BSR/ASHRAE Addendum *cp* to
ANSI/ASHRAE Standard 135-2020

Public Review Draft

Proposed Addendum *cp* to Standard 135-2020, BACnet® - A Data Communication Protocol for Building Automation and Control Networks

First Public Review (September 2023)
(Draft shows Proposed Changes to Current Standard)

This draft has been recommended for public review by the responsible project committee. To submit a comment on this proposed standard, go to the ASHRAE website at www.ashrae.org/standards-research--technology/public-review-drafts and access the online comment database. The draft is subject to modification until it is approved for publication by the Board of Directors and ANSI. Until this time, the current edition of the standard (as modified by any published addenda on the ASHRAE website) remains in effect. The current edition of any standard may be purchased from the ASHRAE Online Store at www.ashrae.org/bookstore or by calling 404-636-8400 or 1-800-727-4723 (for orders in the U.S. or Canada).

This standard is under continuous maintenance. To propose a change to the current standard, use the change submittal form available on the ASHRAE website, www.ashrae.org.

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

ASHRAE, 180 Technology Parkway NW, Peachtree Corners, GA 30092

[This foreword, the table of contents, the introduction, and the "rationales" on the following pages are not part of this standard. They are merely informative and do not contain requirements necessary for conformance to the standard.]

FOREWORD

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

In the following document, language to be added to existing clauses of ANSI/ASHRAE 135-2020 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout. Only this new and deleted text is open to comment at this time. All other material in this document is provided for context only and is not open for public review comment except as it relates to the proposed changes.

The use of placeholders like XX, YY, ZZ, X1, X2, NN, x, n, ? etc., should not be interpreted as literal values of the final published version. These placeholders will be assigned actual numbers/letters only after final publication approval of the addendum.

135-2020*cp*-1 Addition of Authentication and Authorization

Rationale

BACnet/SC provided a mechanism to create secure networks where participation on the network is limited to those devices that possess a trusted TLS certificate. While this kept out rogue actors, it did not differentiate between the devices within the SC network, therefore they all have equal authority.

This framework provides a mechanism for adding strong identity (authentication) and permissions (authorization) to client devices so that target devices can allow or deny certain operations based on that identity and permission.

With this, devices are no longer equal, and fine-grained authorization policies can be specified as needed to limit certain operations to certain clients.

[Replace Clause 17, Virtual Terminal Services, p. 777]
[Note to reviewer: This new service category logically belongs with the other service clauses 13-16 and it had been decided that the outdated Virtual Terminal Services should be removed from the BACnet standard]

17 AUTHENTICATION AND AUTHORIZATION SERVICES

This clause describes mechanisms that are used for authentication and authorization of BACnet devices.

17.1 Overview

BACnet is fundamentally a protocol for the free interchange of information between devices. There are, however, some operations that need to be protected, either from an innocent but misconfigured device, or a deliberately malicious device.

BACnet/SC defines a way to create secure tunnels that are similar in function to VLAN or VPN technologies. These tunnels protect the devices from outside attacks. However, within the tunnels, all the devices have equal permissions. Historically, proprietary solutions were created to define which device is allowed to perform certain operations on other devices. For example, this could have been done with some form of " "Allow List"" in the end device or a ""Firewall Router"" that does packet inspection to allow or deny messages between networks.

Most operations in BACnet are ""open operations"", meaning they can be performed by any device without regard to a particular authorization policy. For example, this would include most ""data sharing"" that consists of reading or reporting public data, like sensor readings. However, some operations, such as writing control commands or changing configurations, are considered "protected operations" and should not be open to all devices to perform.

The process of defining which device can perform which protected operations in which other device(s) is known as authorization and is set by an authorization policy. The process by which a device asserts that it is the specific device that has been granted an authorization is known as authentication.

This clause defines mechanisms that can be used for a variety of situations that require a device to be authorized before it can perform a protected operation in another device. Performing a protected operation requires both authentication of the client device and the presence of policy that authorizes that client to perform the operation. See Clause 17.3.7 for a description of the variety of authentication options that are available for interactions among different capabilities of devices and networks.

For flexibility of deployment and interaction with newer and older client devices, the BACnet authorization mechanism supports two possible flows of authorization policy.

The first method is a "centralized policy" mechanism based on OAuth 2.0. See RFC 6749 and its related extensions. With this mechanism, the applicable authorization policy is delivered "on the fly" by the client device to the target device along with the request for a protected operation. The target device will trust this policy because it has been signed by a third device that it trusts. In OAuth terms, the target device is known as the "resource server", and the delivered policy is known as an "access

token" and the trusted third party that signed the policy is the "authorization server". Aligning with modern OAuth practice, BACnet uses "sender-constrained" tokens that cannot be used if stolen, rather than "Bearer Tokens" that need to be protected.

The second method is a "local policy" flow that can be used where the client device does not, or cannot, send a token. This is analogous to a traditional Access Control List where the actions of the client are authorized by some locally configured rules in the target. In this case, the applicable authorization policy for the client must be pre-configured into the target before the protected operation is attempted by the client.

It is an important design feature that the local policy information is in the same format and has the same capabilities as the policy that is delivered by an access token. Effectively, a token just "delivers a policy". Therefore, to the administrator of authorization policies, the two mechanisms are as similar as possible, and policies can be defined in one way but deployed differently depending on the needs of the devices involved.

Each of these mechanisms has its advantages. The token mechanism allows granting a policy to a client without the need to update the local authorization policy in the target devices. This is especially advantageous for granting temporary access, e.g., for a technician or installer. In this case, the client device carries its authorization with it, and the target devices trust the authorization policy because they can verify that the access tokens were issued by the authorization server that they trust.

