 Section 1.8.3 (Figure 1 & 2 pg 12 of the TIG 2.3)	Octet String Size = 45 Octet[1..8] = FASC-N as specified in FASC_N Octet[9..12] = agency code (4 ANSI.X3.4 characters as defined in SP 800-73 (Section 6.4, p. 34, of the TIG 2.3) Octet[13..16] = organization identifier (4 ANSI.X3.4 characters as defined in SP 800-73 (Section 6.4, p. 34, of the TIG 2.3) Octet[17..25] = DUNS number (9 ANSI.X3.4 numeric characters as defined in SP 800-73 (Figures 1 & 2 of the TIG 2.3) Octet[26..41] = GUID (IPv6 address as defined in SP 800-73 (Figures 1 & 2 of the TIG 2.3) Octet[42..45] = Expiry Date expiry date (4 octets encoded as specified in Clause 20.2.12)
CHUID_FULL	Complete Card Holder Unique Identifier stored as data string. The data elements are decoded using the CHUID tags which are embedded in the data string. See SP 800-73 Section 1.8.3 (Figure 1 & 2 pg 12 of the TIG 2.3)	Octet String Size = n (maximum size = 3397) Octet[1..n] = CHUID data string -- Octet encoding is defined in SP 800-73 (Figure 1 & 2 of the TIG 2.3) using CHUID Tags.
GUID	Global unique identifier represented as IPv6 address	Octet String Size = 16 -- Refer to RFC 2373 for format description and encoding
CBEFF_A	Common Biometric Exchange File Format (CBEFF) Patron format A	Octet String Size = n Octet[1..n] = CBEFF data -- NIST CBEFF Patron Format A (CBEFF) content formatted
CBEFF_B	Common Biometric Exchange File Format (CBEFF) Patron format B	Octet String Size = n Octet[1..n] = CBEFF data -- NIST CBEFF Patron Format B (BioAPI) content formatted
CBEFF_C	Common Biometric Exchange File Format (CBEFF) Patron format C	Octet String Size = n Octet[1..n] = CBEFF data -- NIST CBEFF Patron Format C (ANSI Standard X9.84) content formatted

Format Type **(BACnetAuthentication-FactorType)**	**Authentication Factor Format Description**	**Authentication Factor Value Encoding**[1]
USER_PASSWORD	User name and password	Octet String Size = n, Octet[1] = length of user name string in octets including character set specifier (max 255) Octet[2] = character set for user name string (as specified in 20.2.9 excluding DBCS, i.e. a value of X’01’) Octet[3..m] = string of characters for user name(encoded as specified in Clause 20.2.9) Octet[m+1] = length of password string in octets including character set specifier (max 255) Octet[m+2] = character set password string (as specified in 20.2.9 excluding DBCS, i.e. a value of X’01’) Octet[m+3..n] = string of characters for password (encoded as specified in Clause 20.2.9)

[1] Multi-octet fields are conveyed with the most significant octet first.
[2] In BCD (binary coded decimal) format each octet holds two 4-bit BCD encoded decimal digits. Bits 7 to 4 convey the most significant digit, while Bits 3 to 0 convey the least significant digit.

8.1.3 Credential Data Input Object Type

The Credential Data Input object type is introduced to model the input of Authentication Factors into the PACS, as the key model of the Credential Reader Interface. Input in the name is chosen to be consistent with other BACnet object types which represent the input of information from the physical world into the system, such as the Binary Input object type.

A single physical credential reader which supports multiple authentication factor formats may be represented by multiple Credential Data Input objects when the authentication factor formats are not functionally equivalent or cannot be used interchangeably. An example of a device of this type is a credential reader which contains both a card and biometric reader. In this case a specific Credential Input object would be used for both the card reader and the biometric reader functions.

Alternatively, a single physical credential reader which supports multiple authentication factor formats may be represented by a single Credential Data Input object when the authentication factor formats are functionally equivalent and may be used interchangeably. An example of a device of this type is a credential reader which can read multiple Wiegand formats. It is recommended that a single Credential Data Input object which supports multiple authentication factor formats be associated with a single physical device.

A key for a reasonable set of Credential Data Input objects is the support of reasonable Authentication Policies of Access Point objects.

Credential Data Input	
Identification:	Name, ID, Type, Description, Profile
Present_Value:	Last authentication factor provided including status
Format Types:	Supported Format Types Specification
General Health:	Overall Status and Reliability
Update:	Update information for COV reporting, Read Status
Simulation:	Out of service support for simulation

Figure 8–1, Credential Data Input Object Type

This new object type is defined in Addendum *j* to BACnet 2004 [ADJ] part 6.

8.1.3.1 Identification

The common properties for object identification are used as mandated and optional by the BACnet standard [STD]. This includes a numeric identifier, object name, textual description and a profile reference.

8.1.3.2 Present Value

The key value of the Credential Data Input object type provides the authentication factors read at the Credential Reader front plate and preprocessed, as a value of type BACnetAuthenticationFactor. This value contains a format type identifier (*Format-Type*), a site specific format class identifier (*Format-Class*) and the authentication factor value (*Value*).
The last BACnetAuthenticationFactor provided is exposed by the ***Present_Value*** property of type *BACnetAuthenticationFactor*. If no authentication factor was provided yet, the Value field of BACnetAuthenticationFactor uses format type UNDEFINED, with no octets in the Value field.

The *BACnetAuthenticationFactor* structure has three fields, which are used by the Present_Value property as follows:

Format-Type	This field, of type BACnetAuthenticationFactorType, specifies the format of the authentication factor value in the Value field. The value of this field is one of the format types specified in the Supported_Formats property. If there is no current authentication factor value read by this object, this field shall take on the value UNDEFINED. In addition, if this field contains a value which is not specified in the Supported_Formats property, such as after a modification to the Supported_Formats property or after the Out_Of_Service property changes from TRUE to FALSE, then this field shall take on the value UNDEFINED. If an authentication factor is read that contains errors or that cannot be interpreted as one of the specified format types, this field shall take on the value ERROR.
Format-Class	This field, of type Unsigned, contains the value specified in the Supported_Format_Classes array field which corresponds to the authentication format type in the Format-Type field. If the Supported_Format_Classes property is not present, this field always has a value of zero. If Format-Type has a value of UNDEFINED, then this field has a value of zero.
Value	This field, of type OCTET STRING, holds the authentication factor value data. The encoding of this value is specified in the Format-Type field and defined as outlined in the Authentication Factor Data Model section above. See also the proposed Annex *X* in Addendum *j* to BACnet 2004 [ADJ] part 8.

COV subscriptions on this object will result in notifications sent out each time the Present_Value is updated. This is also the case if the same Authentication Factor is provided again.

The Present_Value is writable when Out_Of_Service is TRUE. When writing, the Format-Type field value must match one of the values of the Supported_Formats property or be UNDEFINED, the Format-Class field must match the value of the corresponding element of the Supported_Format_Classes property, and the Value field must be encoded according the Format-Type field value.

The Credential Data Input object type exposes what formats of Authentication Factors it provides.

The required property ***Supported_Formats***, of type *BACnetARRAY of BACnetAuthenticationFactorFormat*, is used to specify which authentication factor formats are supported by the object. The structure of an element of this array has three fields which are defined as follows:

Format-Type	This field, of type BACnetAuthenticationFactorType, specifies a supported authentication factor format type.
Vendor-ID	This optional field, of type Unsigned16, is required when the Format-Type field has a value of CUSTOM. It contains the BACnet vendor identifier of the vendor which defined the custom format. This value may differ from the Vendor_Identifier property value in the Device object, which identifies the device manufacturer. If the Format-Type field does not have a value of CUSTOM and this field is present it has a value of zero.
Vendor-Format	This optional field of type Unsigned16 is required when the Format-Type field has a value of CUSTOM. It contains a unique identifier which identifies a specific custom authentication factor format as defined by the BACnet vendor in the Vendor-ID field. If the Format-Type field does not have a value of CUSTOM and this field is present it has a value of zero.

The array size of this property must be equal to the array size of the Supported_Format_Classes property. If the size of the Supported_Formats array is increased without entry values being provided, the new array entries are initialized with the Format-Type having a value of UNDEFINED. If Vendor-ID and Vendor-Format are present, they are initialized with a value of zero.

The optional property ***Supported_Format_Classes***, of type *BACnetARRAY of Unsigned*, specifies the values which the Format-Class field of the Present_Value may take on. The value of the i^{th} element of this array shall be used when an authentication factor is read that is of the format defined in the i^{th} element of the Supported_Formats array.

This property is used to distinguish between multiple different supported authentication factor formats, used on a site, of which two or more use the same authentication factor format type and may have colliding value ranges. A value of zero is used as the default where no differentiation is required. Otherwise, the value is site specific and can be any non-zero value.

The array size of this property must be equal to the array size of the Supported_Formats property. If the size of the Supported_Format_Classes array is increased without entry values being provided, the new array entries are initialized with a value of zero.

8.1.3.3 General Health

Properties are defined which indicate the general health of the object. Reliability indication and out of service is supported.

The required property ***Status_Flags***, of type *BACnetStatusFlags*, indicates, by a set of individual flags (i.e. bits), the general health of the object. Each flag is related to specific properties, which provide more details.

- The IN_ALARM flag is always 0. There is no Event_State property in this object.
- The FAULT flag is 1 if the property Reliability has a value other than NO_FAULT_DETECTED.
- The OVERRIDEN flag is always 0.
- The OUT_OF_SERVICE flag is 1 if the property Out_Of_Service is TRUE.

The required property ***Reliability***, of type *BACnetReliability*, indicates the detailed reliability of the credential reader processing this object represents. The following enumeration values may be supported:

NO_FAULT_DETECTED	The process and the object are reliable.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
OPEN_LOOP	Any sensor or communication wiring has an open circuit condition
SHORTED_LOOP	Any sensor or communication wiring has a short circuit condition
PROCESS_ERROR	A processing error was encountered, preventing the process to properly maintain the object.
COMMUNICATION_FAILURE	Proper operation of the object is dependant on communication with a remote sensor or device and communication with the remote sensor or device has been lost.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

The Reliability property is writable when Out_Of_Service is TRUE.

8.1.3.4 Update Information and Read Status

Any consumer of Authentication Factors is required to get any new Authentication Factor preprocessed by the Credential Reader Process. Since COV reporting in BACnet was not designed for streaming, but no new BACnet services shall be introduced for PACS, some additional information is required in properties of the Credential Data Input object type, as well as a specific use of COV reporting.

In COV reporting as defined by BACnet, a notification is sent out on subscription or on re-subscription without a change of the Present_Value. On the other hand, COV notifications are sent on change of a value or change of Status_Flags. If the value is updated by the same value, no COV notification is issued to the subscribers. This is insufficient for the Credential Data Input object, since the same credential may subsequently be used to request access.

The required behavior is as follows:

- The subscriber needs to be able to distinguish whether it received a COV notification due to an update of Present_Value, COV (re-)subscription, or change of Status_Flags.
- The subscriber needs to receive a COV notification whenever Present_Value is updated, regardless whether the new value is different from the last value.
- The subscriber needs to be able to distinguish between valid readings and false readings reported.

To achieve this behavior, an additional property is introduced, which is a BACnetTimeStamp, indicating the last update of Present_Value. This property is considered as the criterion to generate COV notifications, and will always have a change when Present_Value is updated.
The value of this property is conveyed in the COV notification, aside Present_Value and Status_Flags. It allows a client to verify if it got an update of Present_Value, or the COV notification was sent due to (re-)subscription or Status_Flags change.

The required property ***Update_Time***, of type *BACnetTimeStamp*, holds the time stamp of the last update of Present_Value. This property serves as the criterion to generate COV notifications, and is conveyed in COV notifications from this object. To overcome COV startup issues, this property changes its value on each update of the Present_Value property.

The client's behavior can be sketched as follows:

(Re-)Subscription:

1: Read and cache Update_Time from the object in question.
2: Subscribe for COV notifications from the object

Operation:

3: Receive COV notifications. If the COV notification conveys the same last update time stamp as read from the object or cached from last notification, a COV notification caused by the subscription or change of Status_Flags was received.
4: If the COV notification conveys another last update time stamp, the COV notification was caused by a real update of Present_Value. Act on it and cache the last update time stamp.
5: Continue at step 3

If no update has yet occurred, update times of type Time or Date have a "wildcard" (X'FF') value in each octet, and Sequence number update times have the value 0.

8.1.3.5 Simulation and Out of Service

The Credential Data Input object supports Out-Of-Service for simulation purposes. In Out-Of-Service, the Credential Reader Process does not feed authentication factor readings. Writing authentication factors to Present_Value allows simulating reactions of the PACS on simulated authentication factor readings.

The required property ***Out_Of_Service***, of type *BOOLEAN*, is an indication whether (TRUE) or not (FALSE) the Present_Value is prevented from being modified by the Credential Reader Process local to the BACnet device in which the object resides. When Out_Of_Service is TRUE, the Present_Value and Reliability properties may be written.

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 respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred in the input.

8.2 Credential Content Access

The Credential Reader Interface may make Credential content accessible. This is data other than Authentication Factors which are available on an Access Credential. Although this is out of scope of PACS, the Credential Reader Interface allows modeling such content and making it accessible to any process through BACnet. The use of smart cards as one of the examples below does not exclude any other type of sophisticated credential which may hold other data as Authentication Factors.

Some design ideas using existing and upcoming BACnet object types are:

- Use File Objects to model smart card blocks
- Use Octet String Value objects to model smart card blocks
- Use BOOLEAN Value objects to model flags present in a smart card
- Use Value objects to model credits
- Etc.

8.3 Credential Reader Front Plate Elements Access

A Credential Reader process may expose front plate elements at the Credential Reader Interface. For modeling such elements, BACnet provides a rich set of object types. There are various sets of elements present, and every vendor has some unique features. A certain model is therefore always related to a specific product. The following are just examples of possible model elements.
There are some new object types in discussion within the SSPC-135, to make any value of primitive data type available as an object. This includes Character String Value, Integer Value, Bit String Value etc.

- LED Indicators may be made controllable from external processes through Binary Output objects.
- LCD display lines may be made visible through Character String Value objects*.
- Graphical Display control may be made accessible through a collection of objects.
- Raw magnetic stripe reader head data may be made visible through specific Credential Data Input objects.
- Etc.

The following table gives some example modeling possibilities:

Element	**Data Type**	**May be modeled as**
LED indicators	Binary value	• Binary Output object • Binary Value object • Multistate Value object if different ON modes, such as blinking in various frequencies or multiple colors. • Property of a proprietary object
Text Display Line	Character String	• Character String Value object* • Property of a proprietary object
Graphical Display & Graphical Display Elements	<any>	• Character String Value objects* • Binary Value Objects for binary indicators • Unsigned Value objects* for icons • Proprietary objects and properties • Etc.
Raw Magnetic Stripe data	Credential Data Input	• Credential Data Input object using UNDEFINED option • Credential Data Input object using a vendor specific option
Raw Key Input	Unsigned Character String	• Unsigned Value object* • Character String Value object* • Proprietary objects and properties
Etc.		

* There are some new object types in discussion within the SSPC-135, to make any value of primitive data type available as an object. This includes Character String Value, Integer Value, Bit String Value, etc. The Character String Value object is not yet a standard object type.

8.4 Credential Reader States

Credential Reader states may be provided at the Credential Reader Interface in several ways. Base for this are either BACnet objects used for other purposes, or objects specifically instantiated to indicate status.

- Binary Output objects for LED indicators could indicate fault of LED indicator
- Binary Input object may represent tamper contacts, and going into alarm if activated
- Tamper contacts may be made visible and issuing tamper alarms through Binary Input or Life Safety Point objects.
- Etc.

The following table gives some overview over modeling possibilities:

Element	**Data Type**	**May be modeled as**
LED Fault	Reliability	• Reliability of Binary Output object • Reliability of Binary Value object • Any Reliability property of a proprietary object
Tamper contact fault	Reliability	• Reliability of Binary Input object • Reliability of Binary Value object • Any Reliability property of a proprietary object
Tamper	BACnetLifeSafetyState	• Life Safety Point object
Tamper Contact	Binary Value	• Binary Input object

8.5 Example Credential Reader Models

Some sample models of credential readers are outlined. Such models are realized by the Credential Reader Process and exposed at the Credential Reader Interface.

In the object examples, the core values are mentioned only. The full-fledged objects will be according the object definitions as given by the BACnet standard [STD] or Addendum *j* [ADJ]. For properties and values, as well as value notation, refer to the BACnet standard [STD], clause 12 and annex D.

8.5.1 Simple Reader

This example simple credential reader has the following features:

- 26 Bit Wiegand Authentication Factor from magnetic stripe or contact-less (proxy).
- Red and green bicolor LED. Red is always on, but switched to green when access is granted.

This type of reader may be modeled as outlined in the following subsections.

8.5.1.1 Authentication Factor

The sample property values shown (in bold) reflect the situation after reading a standard Wiegand card.

Credential Data Input, Instance 1	Object_Name	Allows to identify the object by name: **“Door5/WiegandReader”**
	Present_Value	Provides the Wiegand value as a BACnetAuthenticationFactor value according the definition for structured Wiegand-26 factors. No Format Class defined: **(WIEGAND26, 0, 'X'896C72)** (Facility Code = 137 = X'89'), Card Number = ,27762 = X'6C72')
	Format_Type	**WIEGAND26**

8.5.1.2 LED switching to green

The sample property values shown (in bold) reflect the situation when the LED is switched to green.

Binary Value, Instance 1	Object_Name	Allows to identify the object by name: **“Door5/ReaderSignal”**
	Present_Value	Indicates whether the LED is switched to green (ACTIVE) or not: **ACTIVE**

8.5.2 Multi-Factor Reader

This example multi-factor reader has the following features:

- PIV Card FASC-N number and PIN Authentication Factors
- A red and a green LED as indicators
- A buzzer
- A tamper switch, which may initiate reader tamper alarm.

This type of reader may be modeled as outlined in the following subsections.

8.5.2.1 Authentication Factors

The sample property values shown (in bold) reflect the situation after reading a PIV card.

Credential Data Input, Instance 5	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-PIV-FASC-N”**
	Present_Value	Provides the FASC-N value as a BACnetAuthenticationFactor value. No Format-Class defined: **(FASC_N, 0, X'04D2162E00018ACA')** Agency Code = 1234 = X'04D2' System-Site Code = 5678 = X'162E' Credential Number = 101066 = X'00018ACA'
	Format_Type	**FASC_N**

The sample property values shown (in bold) reflect the situation after reading a PIN. The PIN is represented as an authentication factor of type SIMPLE_ALPHA_NUMERIC.

Credential Data Input, Instance 6	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-PIN”**
	Present_Value	Provides the PIN value as a BACnetAuthenticationFactor value. The Value field is an ANSI X3.4 encoded character string. Format Class is 100: **(SIMPLE_ALPHA_NUMERIC, 100, X'003132343536')** String = "123456"
	Format_Type	**SIMPLE_ALPHA_NUMERIC**

8.5.2.2 LEDs

The sample property values shown (in bold) reflect the situation when the red LED is switched on.