The local policy has the advantage of working with existing clients that may not be aware of tokens. For example, a first generation BACnet Secure Connect device can present a strong authentication with its TLS certificate, but it cannot present an access token. Therefore, the policy that would be in the token can be pre-configured into the end device and the policy will be evaluated as if the policy had been delivered by the token. The local policy can also work for non-secure devices that are nonetheless in a secure environment. For example, a network of devices that is physically secure, or secured by some other means like a VLAN, can set a policy for devices on that local network to interact with each other.

17.2 Levels of Trust

In addition to the policy deployment options, the authorization policies provide flexibility to allow the site administrator to establish the required level of trust that is appropriate for a protected operation.

For example, one policy could state that only a direct connection is allowed, a different policy could state that the device must be on the same network, and another could state that any network is allowed as long as all the intermediaries are auth-aware. See Clause 17.3.4 for definition of "auth-aware".

The "direct connect" case might be appropriate for situations where the data to be transferred is confidential or critical and the devices do not trust any intermediaries. For example, in BACnet/SC, the intermediaries, i.e., hubs and routers, are "TLS terminating" devices, meaning that once a message has been delivered to the intermediary, it is no longer encrypted internally. It is therefore possible for a compromised intermediary to read the data in the message, modify the message, or even create a new message entirely. With direct connect, the receiver has immediate access to the sender's TLS certificate and thus sees its identity first-hand and data can be exchanged with end-to-end confidentiality.

The "same network" case might be appropriate for some operations because each BACnet/SC network has a common TLS certificate issuer, thus all devices have a common trust anchor for their identity. In this case, the devices have only one intermediary, the hub, and it is possible that all devices and the hub are maintained to a tighter level of monitoring and control than other networks.

Most situations will allow "any network" since that level of policy nonetheless still requires that the identity information is conveyed only by a trusted chain of intermediaries. As is true for any TLS terminating "middle box", e.g., a corporate proxy firewall, BACnet hubs, and routers must be maintained with proper patching and monitoring to ensure that they remain trustworthy.

For end devices, BACnet authentication and authorization provides assurance that a compromised device cannot obtain more privilege than it has normally. Since a device cannot spoof its own identity, it is not possible for it to pretend to be a more powerful device even if it obtains a powerful access token. In practical terms, this means that a compromised VAV box could send bogus information to its AHU, but it cannot start writing anywhere else.

17.3 Authentication

The "identity" of a BACnet device is its device instance number. Authentication is the process of proving that a device's claimed instance number can be trusted. Authentication of BACnet devices is provided by the BACnet datalink layer by some verifiable and unmodifiable means. For example, in BACnet/SC, the attestation of device identity is provided by the operational certificate, and the trust in the certificate is the basis for trust in the identity by the peers on that SC network.

The authentication information is verified at the "first hop" at the datalink level and then relayed by trusted hubs and routers to the destination device. The mechanism for trusted relaying is defined in Clause 17.3.4.

Authentication of human users is provided indirectly by trust of the devices that are performing the human user authentication. Since these mechanisms vary widely, the actual authentication mechanism, e.g., smart card, password, biometrics, etc., is out of scope for the BACnet protocol, except to the extent that user and role information is conveyed by the BACnet services from the source device to the end device. Authentication can be based on individual users and/or their roles. Role information could be obtained from something as advanced as a federated directory system or as simple as a PIN entered on a wall device. Although the user and role numbers are common for all BACnet devices, not all authentication is considered of equal authority. The trust of the authentication is based on the trust of the device that is the source of the information. The ability to perform a protected operation is based on authorization policy, and that policy is based on the sending device.

17.3.1 Claiming Authentication

The originating device claims its identity with data attributes that accompany the NPDU for the protected operation. The authentication information claim consists of a "Source" data attribute which is placed on the original message and not removed as the message passes through hubs and routers.

This method is used, rather than have the first intermediary add this option, for several reasons. The first is that only the originating device knows the user information. Second, adding the options before the message is originally sent means that the intermediaries do not need to modify the structure of the message by adding or removing options when forwarding or routing the messages. And finally, since options are innocently relayed by earlier SC devices, there is no security advantage to have them added by an auth-aware first hop - a malicious device could simply add the bogus options itself and have them relayed. This is why the rules in Clause 17.3.3 are defined to block such fabrications when they are received.

Not all messages require authentication, so the choice to include the authentication data attribute is made by the message originator. If an originating device wishes to authenticate a particular message, it will apply the attribute to the message. If the implementation does not make a distinction of when authentication is needed, it can simply include the attribute on every message, at the expense of a small overhead.

Since this "claim" of authentication can be added maliciously, the acceptance or rejection of such a claim will be determined by the receiver. See Clause 17.3.4.

17.3.2 "Source" Data Attribute

The source of an operation is identified by a "Source" data attribute that accompanies the NPDU for a protected operation. For BACnet/SC, the Source data attribute is conveyed in the 'Source' Data Option. See Clause AB.2.3.X3.

The trustworthiness of the Device field is asserted by the Auth Path field. This Boolean indicates that the Source attribute originated in an auth-aware device and has never left a chain of auth-aware devices on its way to the destination. If the message traverses any non-auth-aware devices, it will be set to false. The originating device sets its initial value to 'true'. See Clause 17.3.3 for the rules that can set this to 'false'.

The Source attribute is defined to be an "Every Segment" data attribute, so it appears on every segment of a segmented message and its value must remain consistent. If any segment is received with a different or absent value, the entire NPDU shall be dropped.

The Source data attribute contains these fields:

Item Name	Data type	Description
Auth Path	Boolean	1 = NPDU never left chain of auth-aware devices, 0 otherwise
Device	Unsigned	The device instance of the originating device
User ID	Unsigned	The user identifier of the initiator of the operation. See Clause 17.4.5
User Role	Unsigned	The user role of the initiator of the operation. See Clause 17.4.5.

The originating device adds the Source attribute to the NPDU for a protected operation. If there is no user information available, both the User ID and User Role fields shall be set to 0.