Binary Output, Instance 1	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-LED-Red”**
	Present_Value	Indicates whether the red LED is switched on (ACTIVE) or not (INACTIVE): **ACTIVE**

The sample property values shown (in bold) reflect the situation when the green LED is switched off.

Binary Output, Instance 2	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-LED-Green”**
	Present_Value	Indicates whether the green LED is switched on (ACTIVE) or not (INACTIVE): **INACTIVE**

8.5.2.3 Buzzer

The sample property values shown (in bold) reflect the situation when the buzzer is off.

Binary Output, Instance 3	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-Buzzer”**
	Present_Value	Indicates whether the buzzer is switched on (ACTIVE) or not (INACTIVE): **INACTIVE**

8.5.2.4 Tamper Switch

The sample property values shown (in bold) reflect the situation when the tamper is off, not indicating tamper. The object notifies an alarm if Present_Value goes to active. Other properties required for intrinsic reporting are not shown.

Binary Input, Instance 1	Object_Name	Allows to identify the object by name: **“NIST-Gate-3-Left-Reader-Tamper”**
	Present_Value	Indicates whether the tamper switch is active or not: **INACTIVE**
	Event_State	Indicates the current event state of the tamper switch: **NORMAL**
	Alarm_Value	Indicates what Present_Value is interpreted as OFF-NORMAL, causing an OFF-NORMAL (i.e. tamper alarm) event state taken: **ACTIVE**
	Event_Enable	Indicates which event state transitions are notified. Here, the transitions TO-OFFNORMAL and TO-NORMAL are notified: **{TRUE,FALSE,TRUE}**

8.5.3 Smart Card Reader supporting two way communication

This example is not completely worked out. Only the additional features are shown as a sample model. The basic reader functionality may be as outlined above for the multi-factor reader. Of course, the smart card is required to be in the reader to communicate information from and to the smart card.

Additional features supported by the reader device are:

- Reader firmware download
- Accessing smart card content
- Sophisticated indicators and display
- PKI
- Hash code support

These features may be modeled as outlined in the following subsections.

8.5.3.1 Reader Firmware Download

The Credential Reader Process enables the download of firmware into the reader device by exposing a File Object at the Credential Reader Interface. New firmware is downloaded into the reader by writing the firmware code into the File Object, using the Atomic-Write-File service. Upload of firmware may be supported by using Atomic-Read-File service.

The sample property values shown reflect the situation after firmware has been downloaded, or reflects the actual firmware present in the reader device.

`File, Instance 1`	Object_Name	Allows to identify the object by name: **`“NIST-Gate-3-Left-Reader-Firmware”`**
	File_Type	Indicates what type of file this object represents: **`“Firmware”`**
	File_Size	Indicates the size of the file / firmware in octets **`148765`**
	Modification_Date	Indicates when file content has last been modified (i.e. downloaded): **`(5-APR-2006, 08:30:49.0)`**
	File_Access_Method	Indicates what file access method is possible on this file: **`STREAM_ACCESS`**

As a possibility, the new firmware may be activated using the ReinitializeDevice BACnet service.

8.5.3.2 Smart Card Content Access

The model outlined assumes the smart card content is organized as blocks of memory, or files. Each block can be modeled as a File Object. Only one example File Object is shown.
Using the File Object allows both reading and writing to the block. The exact content of the block is not specifically modeled, although descriptive properties of the object may indicate what is contained in the block.
Note that communicating with the card itself is a local matter of the Credential Reader Process, and not exposed at the Credential Reader Interface. The File Object creates some façade to hide this from the client role process.

The sample property values reflect a PKI Certificate residing in block 5 of the smart card.

`File, Instance 105`	Object_Name	Allows to identify the object by name: **`“NIST-Gate-3-Left-Reader-Smartcard-Block-5”`**
	File_Type	Indicates what type of file this object represents: **`“PKI Certificate”`**
	File_Size	Indicates the size of the file in octets **`5567`**
	Modification_Date	Indicates when file content has last been modified (i.e. downloaded): **`(5-APR-2006, 08:30:49.0)`**
	File_Access_Method	Indicates what file access method is possible on this file: **`STREAM_ACCESS`**

8.5.3.3 Sophisticated Indicators and Display

Simple indicator model examples have been given in other model examples above.

Modeling hints for more sophisticated indicators and displays are given in section Credential Reader Front Plate Elements Access section above.

For readability of the document, detailed object examples are not outlined here.

8.5.3.4 PKI

The use of Public Key Infrastructure (PKI) for authentication factor validation is expected to be performed by the Credential Reader Process while preprocessing and validating authentication factors.

PKI information such as certificates and keys can be made accessible at the Credential Reader Interface using File Objects etc. See the example object given for Smart Card Content Access above.

8.5.3.5 Hash Code Support

The verification of hash codes and the application of hashing algorithms for authentication factor validation is expected to be performed by the Credential Reader Process while preprocessing and validating Authentication Factors.

Hash codes may be made accessible at the Credential Reader Interface by using various BACnet objects. They may be used for any purpose, but such codes are typically not relevant with Authentication Factors used by client role processes using the Credential Reader Interface.

9. Access Door Interface

The Access Door Interface is provided by the Access Door Process, which is in charge to coordinate and control the individual and specific pieces of a door.
For control, the process knows about how to unlock the door or how to lock it. For events, the door contact can determine whether the door is open or closed but it can't determine whether the door has been forced open or whether the door has been open too long. Likewise, the door lock can determine if it is unlocked but it can't know whether it is unlocked due to a valid card swipe or due to a schedule turning on. While the action is the same, the reporting of the event will be different.

The Access Door Interface provides to client processes a uniform way to control and monitor doors, and may provide detailed information about the physical components of doors. The data model and services provided by this interface are BACnet based.

The Access Door Interface includes:

- An abstract door model suitable for any type of door, as a uniform interface for processes to control and monitor a door. From the PACS perspective, the Authentication & Authorization process is the primary user of this model to control and monitor a door.
- A door equipment elements model, to provide access to single elements comprised in a door, such as locks, deadbolts, contacts, motion detectors, etc.

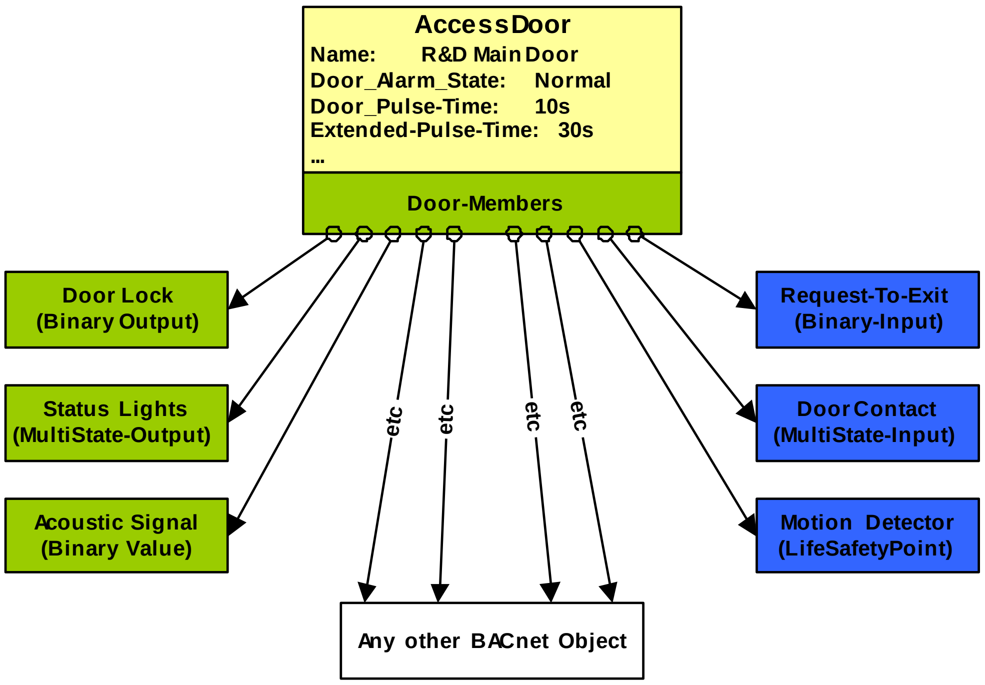

AccessDoor
Name: R&D Main Door
Door_Alarm_State: Normal
Door_Pulse-Time: 10s
Extended-Pulse-Time: 30s
...
Door-Members
Door Lock (Binary Output)
Status Lights (MultiState-Output)
Acoustic Signal (Binary Value)
Request-To-Exit (Binary-Input)
Door Contact (MultiState-Input)
Motion Detector (LifeSafetyPoint)
etc
etc
etc
etc
Any other BACnet Object

Figure 9–1, The Access Door Interface as a collection of standard BACnet objects

9.1 Abstract Door Model

The abstract door model represents any type of door in a uniform way. This enables client processes to control and monitor any door in a generic way. The abstract door model allows both commanding and parameterizing a door for control; and retrieving status information for monitoring.

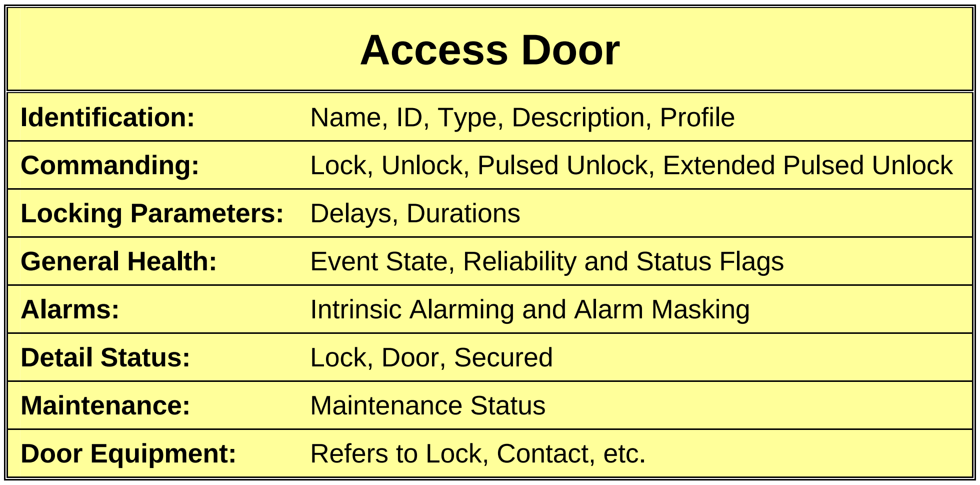
- Commanding: Allows a door to lock, to unlock, to temporarily unlock, or to temporarily unlock with extended time for disabled people's access.
- Parameterizing: Setting various timeouts, and event reporting parameters
- Monitoring: Provides status of the door leaf, the lock, and an overall secured status of the door, reports events of the door on an abstract level.

Additionally, it may indicate what specific physical door hardware elements belong to the door.

The abstract door model is realized by the Access Door object type.

9.1.1 Access Door Object Type

A BACnet object of type Access Door represents the combined physical characteristics of a physical access-controlled door in an abstract way.
It has a relationship to all the physical door hardware and devices that are commonly associated with a door such as a door contact, door lock, Request-To-Exit, etc. Standard BACnet objects may be used to represent the individual components of the physical door hardware, and these objects themselves may be exposed to the outside world. Alternatively, a vendor may wish to not expose, for example, which binary output controls the lock. In fact, the vendor may wish to hide that binary output object completely from the outside world. The Access Door object allows the vendor to do this by having a central point of control for the physical door hardware, and not exposing the relationship to the hardware elements.



Access Door	
Identification:	Name, ID, Type, Description, Profile
Commanding:	Lock, Unlock, Pulsed Unlock, Extended Pulsed Unlock
Locking Parameters:	Delays, Durations
General Health:	Event State, Reliability and Status Flags
Alarms:	Intrinsic Alarming and Alarm Masking
Detail Status:	Lock, Door, Secured
Maintenance:	Maintenance Status
Door Equipment:	Refers to Lock, Contact, etc.

Figure 9–2, Access Door Object Type

This object type is defined in Addendum *f* to BACnet 2004 [ADF].

9.1.1.1 Commanding

As an Access Door object could be controlled from multiple sources, such as from the Authentication & Authorization Process when a person uses a valid credential, a schedule to allow free access during the day, a schedule to lock-down the door after hours, the manual operator or through other algorithmic control, the Access Door object is required to have a priority array. The priority array is used to arbitrate conflicting commands coming from different sources.

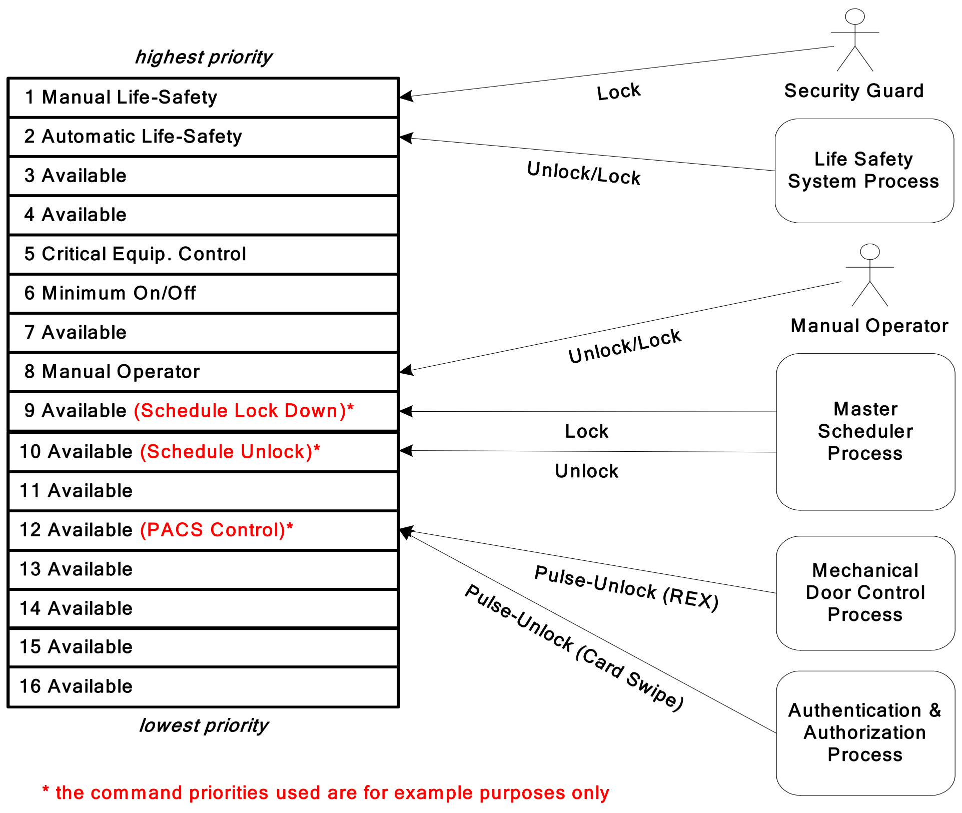


Figure 9–3, Example Access Door Priority Array

In this example, competing commanding entities have been assigned available priorities based on the specific application. Note that priority assignment is site dependant and therefore is not specified in the BACnet standard.

Because this object represents a physical door it has a Present Value that allows locking or unlocking the door, which is the basic control function in physical access control. In addition to commanding the door to unlock or lock for an indefinite period of time, this object has the ability to “pulse unlock” the door for a specified period of time.

The following commands are known, and defined in the new enumeration *BACnetDoorValue*. Proprietary extensions are not foreseen.

LOCK	The door is commanded to the locked state
UNLOCK	The door is commanded to the unlocked state.
PULSE_UNLOCK	The door will be commanded to the unlocked state for a maximum of Door_Pulse_Time time, after which the value will be automatically relinquished from the priority array at the commanded priority. It is permissible for the Access Door Process to relinquish the value from the priority array before the Door_Pulse_Time time has expired. The conditions when this may occur are considered a local matter. If a value of PULSE-UNLOCK is written at a given priority and the Present_Value is currently being commanded, at any value, at a higher priority then the lower priority value will be relinquished immediately.
EXTENDED_PULSE_UNLOCK	The door will be commanded to the unlocked state for a maximum of Door_Extended_Pulse_Time time, after which the value will be automatically relinquished from the priority array at the commanded priority. It is permissible for the Access Door Process to relinquish the value from the priority array before the Door_Extended_Pulse_Time time has expired. The conditions when this may occur are considered a local matter. If a value of EXTENDED-PULSE-UNLOCK is written at a given priority and the Present_Value is currently being commanded, at any value, at a higher priority then the lower priority value will be relinquished immediately.

The PULSE_UNLOCK is essentially a temporal version of the UNLOCK value, as it will be automatically relinquished, at the commanded priority, after the pulse unlock time has expired. The PULSE_UNLOCK (and EXTENDED_PULSE_UNLOCK) values are typically used by the Authentication & Authorization Process when a valid credential is presented to gain access to the door for a limited period of time.

The PULSE_UNLOCK (and EXTENDED_PULSE_UNLOCK) self-relinquishing behavior, although unusual, is necessary because there exist situations where the commanding process cannot relinquish an UNLOCK command. For example, the pulse time for certain applications may be too short or too precise for accurate control from a remote device. Also, the Access Door may be commanded at the same priority by more than one process. When this is the case, the commanding process cannot know whether another process has subsequently commanded the door to unlock. Finally, some situations require that the door relocks immediately after the door has been opened rather than relock when the pulse time expires. An object commanding the door may not have this information.

The commanding is done through writing the property ***Present_Value***, which is of type *BACnetDoorValue*. Command prioritization requires having the properties ***Relinquish_Default*** of type *BACnetDoorValue* and ***Priority_Array*** of type *BACnetPriorityArray*. Their use is according the BACnet standard [STD] clause 19.2.

Note that the Present_Value represents the commanded state of the door, which does not necessarily correspond to the physical state of the door or door lock.

The Present Value of the Access Door object is defined for a standard access controlled door, where the control operation is to lock or unlock. However, this does not exclude motorized devices such as sliding doors, parking gates, etc where the operation is to open or close. In these cases, LOCK shall be equivalent to closing the door and UNLOCK shall be equivalent to opening the door.

9.1.1.2 Locking Parameters

The Access Door object contains parameters used for locking and unlocking.

The maximum duration of time for how long a door is unlocked for a PULSE_UNLOCK command.
This duration is exposed by the optional property ***Door_Pulse_Time***, of type *Unsigned*, which holds the number of tenths of seconds.

The maximum duration of time for how long a door is unlocked for an EXTENDED_PULSE_UNLOCK command.
This duration is exposed by the optional property ***Door_Extended_Pulse_Time***, of type *Unsigned*, which holds the number of tenths of seconds.

The duration of time the physical door lock will delay unlocking when the Present_Value changes to a value of PULSE_UNLOCK or EXTENDED_PULSE_UNLOCK.
This duration is exposed by the optional property ***Door_Unlock_Delay_Time***, of type *Unsigned*, which holds the number of tenths of seconds to delay unlocking.