17.3.3 Validating Authentication

The initial validation of the identity claim of the Source data attribute shall be enforced by the first device receiving the original message at the datalink level since that is where the cryptographic proof of identity can be provided by the sender.

In BACnet/SC, the identity of a device is provided by a Subject Alternative Name (SAN) entry in the TLS certificate. The SAN entry shall be a URI in the form defined by Clause Q.8. Additionally, the relaying capabilities are provided by query parameters on the URI. If the peer is a router, the query parameter "router" is appended and if the peer is a hub, the parameter "hub" is appended. The following are examples of valid URIs, depending on the functions performed by device 1234: "bacnet://1234", "bacnet://1234?router", "bacnet://1234?hub", "bacnet://1234?router&hub", "bacnet://1234?hub&router".

During the mutual TLS authentication phase of a BACnet/SC connection establishment, each peer shall take notice of the other peer's certificate. If the certificate has a Subject Alternative Name with the "bacnet" URI scheme, that URI shall be parsed, and the device instance and "router" and "hub" indicators shall be remembered in the context of the connection.

If there is a mismatch between the identity provided by a datalink and the instance of the Device object, or if there is a mismatch between datalinks on multiple ports, it is a local matter how the device chooses to behave until the situation is resolved. For example, the device could decide to adopt the datalink identity, or it could decide that the need to maintain a stable identity temporarily outweighs the need to send authentication messages as a client. To prevent unexpected behavior, tools that deploy datalink identities shall attempt to notice and report a mismatch before deploying the identity.

17.3.4 Auth Aware Devices

Devices performing the authentication and authorization mechanisms are designated as "auth-aware" devices. Secure devices that predate the publication of this clause are designated as "non-auth-aware" devices. While both are "secure", the distinction is important to the trust of the client identity and therefore the authorization.

The identity information claimed by auth-aware devices will be relayed to the destination by non-auth-aware devices, but it will not be trusted for authorization decisions unless the entire chain of intermediaries is auth-aware. This is because trust in the authentication information is based on trust that the "first hop" device validated the identity of the originator.

Imagine a chain of humans, some of whom have been trained in the process of checking IDs and some have not.

> Alice says "I'm Alice" → Intermediary X → Intermediary Y → "I'm Alice" reaches Bob.
>
> Bob wants to ask Y, "Did you see Alice's ID?"
>
> If Y answers "what?", then Bob knows the answer is "no".
>
> A trained Y would ask X "Did *you* see the cert?". If X answers "what?", then Y will answer Bob with a "no".
>
> But if X answers "Yes, I saw it", then Y can answer Bob with a "yes".

This flow of questions is inefficient of course. So, rather than "ask upstream" for every message, Y could assure Bob, once, "I will only send you things if I have personally seen the ID or if I trust that the person who sent it to me has done the check." Likewise,

of course, X would have said the same thing to Y. From then on, when X sends something to Y, Y can send it to Bob, and Bob can trust that the ID was checked.

In BACnet, therefore, it is necessary that all auth-aware intermediaries need a way to know that any upstream intermediaries are also auth-aware. With this knowledge, if the identity successfully reaches the destination, the destination will know that the identity was checked at the point of origin, no matter how many intermediaries were traversed.

In BACnet/SC, the assurance of being auth-aware is provided by the exchange of "Hello" options during the connection phase. Only auth-aware devices know about the "Hello" option, so its presence becomes the assurance of understanding identity checks. An auth-aware initiating peer shall send a "Hello" option with the Connect_Request message, and an auth-aware accepting peer shall send a "Hello" option with the Connect_Accept message. If the "Hello" option is present, the peer shall be marked as auth-aware, else it shall be marked as non-auth-aware. It should be observed that a compromised device that is normally non-auth-aware could make itself appear to be auth-aware by sending a Hello. But it still cannot impersonate another device or claim to be a hub or router because those aspects are protected by the TLS certificate. So, while compromised, its actions are limited to what the original device was already allowed to do.

17.3.5 Relaying Authentication

The datalink level rules for validating and relaying authentication information are symmetrical and are the same for devices, hubs, routers, and direct connect, so they are written in terms of "sending peer" and "receiving peer" and apply to all secure connection types. The following operates on the Auth Path field of the "Source" data attribute. If "Source" is absent, these rules are ignored.

Outbound:

1. If Auth Path is true and the receiving peer is non-auth-aware, the sending peer changes Auth Path to false.

Inbound:

1. If Auth Path is true and sending peer is non-auth-aware, then drop the message.
2. If Auth Path is true and sending peer does not have a datalink identity, then drop the message.
3. If Auth Path is true and ‘device’ does not match the sender’s datalink identity and the sending peer is not marked as a router or hub, then drop message.

From these rules, it can be observed that:

- Given the outbound rule, there is no way for a non-auth-aware device to originate or relay an Auth Path true. If it does, something upstream is compromised or broken and inbound rule #1 will catch it. In other words, network analyzers should never see Auth Path true when it is in fact on a non-auth-aware path.
- Only auth-aware hubs and routers can send a Source identity that does not match their own identity. It is essential that they be allowed to do this of course, since their job is to relay messages from other devices. Note that simply giving a non-auth-aware hub or router a certificate marked "router" or "hub" does not make it auth-aware.
- If an auth-aware device originates Auth Path true, but the message then passes through non-auth-aware intermediaries, it will be changed to false for the rest of its journey, even if it then goes back through auth-aware intermediaries before reaching the destination. This is analogous to Secure Path for BACnet/SC.
- It is possible for a malicious device to get Auth Path true passed through innocent non-auth-aware intermediaries, but it will be blocked at the first auth-aware device. But if there are no auth-aware intermediaries along the way, and the bogus Auth Path true reaches a non-auth-aware destination, it will have no effect because the non-auth-aware destination does not understand the Source attribute anyway.
- An auth-aware sending peer will change Auth Path to false before sending to a non-auth-aware peer, but it will leave the Source attribute it in the message because:
    - The sending peer does not want to add or remove attributes for performance reasons, but it can easily change Auth Path flag from true to false in situ.
    - If the non-auth-aware receiving peer is the final destination, it will just ignore the Source, so it’s harmless to include it.
    - If the non-auth-aware receiving peer is a router, there might be a downstream auth-aware device that actually wants the Source even if the Auth Path flag is false, so it should stay in the message.
- All of this happens in the datalink level with no NPCI inspection or message structure modification.

17.3.6 Effect on Authorization Policies

When specifying an authorization policy, either a local policy or one delivered by an access token, the required client authentication method is a choice of ‘any-method’, ‘secure-path’, or ‘authenticated’.

- The ‘authenticated’ choice means that Source must be present, and its Auth Path flag must be true.
- The ‘secure-path’ choice means that Source can be used if Secure Path is true (even if Auth Path is false), or a Secure Path I-Am or Secure Path ReadProperty of the originating device can be used to determine device instance.
- The ‘any-method’ choice means that the target device can use a plain I-Am or plain ReadProperty to determine device instance.

These are hierarchical from less constrained to more constrained: ‘any-method’ implies that ‘secure-path’ and ‘authenticated’ are also allowed, and ‘secure-path’ implies that ‘authenticated’ is also allowed.

When an auth-aware device receives Auth Path true, it can trust the Source for authorization decisions. If Auth Path is false, then the Source could be malicious, or it could simply have passed through a non-auth-aware path. In this situation, Source might still be useful, but only if the policy has been relaxed to "secure-path" or "any-method". Such a policy can be used to protect against misconfigured devices but not compromised secure devices.

In addition to the "method" used for authentication, the location of the originating device is also significant for policy making. The locations are generalized into three categories: ‘direct-connect’, ‘local-network’, and ‘any-network’.

- The ‘any-network’ choice means that there is no restriction on the location of the originating device.
- The ‘local-network’ choice means that the originating device is on the same BACnet network as the target device, with no BACnet routers involved, i.e., there is no SNET/SADR in the NPCI.
- The ‘direct-connect’ choice means that the originating device is directly connected to the target device, e.g., BACnet/SC Direct Connect, with no hubs or routers involved.

These are hierarchical from less constrained to more constrained: ‘any-network’ implies that ‘local-network’ and ‘direct-connect’ are also allowed, and ‘local-network’ implies that ‘direct-connect’ is also allowed.

17.3.7 Authentication Scenarios

A BACnet internet often contains a mixture of device capabilities that were installed over time. Additionally, the devices likely have a variety of security requirements based on the criticality of their function. To support a wide range of appropriate interactions, the combination of authentication method and authentication origin allows for fine grained tailoring of authorization policies to meet the specific needs of the installation.

In a mixed system, there are generally four levels of security.

- A non-secure network is the lowest level and could allow a rogue device to be connected to the network and send messages to any other network.
- An externally-secured network is one where communication between the non-secure devices on that network are secured by some external means, such as a VLAN or physical security.
- A secure network is a network secured by cryptographic proof that a device has a right to join the network, e.g., BACnet/SC. All devices on this network are known as "secure devices".
- An "authenticated device" is a secure device that has been given a verifiable identity, e.g., a BACnet/SC certificate with an identity in it.

The difference between "secure" and "authenticated" is that a compromised "secure" device can claim to be another device, whereas a compromised "authenticated" device cannot. Therefore, even if an authenticated device is compromised such that its behavior is no longer predictable, that device cannot claim to be a different device and therefore its authorization is limited to what the uncompromised device was already capable of doing.

In addition to the mixture of device and network types, there is a mixture of requirements for protected operations that need to be authorized. Some operations are critical and must be protected against malicious devices, and some are merely in place to protect against misconfigured client devices.

To accommodate the various security requirements between these different groups of devices, BACnet provides two parameters, "method" and "location", to determine when an authorization policy applies to a given client. The combinations are summarized in the following table.

Method	Location	Description
authenticated	direct-connect	The highest level of security, when possible. • No intermediate devices can see or modify the messages. • Client and target both have direct access to each other's TLS certificates, and thus directly validate the authentication information of the peer. • Will not be available between devices that are not directly reachable (e.g., behind NAT or otherwise lacking IP routing) or for devices that do not initiate or accept the optional direct connect capability. • To succeed, the client must have an identity in its certificate. Note that because no relaying is involved, the client is not actually required to be auth-aware since its certificate speaks for itself.
authenticated	local-network	The target trusts only the behavior of a single hub. • The target validates the identity of the hub through direct visibility of the hub's certificate. • It then trusts that the hub will validate the identity of the client by direct observation of the client's certificate. • To succeed, the hub must be auth-aware and both the hub and client must have an identity in their certificate.
authenticated	any-network	The target trusts the behavior of the hubs and routers on the path between client and target. • The target validates the identity of its hub peer through direct visibility of the hub's certificate. • It then trusts that each link in the chain has validated the identity of the upstream peer, all the way to the peer that initially validated the identity of the client by direct observation of the client's certificate. • To succeed, all hubs and routers in the chain must be auth-aware and have an identity in their certificate.
secure-path	direct-connect	Applies to secure clients that do not have an identity in their certificate. • The target knows that the client is a secure device with a valid certificate but cannot trust the identity claim. • Thus, authorization is based on the client being a "secure device", but not based on a specific identity. • The same limitations of reachability and optional capability mentioned in authenticated/direct-connect apply.
secure-path	local-network	The target trusts only the behavior of a single hub. • The target knows that the client is a valid secure device because it is connected to the same hub. • Same observations as secure-path/direct-connect about relying on a specific identity.
secure-path	any-network	The target trusts the behavior of the hubs and routers on the path between client and target. • The target trusts that the client's message originated on a secure network (e.g., BACnet/SC) and that is had never left a chain of secure networks. • Same observations as secure-path/direct-connect about relying on a specific identity.