9.1.1.3 General Health

The property ***Status_Flags***, of type *BACnetStatusFlags*, represents four Boolean flags that indicate the general "health" of the physical door. 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:

- The IN_ALARM flag is logically FALSE (0) if the Event_State property has a value of NORMAL, otherwise logical TRUE (1).
- The FAULT flag is logically TRUE (1) if the Reliability property does not have a value of NO_FAULT_DETECTED, otherwise logical FALSE (0).
- The OVERRIDDEN flag is logically TRUE (1) if the object has been overridden by some mechanism local to the BACnet Device. In this context “overridden” is taken to mean that the physical door is no longer tracking changes to the Present_Value property and the Reliability property is no longer a reflection of the reliability of the physical inputs(s) and output(s). Otherwise, the value is logical FALSE (0).
- The OUT_OF_SERVICE flag is logically TRUE (1) if the Out_Of_Service property has a value of TRUE, otherwise logical FALSE (0).

The property ***Event_State***, of type *BACnetEventState*, indicates the event state associated with the object. The following event states are supported:

NORMAL	The door has no active alarm or fault state. Door_Alarm_State has a value which is neither in Alarm_Values nor in Fault_Values.
OFFNORMAL	Door_Alarm_State has a value which is listed in Alarm_Values.
FAULT	Door_Alarm_State has a value which is listed in Fault_Values or Reliability is not NO_FAULT_DETECTED.

The property ***Reliability***, of type *BACnetReliability*, provides an indication of whether the Present_Value, Door_Alarm_State or the operation of the physical inputs or outputs which comprise this door are "reliable" as far as the BACnet device can determine and, if not, why. This property is writable when Out_Of_Service is TRUE.The Reliability property for this object may have any of the following values:

NO_FAULT_DETECTED	The process and the object are reliable.
MULTI_STATE_FAULT	Door_Alarm_State has a value which is listed in the Fault_Values property.
CONFIGURATION_ERROR	The configuration of the device or object has an error preventing reliable processing.
UNRELIABLE_OTHER	An unspecific reason leads to unreliable processing.

The property ***Out_Of_Service***, of type *BOOLEAN*, is an indication whether (TRUE) or not (FALSE) the logical door which this object represents is not in service. This means that the Present_Value property is decoupled from the physical door and will not track changes to the physical door 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 door when Out_Of_Service is TRUE. While the Out_Of_Service property is TRUE, the Present_Value and Reliability properties, and if present, the Door_Status, Lock_Status and Door_Alarm_State 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, and if present the Door_Status, Lock_Status and Door_Alarm_State properties, shall respond to changes made to these properties while Out_Of_Service is TRUE, as if those changes had occurred to the physical door.

9.1.1.4 Intrinsic Alarming

The Access Door object uses the CHANGE_OF_STATE alarm algorithm and standard BACnet alarming properties for intrinsic event reporting. The Access Door provides overall alarming on an abstract level. Specific door equipment may provide specific alarming if such equipment is exposed by specific BACnet objects.

However, alarm generation for the Access Door object is different than other BACnet objects, as the alarm is not generated by monitoring the Present_Value property but rather a dedicated alarm state property Door_Alarm_State. The sole purpose of this property is to be a trigger for generating alarms. How the value of this property is set is considered a local matter of the Access Door Process.

A new *BACnetDoorAlarmState* enumeration is defined for door specific alarms such as Door-Open-Too-Long, Forced-Open, Tamper conditions, etc.

NORMAL	The door is in normal condition.
ALARM	An unspecific alarm has been detected.
DOOR_OPEN_TOO_LONG	The door was left open for too long time. This condition occurs when the Present_Value has a value of LOCK and one of the following conditions exist: • The Present_Value had a previous value of PULSE_UNLOCK and the door has been in a continual open state for Door_Open_Too_Long_Time tenths of seconds after the Door_Pulse_Time has expired. • The Present_Value had a previous value of EXTENDED_PULSE_UNLOCK and the door has been in a continual open state for Door_Extended_Open_Too_Long_Time in tenths of seconds after the Door_Extended_Pulse_Time has expired. • The Present_Value had a previous value of UNLOCK and the door has been in a continual open state for Door_Open_Too_Long_Time tenths of seconds
FORCED_OPEN	The door was forced open.
TAMPER	Any door equipment was tampered with.
DOOR_FAULT	Any fault was detected.
LOCK_DOWN	The door is locked down.
FREE_ACCESS	The door allows free access.
EGRESS_OPEN	The door was opened due to egress (exit).
<Proprietary Enum Values>	A vendor may use other proprietary enumeration values to indicate alarm states other than those defined by the BACnet standard [STD]. For proprietary extensions of this enumeration, see clause 23.1 of the BACnet standard [STD].

The optional property ***Door_Alarm_State***, of type *BACnetDoorAlarmState*, holds the current alarm state of the Access Door. This property is required to be present when intrinsic reporting is supported by the object. This property, when present, is writable when Out_Of_Service is TRUE.

The optional property ***Alarm_Values***, a list of *BACnetDoorAlarmState* values, holds the alarm states which cause a TO_OFFNORMAL transition.

The optional property ***Fault_Values***, also a *List of BACnetDoorAlarmState* values, holds the alarm states which cause a TO_FAULT transition.

For the DOOR_OPEN_TOO_LONG state, a delay parameter property may be exposed by the Access Door object. This is the time to delay before setting the Door_Alarm_State to DOOR_OPEN_TOO_LONG after it is determined that a door was open for a too long time.
The delay time is exposed by the property ***Door_Open_Too_Long_Time***, of type *Unsigned*, which holds the number of tenths of seconds to delay the DOOR_OPEN_TOO_LONG alarm state.

If intrinsic reporting is supported, the Access Door object has a set of standard properties related to event notification and acknowledgment:

The optional property ***Time_Delay***, of type *Unsigned*, either specifies the minimum period of time in seconds that the Door_Alarm_State must remain equal to any one of the values in the Alarm_Values property before a TO-OFFNORMAL event is generated or remain not equal to any of the values in the Alarm_Values property before a TO-NORMAL event is generated.

The optional property ***Notification_Class***, of type *Unsigned*, specifies 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.

The optional property ***Event_Enable***, of type *BACnetEventTransitionBits*, allows specifying which event state transitions are reported.

The optional property ***Acked_Transitions***, of type *BACnetEventTransitionBits*, indicates acknowledgement of reported event state transitions.

The optional property ***Notify_Type***, of type *BACnetNotifyType*, specifies whether the events are notified as ALARM or EVENT. The notify type is a classification of notifications typically used in client role processes. It has no effect on the server role process behavior.

The optional property ***Event_Time_Stamps***, an *BACnetArray of BACnetTimeStamp*, holds the time stamp as conveyed in the most recent notifications for the individual event state transitions.

9.1.1.5 Alarm Masking

The standard PACS concept of "alarm masking", which is used to temporarily prevent specific types of alarms from being generated while some external condition exists, is introduced to BACnet in the Access Door object.

When a specific alarm state is masked, the access door object is prevented from taking an alarm state while that state is in the optional property ***Masked_Alarm_Values*** list. This optional property is a list of *BACnetDoorAlarmState* values.

To understand how alarm masking is typically used in the access control industry, consider the following example:

Many access-controlled doors have a card reader or other authentication device to restrict access when going through the door in one direction but allow free egress (exit) when going through the door in the opposite direction. To be able to detect and report a FORCED_OPEN condition, the access control system needs to be able to determine when the door can legitimately be in the OPEN state. When going through the door in the direction where access is checked this determination is obvious since a person must present a valid credential for the door to be unlocked and then be opened. If going through the door in the opposite direction is achieved by manual egress, such as turning the door handle, then the door is not unlocked but yet the door is opened. To distinguish between this situation and a FORCED_OPEN condition a motion detector or other occupancy detector is used to determine that the door was opened from the secure side of the door and therefore was legitimately opened. In this case, the motion detector will prevent (mask) from taking the FORCED_OPEN state, so the Access Door object will not generate this alarm.

The alarm masking functionality also allows specific types of states to be masked based on a recurring basis for specific time intervals without having to modify the original Alarm_Values list. For example, it is common practice in sites to mask DOOR_OPEN_TOO_LONG alarms for certain doors during regular office hours or mask FORCED_OPEN alarms when the associated security zone is disarmed to prevent nuisance alarms.

Functionally, alarm masking could be achieved in the Access Door object without making it externally visible since the determination of when to generate an alarm can be done internally by the local controller. However, it was decided to include this feature because this is a standard mechanism within the physical access control industry.
In addition, the alarm masking concept is required for the standardized definition for the Secured_Status property, which gives an indication of whether the physical door is in a secured state.

9.1.1.6 Detail Status

The Access Door object may expose detail status of key elements of a door:

- Door Status
- Lock Status
- Secured Status

There is no intrinsic event reporting of such detailed states foreseen. Event Enrollments using CHANGE_OF_STATE algorithm may be applied for this.

The Door Status indicates the actual state of the door leaf. It is exposed by an optional property ***Door_Status***, which holds a value of the new enumeration *BACnetDoorStatus*. This property, when present, is writable when Out_Of_Service is TRUE. The enumeration is defined as follows:

CLOSED	The door leaf is closed.
OPENED	The door leaf is opened.
UNKNOWN	The status of the door leaf is unknown.

The Lock Status indicates the monitored (as opposed to the commanded) status of the lock of the door. It is exposed by an optional property ***Lock_Status***, which holds a value of the new enumeration *BACnetLockStatus*. This property, when present, is writable when Out_Of_Service is TRUE. The enumeration is defined as follows:

LOCKED	The lock is locked.
UNLOCKED	The lock is unlocked.
FAULT	The lock status input associated with the door lock is unreliable.
UNKNOWN	There is no lock status input associated with the door therefore the status of the physical lock is unknown.

The Secured Status indicates the overall secured status of the door. It is exposed by an optional property ***Secured_Status***, which holds a value of the new enumeration *BACnetDoorSecuredStatus*. If this property is present then the Door_Status property must be present. This enumeration is defined as follows:

SECURED	This state is taken if, and only if all of the following conditions are met: • the IN_ALARM flag of the Status_Flags property is FALSE, and • the Masked_Alarm_Values list, if it exists, is empty, and • the Door_Status property has a value of CLOSED, and • the Present_Value property has a value of LOCK, and • the Lock_Status property, if it exists, has a value of LOCKED or UNKNOWN.
UNSECURED	The door is not in a secured state. One or more of the conditions for SECURED are not met.
UNKNOWN	The secured status is unknown.

9.1.1.7 Maintenance Status

The Access Door may expose a maintenance status. This indicates whether the door has detected the need for maintenance.

The maintenance status is exposed by the optional property ***Maintenance_Required***, which is of type *BACnetMaintenance*. This enumeration and property is already defined, and is reused from the Life Safety Point object.

9.1.1.8 Door Equipment Elements

The Access Door object may expose its relationship to BACnet objects representing specific door equipment such as I/O devices, authentication devices, schedules, programs, or other objects that are associated with the physical door.

The relationship is exposed by the optional property ***Door_Members***, of type *List of BACnetDeviceObjectReference* to BACnet objects associated with the door.

It is a local matter as to how this array is used and which objects are referenced in this array. The array may be empty or not present if the vendor does not wish to expose the individual objects that make up this physical door.

9.2 Door Equipment Elements Model

The door equipment details model is used to model single elements comprised in a door, such as locks, deadbolts, contacts, motion detectors, etc.

The concrete model varies among type of the door, sophistication of the door, vendor’s discretion, etc. There is no standardized model for this.

For any element it is expected that the existing set of BACnet object types is sufficient for modeling such elements. Some possible representations are:

Lock Control	Binary Output
Lock State	Binary or Multi-State Input
Door Contact	Binary or Multistate Input
Motion Detector	Binary or Multistate Input
REX Button	Binary or Multistate Input
Drive	Analog Output

An Access Door object may refer to such objects in its Door_Members property.

10. Event Reporting and Logging

Event reporting and logging are key features of a PACS. The PACS reports events from the equipment it controls, and what it decides when authenticating and authorizing users for access.

10.1 Event Reporting

The PACS reports various events to clients. Clients may be PACS server devices performing overall logging, workstations or other operator UIs. Possible events are:

- Access transactions and alarms, such as access denied or grant decisions
- Occupancy state of access zones
- Door alarms and faults
- Credential Reader alarms and faults

There are classes of events which may be reported, and maybe handled differently:

- Authentication and authorization alarms requiring operator attention
- Authentication and authorization transactions to be logged
- Occupancy states requiring operator attention or being logged
- Equipment alarms requiring operator attention, such as door alarms or credential reader tampers
- Equipment faults requiring operator attention
- Etc.

PACS event reporting is entirely based on the standard BACnet event reporting, logging and acknowledgement mechanisms. All objects used for modeling a PACS define their own intrinsic event reporting by using standard BACnet event reporting mechanisms. Event Enrollment objects may be supported for more sophisticated event reporting.

With exception of the ACCESS_EVENT algorithm used by the Access Point object for intrinsic reporting, existing BACnet event algorithms are applied.
Event reporting may either follow BACnet’s intrinsic event reporting or algorithmic event reporting using Event Enrollment objects.
Event enrollment by client role processes is done at standard Notification Class objects provided by the interfaces. The setup and configuration of event reporting is done using the respective properties of the event generating objects and Notification Class objects.

10.1.1 New Event Algorithm ACCESS_EVENT

A new event algorithm ACCESS_EVENT is introduced for the stateless event reporting of the Access Point object. Stateless means that the Access Point or Event Enrollment object does not stay in a respective event state until a condition disappears. Events of an Access Point are related to a certain user action and Authentication & Authorization Process decision. They are instantaneous, and after the event the Access Point is again in normal condition. For the sake of consistency, the Event_State property in the Access Point or Event Enrollment object is present and remains in NORMAL Event State.

When different event detection and reporting setups on a single Access Point are needed, the Event Enrollment object is used. The new ACCESS_EVENT algorithm is defined to enable Event Enrollment objects to report access events.

The event algorithm is described from the viewpoint of the Event Enrollment object. But the same algorithm is applied by the Access Point object for intrinsic reporting of Access Alarm events and Access Transaction Events.

This new event algorithm is defined in Addendum *j* to BACnet 2004 [ADJ] part 7.

An ACCESS_EVENT event occurs when the value of the referenced property (typically the Access_Event property of the Access Point) changes and becomes equal or is updated to one of the values contained in the List_Of_Access_Events. Since the events are stateless, no Time_Delay is involved in the conditions to determine an event.
The value of the referenced property at the time an ACCESS_EVENT occurs shall be used in carrying out the algorithm until the next ACCESS_EVENT. For the purposes of event notification, ACCESS_EVENT generates TO-NORMAL transitions.
There is no clearing condition defined, since NORMAL event state is never left, and the events are stateless.

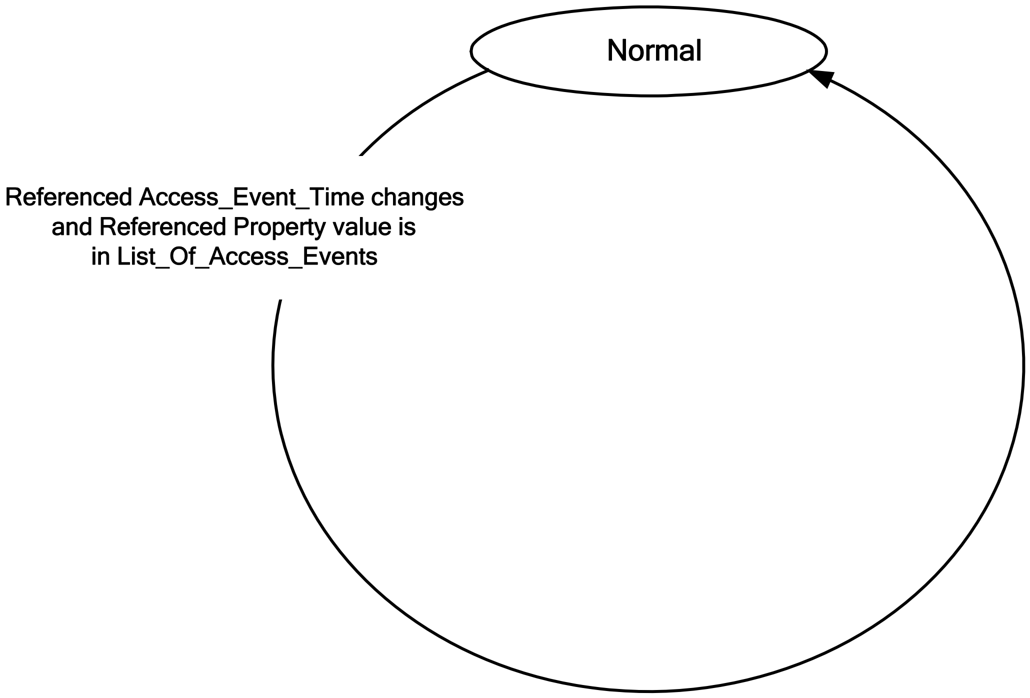

Normal
Referenced Access_Event_Time changes and Referenced Property value is in List_Of_Access_Events

Figure 10–1, ACCESS_EVENT Algorithm

Event Enrollment objects supporting ACCESS_EVENT enrollments use the Access Point's Access_Event and Access_Event_Time property as the monitored (referenced) value, and need the following event parameters.

Event_Type	Event_States	Event_Parameters
ACCESS_EVENT	NORMAL	List_Of_Access_Events Access_Event_Time_Reference

List_Of_Acces_Events contains all Access Event values which, when taken by the monitored value, cause a TO_NORMAL transition, notified by an ACCESS_EVENT event notification. Whenever the referenced Access_Event_Time property changes, the monitored value, as referenced by the Object_Property_Reference property of the Event Enrollment object, is evaluated again to determine if an event notification has to be issued.

Event parameters are those event related configuration values that are stored by the Event Enrollment object itself in its Event_Parameters property. For details see also the Event Enrollment object definition in the BACnet standard [STD] clause 12.

Note that there is a single List_Of_Access_Events only. The ACCESS_EVENT algorithm is not intended to differentiate between Access Alarm Events or Access Transaction Events, as the Access Point does with intrinsic reporting. Supporting both levels of Access Events is achieved by setting up distinct Event Enrollment objects. The differentiation between Access Alarm Events and Access Transaction Events is achieved by appropriate setup of the Event Enrollment objects and associated Notification Classes.

The notification parameters conveyed in an ACCESS_EVENT event notification are:

Event_Type	Notification_Parameters	Description
ACCESS_EVENT	Access_Event	The new value of the referenced property
	Status_Flags	The Status_Flags of the referenced object
	Access_Event_Tag	The Access_Event_Tag of the referenced object
	Access_Event_Time	The time of update of the referenced property, as indicated by the referenced Access_Event_Time property.
	Access_Credential	The Access_Event_Credential of the referenced object
	Authentication_Factor, if present	The Access_Event_Authentication_Factor of the referenced object

10.2 Logging

The PACS data model exclusively uses BACnet event reporting mechanisms. All events are notified using either UnconfirmedEventNotification or ConfirmedEventNotification.

Event logging is done in a standard way by using the Event Log object as defined by the BACnet standard addendum *b* to 135-2004. There are no extensions to this object needed, since it is capable of logging any kind of UnconfirmedEventNotification or ConfirmedEventNotification.

Event Log objects may be instantiated locally in a PACS controller, or remotely in any other device such as a PACS server device.