any-method	direct-connect	Invalid combination. • Direct connection is only possible between secure devices. • secure-path/direct-connect or authenticated/direct-connect should be used instead.
any-method	local-network	Useful for local devices that are protected by some external means, such as VLAN. • If a target knows that all devices on its local network are protected by some means such as a VLAN or physical restrictions, then this allows the trust of peers on that same network. • Can be used by non-secure devices to create an "allow list" for trusted local peers. • Remote (routed) traffic is not trusted because it could be from networks that are not protected by the same external means as the local network.
any-method	any-network	Useful for creating unauthenticated "allow lists". • Can be used to protect against innocent misconfiguration of clients. • Does not protect against malicious intent.

17.4 Authorization

Once a client device has been authenticated to a target device, the target device needs a second piece of information to allow the protected operation. This is provided by an authorization policy. The target device consults this policy to determine if the protected operation is allowed. If it is not allowed, a suitable error response will be returned for confirmed services, or the operation will be silently discarded for unconfirmed operations. The following clauses define the information in a policy and the options for deployment.

17.4.1 Policies

To perform a protected operation in a target device, the client device must be granted authorization to do so. Authorization of protected operations is provided by an authorization policy that considers the authentication of the sending device and the user/role information. An authorization policy specifies the following:

- The time range that the policy is valid.
- The client device that is allowed to use the policy.
- The network location of the client and the method by which it was authenticated.
- The user and/or role that is allowed to use the policy.
- The target device(s) that the policy applies to.
- The set of permissions that the client is granted to perform on the target device(s).

This is represented, both in local policies and in access tokens, as a BACnetAuthorizationPolicy construct. See Clause 12.11.X3 for detailed description of the components of an authorization policy.

17.4.2 Policy Details

Authorization policies are defined using an instance of the BACnetAuthorizationPolicy type. Each policy contains the following fields:

Field	Type	R/O
not-before	BACnetDateTime	O
not-after	BACnetDateTime	O
client	Unsigned	R
origin	BACnetAuthenticationOrigin	R
method	BACnetAuthenticationMethod	R
user-id	Unsigned	O
user-role	Unsigned	O
scope	BACnetAuthorizationScope	R
extension	BACnetExtension	O

The ‘not-before’ and ‘not-after’ fields are exclusive local datetimes defining the period when the policy is in effect. If ‘not-before’ is absent, the policy is in effect immediately and if ‘not-after’ is absent, it has no expiration.

The ‘client’ field is the device instance of the device that the policy applies to.

The ‘origin’ indicates the origin of the client with respect to the target when the policy is enforced. The choices are hierarchical, with a "farther" choice implying that "closer" choices are also valid.

Choice	Meaning	Hierarchy
any-network	originator can be on remote BACnet networks	‘any-network’ will allow ‘local-network’ and ‘direct-connect’
local-network	originator must be on the same BACnet network	‘local-network’ will allow ‘direct connect’
direct-connect	originator must make direct connection	‘direct-connect’ only allows direct connections

The ‘method’ indicates the method by which the client was authenticated to the target. The choices are hierarchical, with a "less constrained" choice implying that "more constrained" choices are also valid. See 17.3.5.

Choice	Meaning	Hierarchy
any-method	Can use non-secure I-Am or ReadProperty	‘any-method’ will allow ‘secure-path’ and ‘authenticated’
secure-path	Must use Secure Path	‘secure-path’ will allow ‘authenticated’
authenticated	Must use Auth Path	‘authenticated’ only allows authenticated

The ‘user-id’ and ‘user-role’ indicate the user and role that the policy applies to. If the ‘user-id’ is absent, it means "any user" and if the ‘user-role’ is absent, it means "any role". Clause 17.4.5 defines meanings for the value 0 for both fields, however, note that the value 0 in a policy matches the value 0 only - it is not a "wildcard". Non-secure devices, and datalinks that cannot convey data attributes, cannot provide this information, so policies for those clients must specify "any user" and "any role" by setting these fields to be absent.

The ‘scope’ field indicates the scopes that are allowed by this policy. It consists of two parts, one for standard scopes and one for non-standard "extended" scopes. For compactness, the standard scopes are represented as a BitString indicating selections from the collection of the standard scope identifiers. The second part is for "extended scopes", represented as list of character strings. There is no implied hierarchy among any scope entries, i.e., no scope entry implies the presence of another.

The meaning and structure of proprietary extensions to the policy is uniquely identified by a URI in the ‘type’ field of the ‘extension’ field. For example, an extension type of "https://example.com/auth#foo" specifies an extension controlled by the owner of "example.com". The URI is not required to be dereferenceable. If a device is given a proprietary extension that is does not understand, then it shall ignore the entire policy, and if it is configured into the Authorization_ACL property, the device shall indicate an error in the Authorization_Status property.

17.4.3 Policy Deployment

For flexibility of deployment and interaction with new and existing clients, the BACnet authorization mechanism supports two possible flows of information.

The first is a "centralized policy" mechanism based on OAuth 2.0. See RFC 6749 and its related extensions and amendments. With this mechanism:

- An Authorization Server issues Access Tokens to a Client that authorizes the actions that can be taken by that Client.
- The tokens are signed by the Authorization Server and bound to the Client, so they cannot be used by others if stolen.
- The Client either tries a protected operation and finds that it needs a token, or it knows to ask for one ahead of time.
- The Client requests an Access Token from the Authentication server for a specific target device or group of devices.
- The request contains a specific "authorization scope", a set of permissions that are being requested by the Client.
- If the Authorization Server contains a policy allowing the request, it returns an Access Token for the requested scope.
- The Client presents the token to the Resource Server along with the protected operation.
- The Resource Server verifies the token’s signature with a key that it has been configured to trust.
- If the protected operation is allowed by the token's scope, then the operation succeeds.