The Event Log object stores event notifications issued by any BACnet object. The association of Event Log objects to event sources is not exposed by the Event Log object, and is considered a local matter. Remote Event Log objects implicitly require subscribing at the remote Notification Class objects of the remote event source objects of interest, while local Event Log objects may obtain event notifications through some internal mechanism, not exposed in BACnet.

The Event Log object has mechanisms to manage its log buffer, including high-water mark events (BUFFER_READY), ReadRange service support to retrieve logged entries from its log buffer, and commands to clear a log buffer.

11. Functions and Features Inherent in the BACnet Framework

The following are summarized in this section as they are not Access Control specific, but are the framework upon which the PACS extensions are constructed. This framework will be required in whole or in part to implement a PACS, or an interface to a PACS.

11.1 BACnet Data Types

BACnet application service parameters and properties of objects build on the same data type definitions. A set of defined primitive data types are used, or constructions of complex types based on the primitive types. The formal definition of these data types, as well as their encoding using the BACnet encoding scheme, is done using ASN.1.

11.1.1 Primitive Data Types

BACnet defines 13 primitive data types. These are used as data types of properties or application service parameters. They also serve as the base for constructed data types. This set of primitive data types is not extended for new applications such as PACS.

The defined primitive data types are:

NULL	"No Value"
BOOLEAN	TRUE or FALSE
Unsigned	Unsigned number. Basically unlimited in size/range. Limited size subtypes are: Unsigned8: Range 0..255 Unsigned16: Range 0..65535 Unsigned32: Range 0..4294967295 BACnet does not use Unsigned values larger than 32 bits. So if Unsigned is specified, applications must be ready to handle 32 bit values.
INTEGER	Signed number. Basically unlimited in size/range.
REAL	ANSI/IEEE-754 single precision floating point number
Double	ANSI/IEEE-754 double precision floating point number
OCTET STRING	Block of octets. Basically unlimited in size.
CharacterString	Character Strings. Supported character sets: • ANSI X3.4 (ASCII) • IBM™/Microsoft™ DBCS (Double Byte Character Sets) • JIS C 6226 (Japanese) • Unicode ISO 10646 (UCS-4 encoding) • Unicode ISO 10646 (UCS-2 encoding) • ISO 8859-1 ISO Latin 1 • Unicode ISO 10646 (UTF-8 encoding, with upcoming Addendum *h* to BACnet 2004)
BIT STRING	String of bits, or bit set. Basically unlimited in number of bits
ENUMERATED	Value of a defined enumeration.
Date	Date. Four octets: Year, Month, Day of Month, Day of Week. X'FF' in any of the octets is considered "wildcard".
Time	Time of day. Four octets: Hour, Minutes, Seconds, Hundredths of Seconds. X'FF' in any of the octets is considered "wildcard".
BACnetObjectIdentifier	32 Bit BACnet Object Identifier. Constructed: • 10 most significant bits for Object Type • 22 least significant bits for Object Instance of Object Type

11.1.2 Constructed Data Types

SEQUENCE: Structured data types are a sequence of fields of various types. Each field may be of a primitive data type or constructed data type.

CHOICE: Choices may be defined. One of the defined options is used in a property or application service parameter. Each of the available options may again be of primitive data type or constructed data type.

SEQUENCE OF: Sequences of elements of the same BACnet Data Type may appear as properties (arrays or lists), and in application service parameters. Each element may be of primitive data type or constructed data type. Each element must comply with the same data type definition.

ABSTRACT-SYNTAX.&Type: BACnet allows for proprietary data types. A set of possible data types may be used as a structure field or choice option. This is then defined as data type ABSTRACT-SYNTAX.&Type. This indicates that any primitive or constructed data type may be used. The construction must be based on BACnet's primitive data types, but SEQUENCE, CHOICE or SEQUENE OF is possible. The encoding must follow the BACnet encoding rules.

11.1.3 Array and List Properties

Properties may be an Array or a List of elements of the same BACnet Data Type.

11.1.3.1 Array Of

These are properties which contain a sequence of size N of elements (SEQUENCE OF) of the same type. There are no restrictions on element values. Multiple elements may have the same value.

Each element is addressable by an index. Index zero provides the number of elements in the array. Index 1 provides the first element. Index N provides the last element.

In the BACnet Standard, such properties are specified as a *BACnetArray[N] of Data Type*, where the size N may be specified by the standard, or left to the application as a local matter.

If the Data Type of the array elements is a list (i.e. SEQUENE OF), then the content of such elements can be accessed as a whole only using read and write application services. List manipulation services may be used to access individual elements within a list that is an array element. The property definition for this is *BACnetARRAY[N] of List of Data Type*.

11.1.3.2 List Of

These are properties which contain a sequence of undefined size of elements (SEQUENCE OF) of the same data type. There are no two elements which contain the same value. It could also be seen as a set of values.
Elements are addressed by value. Specific application services (i.e. list manipulation services) are available to add or remove an element of a list, identified by its value. The content of the list may be read by reading the entire list, or may be modified by writing the entire list.
In the BACnet Standard, such properties are specified as a *List of Data Type*. The maximum size is a local matter, and typically limited by the resources available in a device.

11.2 Protocol Stack

The BACnet protocol stack is used to transfer application services between physical devices. Several networking options are defined, but these are transparent to the processes performing application functionality and communicating with other processes.

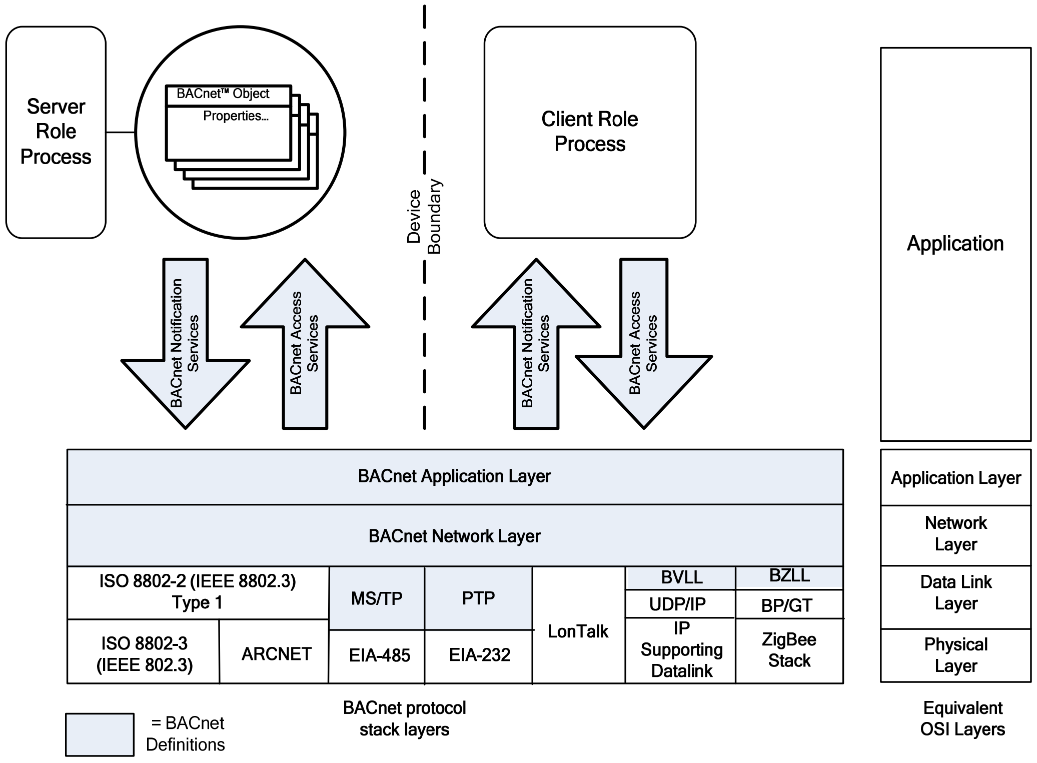


Figure 11–1, BACnet Protocol Stack Overview

11.2.1 BACnet Application Layer

The application layer supports two types of transmission of application services. The application services of BACnet use either one:

- **Confirmed Services** include a request and a response message being transferred. Both the request or response messages may be segmented to convey large amounts o data. Confirmed services are addressed to a single device only (i.e. Unicast).
- **Unconfirmed Services** consist of a request message only. No response is expected. Segmentation is not supported for such messages. Unconfirmed services may be addressed to a single device (i.e. Unicast), to a group of devices (i.e. Multicast) in a BACnet network if supported by the datalink, to all devices of a BACnet network (i.e. Local or Remote Broadcast) or to all devices of an inter-network (i.e. Global Broadcast).

The BACnet application layer also includes a fixed data encoding definition, which defines how data is encoded for transmission. This encoding is closely related to the BACnet Data Type definitions in ASN.1.

11.2.2 BACnet Network Layer

The Network Layer supports routing of messages through the inter-network, and connection control of temporary PTP based links. The basis for routing is Network Numbers associated to each BACnet network. A BACnet network is built of a single data link type, each having its own MAC address space. The sum of all networks of a system form an inter-network. It provides a uniform interface to the application layer, hiding specifics of data links which are used in a network..