The second is a "local policy" flow where the client device does not, or cannot, send a token. This is analogous to a traditional Access Control List where the client is authenticated remotely but authorized locally. With this mechanism:

- The client authenticates itself with its identity credentials and this authentication is conveyed to the Resource Server.
- The Resource Server consults its local policy to determines the client’s scope of authorization.
- If the protected operation is allowed by the scope, then the operation succeeds.

It is an important design feature that the local policy information is in the same format and has the same capabilities as the information which is delivered by an access token. Effectively, a token just "delivers a policy". Therefore, to the administrator of authorization policy, the two mechanisms are as similar as possible, and policies can be defined in one way but deployed differently depending on the needs of the devices involved.

Each of these mechanisms has its advantages. The token mechanism allows granting a policy to a client without the need to update the local authorization policy in the target devices. This is especially advantageous for granting temporary access, e.g., for a

technician or installer. In this case, the client device carries its authorization with it, and the target devices trust the authorization policy because they can verify that the access tokens were issued by the authorization server that they trust.

The local policy has the advantage of working with existing clients that may not be aware of tokens. For example, a first generation BACnet Secure Connect device can present a strong authentication with its TLS certificate, but it cannot present an access token. Therefore, the policy information that would be in the token can be pre-configured into the end device and the policy will be evaluated as if the policy had been delivered by the token. The local policy can also work for non-secure devices that are nonetheless in a secure environment. For example, a network of devices that is physically secure, or secured by some other means like a VLAN, can set a policy with a client method of "any-method" and client origin of "local-network". That policy would allow devices on that local network to interact with each other without authentication.

17.4.4 Authorization Scopes

Authorization policies define the permissions a client device has for protected operations at the target device. The set of individual permissions constitutes the combined "authorization scope" for the policy. The individual permissions are referred to as "scope identifiers", or sometimes just "scopes".

This standard defines "standard scopes" for common operations. Use of standard scopes greatly improves interoperability; therefore, their use is strongly recommended wherever their semantics apply.

If one of the standard scopes does not apply to a particular protected operation, a device can define its own set of "extended scopes". A human-readable description of these extended scopes can be provided by the Authorization_Scopes property. User interfaces can use the information in this property to construct textual choices to be presented along with the standard scopes.

[Note to reviewer, this table is moved here from Clause W.3.5]

Table 17-1. Predefined Scopes

Scope Identifier	Meaning
view	View (view private data - public data does not need authorization)
adjust	Adjust setpoints
control	Runtime Control (write values for the purpose of controlling actions)
override	Command Override (placing objects out of service, commanding at priorities that would limit runtime control, etc.)
config	Configuration and Programming (configuration and programming actions, adding objects etc.)
bind	Configuration of external references for which the server device will use its own credentials to read or write.
install	Installation (more technical configuration actions, adding IO points, uploading different application programs)
auth	Configuration of authorization-related data.
infrastructure	Announcements of network topology and time information.

For brevity, standard scopes are encoded using bits in the BACnetAuthorizationScope construct. Extended Scopes are encoded as a list of character strings.

17.4.5 Users and Roles

[Note to reviewer, this clause is moved here from Clause W.3.11]

Auditing and local access policies are aided by knowing the identity of the user or entity that initiates an action and what roles they play. Basic identification of this information can be provided by a pair of numbers known as User ID and User Role.

User ID values are positive integers that represent unique human users or processes within a BACnet system. User ID 0 is reserved to indicate that the user is unknown; it is commonly used in conjunction with User Role 0 or 1.

User Role values are positive integers used to group access rights. Example roles could be: HVAC operator, technician, etc. User Roles 0 and 1 are reserved to mean "the system itself". User Role 0 is used for programmed device-to-device communication that is not initiated by human action. User Role 1 is used for device-to-device communication that is initiated by an "unknown human", such as the changing of a setpoint based on button presses on a thermostat. Other User Role values may also be used for device-to-device communication to indicate a particular subsystem that is performing the action, but those values are not restricted by this standard and are taken from the same set of numbers as are used for roles for human users and groups. The values 0 and 1 are the only ones that are reserved specifically for this purpose and shall not be assigned to human user roles.

Assignment of the values for User ID and Role is based on local site policy, but they should be unique across all BACnet devices in a given security context, such that User ID 1234, for example, means the same regardless of its source or destination.

17.4.6 Configuration

The local policy is defined by the Authorization_ACL property of the Device object. See Clause 12.11.X3. The centralized authorization policy, defining the information for the authorization server for a device is defined by the Authorization_Server property. See Clause 12.11.X1.

Each device trusts one authorization server. However, for flexibility in deployment scenarios, there is no requirement that all devices trust the same authorization server. Therefore, a site might have more than one authorization server, each serving a different population of devices. Note that the targets of broadcasts and groups must share the same authorization server since a single access token must be trusted by all intended recipients.

Devices that initiate the AuthRequest service are only required to be able to request access tokens from their own authorization server. If a site uses different authorization servers for different populations of devices, there is a possibility that a client cannot naturally get an access token for a target in a different population. This can be overcome in several ways: 1) the target device could be configured with a local policy for the client device so that a token is not necessary, or 2) the client's authorization server could relay the request to the target's authorization server, or 3) an advanced client can read the target's Authorization_Server property to determine the authorization server to use.

17.4.7 Access Tokens

An access token is a mechanism for delivering an authorization policy to the target device. If a suitable authorization policy is not already configured in the "local policy" option, a client device can request an access token from the authentication server and then deliver that token along with the protected operation.

If a token is delivered with a protected operation, then that token is the only thing considered for authorization. Any local policies and any previously received tokens are ignored. If the delivered token is invalid, then there is no authorization for the operation, i.e., there is no "fall back" to local policy or previous tokens. Failures can be indicated in the Authorization_Status property of the Device object.

An access token is an access policy wrapped with information that allows the target device to trust the authenticity and validity of the policy. An access token contains the following information:

- The device instance of the authorization server that generated the token, known as the 'issuer'.
- An identifier that, when combined with the 'issuer' identifier, uniquely identifies the token.
- The date and time the token was issued.
- The "audience" of the token, i.e., the list of devices and groups that can be the target of the token.
- The policy itself. See Clause 17.4.1 for a description of what a policy defines. This contains the identification of the client device that is authorized to use the token (the "client binding" or "sender constraint") and further restricts how and where the client can use the token.
- A digital signature, signed by the authorization server, that ensures the authenticity of the preceding data.

An access token is requested from an authorization server using the AuthRequest service. This request explicitly contains the client identifier of the device that will use the token. Therefore, it is possible for one device to retrieve a token that will be used by another device. This is known as the "on behalf of" data flow.

An access token is bound to a particular client, i.e., it is "sender constrained", it is not a "bearer token". They can only be successfully used by a properly authenticated client device. The tokens can also be optionally bound to a user and/or role. If a client manages multiple users and has multiple tokens for a single target device, it is the client's responsibility to ensure that it selects an access token that matches the user and role information in the Source data attribute since a mismatch will result in the token being rejected at the target. Similarly, if a client manages multiple tokens for different operations on the same target device, it is the client's responsibility to ensure that it selects the correct token for a given protected operation.

Since access tokens are sender constrained, the only reason to keep them confidential is to hide the fact that one device has been given authorization for another device. Since this information is usually not secret in a typical BACnet installation, there is no reason to apply special protection to the acquisition or transmission of the tokens. They are effectively "public information". If it is desired that a particular policy contained by a token remain confidential, it is possible that the authorization server could be configured to only issue tokens directly to the client device and only via a direct connection. The client devices would then be required to keep this information private and only use the token in a direct connection. Note that it is not possible to hide the fact that a device is communicating with another device. However, a direct connection can hide the contents of the communication. The token policy contains a setting for 'direct-connect' to support the limitation of the client to target communications. However, any restriction on the process of issuing these tokens from authorization server to client is a local matter.

The details of the BACnetAccessToken construct are as follows:

Field	Type	R/O
issuer	Unsigned	R
issued	BACnetDateTime	R
audience	Sequence of Integer	R
policy	BACnetAuthorizationPolicy	R
key-id	Unsigned	R
signature	OctetString	R

The 'issuer' field indicates the device instance of the authorization server that issued the token.

The 'issued' field indicates the local date and time of the authorization server when the token was created by the authorization server.

The 'audience' field indicates where the token can be used. This contains a list of devices and/or groups represented as signed integer values and is required to contain at least one value. Positive numbers are device instances and negative numbers are groups, e.g., group 5 is represented as -5. Group number 1 is reserved to mean "all devices".

The 'policy' defines the actual policy details that are being delivered by this token. The rest here can be considered just a "wrapper" around this policy.

Note that the 'policy' includes important information that restricts who can use the access token. The 'client', 'origin', and 'method' are critical to match the actual token delivery before the policy can be accepted. See Clause 17.2.8 Token Acceptance.

The 'key-id' field indicates which of the two possible signing keys in the Authentication_Configuration property was used to sign the token. A value of 1 means 'signing-key-1' and a value of 2 means 'signing-key-2'.

The 'signature' field is the final field of the ASN-1 production of a BACnetAccessToken. It is a digital signature of all the octets preceding the 'signature' production. The algorithm shall be understood by the authorization server and the target device. The client does not use this information. The public key(s) required to validate the signature is(are) configured into the Authorization_Server property of the Device object of the target device. The private key shall be maintained confidentially by the authorization server.

17.4.8 Token Validation

When a client attempts a protected operation that is not covered by a local policy, the client must present an access token in a "Token" data attribute along with the NPDU. A "Token" data attribute contains the information in the BACnetAccessToken

construct. The fields of this datatype are discussed in Clause 17.4.7. For BACnet/SC, the "Token" data attribute is conveyed with a "Token" Data Option.

The following procedure is described as a function that returns "success" or an error code and possibly data for a "Hint". The actual implementation is a local matter. Since validation of a digital signature is a computationally intense operation, it is a local matter where in this process the validation takes place. If a device is concerned with leakage of information, it can choose to validate the signature up front. If the device is more concerned with denial-of-service attacks, then the validation can be put off until the end. In any case, the use of a token cache will greatly aid in the computational burden of repeated presentation of the same token.

Upon receipt of an access token, the target device shall, in any order:

a) Optionally check if the token has been revoked - a non-required local matter behavior. If it has been revoked, return a REVOKED_TOKEN error code.
b) Check that the ‘audience’ field matches the target device or one of the groups listed in the Authorization_Groups property. If not, return an INVALID_AUDIENCE error code.
c) Check the ‘not-before’ and ‘not-after’ fields if present. If the token is not currently valid, issue an INVALID_TOKEN error code.
d) Check that there is an "Authentication" data attribute along with the "Token" data attribute. If not, return a "NOT_AUTHENTICATED" error code.
e) Check that the ‘client’ in the policy matches the ‘instance’ in the "Authentication" data attribute. If not return an INVALID_CLIENT error code.
f) Check that the ‘origin’ in the "Authentication" data attribute matches the ‘origin’ in the policy. If not, return an INVALID_CLIENT_ORIGIN error code.
g) Check that the authentication method matches the requirement of the ‘method’ in the policy. If not, return an INVALID_CLIENT_METHOD error code.
h) Check if the policy requirement for user-id matches the ‘user-id’ in the "Authentication" data attribute. If not, return an INVALID_USER error code.
i) Check if the policy requirement for user-role matches the ‘user-role’ in the "Authentication" data attribute. If not, return an INVALID_ROLE error code.
j) Check that the ‘scope’ will allow the protected operation. If not, return the appropriate standard error code for the required scope or the EXTENDED_SCOPE_REQUIRED error code and the required extended scope identifier to be used with a "Hint" data attribute with the response.
k) Check the ‘type’ field of the ‘extension’ field. If it is unknown, return an UNKNOWN_EXTENSION error code.
l) If the extension ‘type’ is known but the extension ‘data’ is bad, return an INVALID_EXTENSION error code.