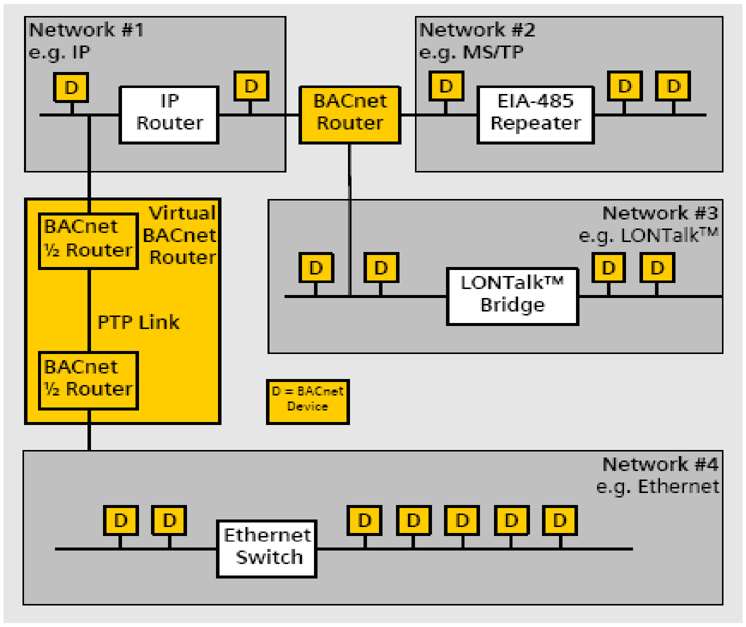


Figure 11–2, BACnet Inter-Network

11.2.3 Data Link Layers

BACnet defines some data link layers, and defines the use of standardized data links. Proprietary data link layers may be used, but are required to support features expected by the BACnet Network Layer. Virtual data link layers allow running multiple BACnet devices in a single physical device.

BACnet defined data link layers

- MS/TP: Master-Slave/Token-Passing using EIA-485 physical layer
- PTP: Point-To-Point using EIA-232 or Modem physical layers
- BVLL: Intermediate layer above UDP, to provide data link features expected by the BACnet Network Layer, but not sufficiently supported by UDP and IP.

Used standard data link layers:

- ISO 8802-2 link layer control over Ethernet (IEEE 802-3)
- ISO 8802-2 link layer control over ARCNET
- LONTalk by using maximum size (228 octets) explicit messages, on any physical layer supported by LONTalk
- BACnet over ZigBee BACnet protocol tunneling cluster (BP) and generic tunneling cluster (GT) as defined in the ZigBee Commercial Building Automation (CBA) profile and addendum q to the BACnet standard [STD].

11.3 Standards

BACnet is an ANSI standard (ANSI/ASHRAE 135-2004) as well as an ISO standard (ISO 16484-5). Further, BACnet normatively references other standards wherever appropriate and useful. Some of the standards referenced by BACnet which are relevant to Access Control are listed below.

Networking:

- Transport protocols such as Ethernet (IEEE 802-3) and IP
- EIA 232 and EIA 485 for serial connections and networks

Data Definition and Encoding

- ASN.1 (ISO 8825) for data definition and encoding specification (Encoding itself is BACnet specific!)

Network Security:

- AES for encryption
- SHA-256 for message authentication

Character Sets:

- ANSI X3T4 (a.k.a. ASCII)
- Unicode character sets and character encodings
- ISO 8859 character sets

11.4 Extensibility

BACnet supports extensions of data models and services for proprietary purposes. See clause 23 of the BACnet standard [STD]. Possible proprietary extensions include:

- Object types
- Properties of objects
- Enumerations
- Services on any protocol layer

Further, BACnet is designed for standard extensions and continuously maintained for extensions and improvements. ASHRAE is the maintenance organization of the standard and ASHRAE's Standing Standard Project Committee SSPC-135 is the responsible body.

11.5 Scalability

BACnet supports scalability on various levels:

- System size: BACnet allows addressing 4,194,303 single addressable devices in a system, where each device may contain up to 4,194,303 object instances of each BACnet object type.
- Device performance: BACnet can be implemented on small 8 bit platforms up to high performance computers such as 64 bit GHz-range machines.
- Network performance: BACnet supports networking technologies from 2400 baud serial links up to 10 Gigabit Ethernet LAN and beyond.
- Network topology: BACnet allows creating systems with global span. Various networking technologies support local and wide area networks of almost any topology.

11.6 System Security

In BACnet, two levels of system security are contemplated.

- Network Security: A completely revised definition of network security within BACnet is currently in public review at ASHRAE. This includes features such as encryption, data integrity, device authentication and user authentication. These features are based on well known standards such as AES, HMAC, MD5 and SHA-256. For details see Addendum g to the 2004 version of the BACnet standard [STD], as available from ASHRAE web site (www.ashrae.org) or, when approved, at the BACnet web site (www.bacnet.org).
- Logical Access: The Life Safety and Security Working Group (LSS-WG) within ASHRAE SSPC-135 is creating standard definitions for logical access control for data and services within a BACnet system. At present this functionality is addressed in a vendor proprietary matter.

Additionally, BACnet/IP may be used on IP networks that use IP inherent security mechanisms, such as IPsec, Virtual LANs or VPN techniques. The disadvantage with such solutions is that no user or role identification is conveyed with BACnet messages, so server side authorization is not effectively supported.

11.7 IT Systems Connectivity

BACnet is currently in the process of adding a Web Services (SOAP) definition to allow using SOAP to access BACnet based systems. For details see Addendum c to the 2004 version of the BACnet standard [STD], as available from ASHRAE web site (www.ashrae.org) or the BACnet web site (www.bacnet.org). It is expected that the conversion of BACnet accessible data into SOAP accessible data can be achieved in a generic way such that the SOAP implementation is transparent with regard to PACS.

BACnet supports the implementation of generic IT oriented machine or user interfaces, such as OPC servers, Web Pages Servers, REST servers etc. Such interfaces typically are transparent to extensions made to the data model and services.

With the use of IP networks as BACnet data link, BACnet may run over IP based IT infrastructure. Through this, IT departments may manage the BACnet network infrastructure.

11.8 Functionality Deployment

BACnet provides complete freedom of functionality deployment. A device may contain just simple functional elements, or be very sophisticated containing a huge set of functions of various application domains. The physical structure of the system is completely independent of the functional / logical structure of the system.

11.9 Networking Technology Independence

BACnet supports various networking technologies:

- Networking as defined by the BACnet standard: MS/TP (Master-Slave/Token-Passing on EIA-485), PTP (Point-To-Point on EIA-232 or Modem)
- Networking technologies used: Ethernet, ARCnet, LON
- Networking technologies supporting IP, which includes wide area networks and wireless LANs (802.11).
- ZigBee based wireless networking
- Proprietary and virtual networking technologies are permitted as a local matter, but transparent to other BACnet devices.

11.10 Device Implementation Technology Independence

BACnet does not prescribe, require or imply hardware and software technologies devices are to be implemented with. A manufacturer has complete freedom to choose CPU families, operating systems, programming languages and environments etc.

12. Terms & Abbreviations

ASN.1	Abstract Syntax Notation One. ISO standard 8824 defined language for the definition of syntaxes and data types. Mainly used to define protocol data models.
CBEFF	Common Biometric Exchange File Format. NISTIR 6529. Describes a set of data elements necessary to support biometric technologies in a common way
CHUID	Card Holder Unique Identifier. Data format used in PIV smart cards.
COV	Change-Of-Value. A reporting mechanism of BACnet. A subscriber process subscribes for change of value notifications from an object (Object-Level COV), or a property of an object (Property-Level COV).
ETS	Electronic Ticketing System
FASC-N	U.S. government Federal Agency Smart Card Number
FIPS-201	U.S. Federal Information Processing Standard 201 [FIPS-201]
GUID	Globally Unique Identifier
HMAC	Hashed Message Authentication Code
HRMS	Human resources management system
HVAC	Heating, Ventilation & Air Conditioning
IDS	Intrusion Detection System
IP	Internet Protocol. Up to now, BACnet supports IPv4, but there are activities to extend it to be capable to run over IPv6.
MAC	Media Access Control. A sub-layer of the OSI data link layer, managing access to a shared medium.
NIST	United States Government’s National Institute of Standards and Technology
OPC	OLE (Object Linking and Embedding) for Process Control. See www.opcfoundation.org
OSI	Open System Interconnection. ISO standard 7498 defining a model of seven protocol layers.
OSIPS	Open Systems Integration And Performance Standards. SIA initiative to develop integration and performance standards for security products.
PACS	Physical Access Control System
Passback	This term is used in the Access Control industry for the act of passing back an Access Credential to a further user to allow this user to request access.
PIN	Personal Identification Number
PIV	Personal Identity Verification. This is the Personal Identity Verification standard as defined by NIST in FIPS-201, in the course of US government initiatives related to HSPD-12, the US Homeland Security Presidential Directive Number 12: Policy for a Common Identification Standard for Federal Employees and Contractors
PKI	Public Key Infrastructure. Authentication and cryptography methods using asymmetric public and private key pairs.
RBAC	The Role Based Access Control (RBAC) model for access rights and owner structuring is introduced by NIST and used widely by the IT industry. See also ANSI/INCITS 359-2004.
REQ	See REX
REST	Representational State Transfer. Roy Fielding: "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use." See various resources on the world-wide web.
REX	Request-to-Exit button
SIA	Security Industry Association (www.siaonline.org)
SOAP	Simple Object Access Protocol. XML based protocol to access information over HTTP transport protocol.

13. Revision History

September 27, 2008	Updated for consistency with fourth public review version (September 2008) of Addendum *j* to BACnet 2004
March 28, 2008	Updated for consistency with third public review version (March 2008) of Addendum *j* to BACnet 2004
October 19, 2007	Updated for consistency with second public review version (September 2007) of Addendum *j* to BACnet 2004
March 1, 2007	Updated for consistency with first public review version of Addendum *j* to BACnet 2004 and published Addendum *f* to BACnet 2004
November 3, 2006	Renamed to BAC-10-06. Minor corrections and updates. Added contributors list on front page. Added this revision history.
October 31, 2006	Derived from HJM-002-23, becoming HJM-002-24.