The signature is validated as follows:

1) Optionally consult a cache of previously received tokens to see if the signature has already been validated. If the result was a failure, then return an "INVALID_SIGNATURE" error code, else return success. Note that any cache of previous tokens must contain either the entire token exactly as received, or a hash of the token using a secure hash algorithm. A simple checksum or the retention of a few key fields is not sufficiently secure to detect tampering.
2) Look at the ‘key-id’ field in the token to see which key in the Authorization_Server property should be used for validation. A ‘key-id’ value of 1 will select ‘signing-key-1’ and a value of 2 will select ‘signing-key-2’.
3) Validate the ‘signature’ field against all octets coming before the ‘signature’ field in the BACnetAccessToken production using the selected public key.
4) Optionally store the token, or a secure hash of the token, for both success and failure, in a cache to avoid validation of the signature in the future.

17.4.9 Hints

When a protected operation fails, the client may need to know how to recover in an automated fashion. This usually means knowing what scope is needed. A "hint" data attribute returned with the error response can provide that information. For BACnet/SC, the Hint data attribute is conveyed in the ‘Hint’ Data Option. See Clause AB.2.3.X5.

When the EXTENDED_SCOPE_REQUIRED error code is returned, the responding device shall include a "Hint" data attribute with the error response that indicates the required scope.

For error responses that can include multiple failures, e.g., ReadPropertyMultiple, the returned hint shall include the authorization scope that will allow at least the first failed operation to succeed. It is a local matter whether the responding device tries to make a combined authorization scope that will allow all protected operations to succeed.

17.5 Conformance Requirements

The BACnet authentication and authorization mechanism allows flexibility in implementation while maintaining compatibility for all customer and site deployment choices.

17.5.1 Client Conformance

Initiating the AuthRequest service is an optional functionality in client devices. Auth-aware client devices can obtain access tokens by either actively using the AuthRequest service, or passively accepting access tokens configured into it by some local means.

Active token retrieval by the client can be advantageous in situations where the target device's requirements for authorization change. For example, if the target device is reconfigured to require a different scope for a given resource that the client is currently accessing, it will begin to reject the protected operation. If the client is able to automatically retrieve an access token from an online authorization server, then access can be automatically restored. This can also be used when a client begins to talk to a target for the first time and, through rejections, it "learns" what permissions are needed.

Passive configuration of access tokens can be advantageous in situations where the authorization server is not online, either because it is temporarily offline, or the site is designed such that the authorization server is a temporary function that is never expected to be online continuously. In this case, a "helper tool" retrieves the access token and configures them into the client device. It is a local matter how passive configuration is performed. Note that access tokens are not secrets and therefore can be configured "in the clear". See also Clause 12.11.X5.

These methods are not exclusive, and clients can be implemented to support both for flexible adaptation to deployment choices. The capabilities of client devices shall be declared on the PICS statement.

17.5.2 Client Helper Conformance

For client devices that do not retrieve their own access tokens, a helper tool can be used to retrieve the access tokens and configure them into the client by some means. To support site deployment choices for the authorization server function, a helper tool shall be capable of using the AuthRequest service to retrieve tokens. It is a local matter if the helper tool can also obtain tokens by some non-standard means. See Clause 17.5.4.

17.5.3 Target Conformance

To support the site deployment choices of token-based authorization and/or local-policy-based authorization, secure target devices that support authorization are required to support both methods, and both Authorization_ACL and Authorization_Server properties shall be present. The capacity of local policy configuration shall be declared on the PICS statement.

Note that the Authorization_ACL property can also be used by non-secure target devices to protect against misconfigured clients. In this case, support of token-based authorization is not possible and the Authorization_Server property is not used.

17.5.4 Authorization Server Deployment Options

Since target devices are required to support local policies in the Authorization_ACL property, the use of access tokens and an authorization server is not required on every site.

It is a site deployment choice whether the authorization server is always online as a reachable BACnet device. If long term tokens are created and no subsequent changes are made to target scope requirements, the authorization server is not needed to be online. Note that BACnet access tokens are "self-contained" and therefore do not require an active connection between the resource server and the authorization server for token validation.

Vendor-specific helper tools might retrieve tokens by some mechanism other than the standard AuthRequest service. See Clause 17.5.2. However, to support helper tools and clients from other vendors, all authorization servers are required to support the AuthRequest service as an online BACnet device when needed.

17.6 AuthRequest Service

When a client determines that it needs an access token for a particular device, it uses the AuthRequest service to request a token from an authorization server.

The determination of which authorization server to contact for a given target is a local matter, with the exception that a client is required to use its own authorization server by default. The ability for the client to determine a different authorization server, by reading the target device's Authorization_Server property or by some other means, is optional. Likewise, the ability of an authorization server to relay this request to another authorization server on behalf of the original requestor is also optional.

This service grants Access Tokens. This service does not require a secure datalink like BACnet/SC because tokens are non-confidential. The tokens are bound to a specific client so they cannot be used if stolen and they are signed so they cannot be forged.

The service explicitly specifies the client to bind the token to, so it naturally supports the "on behalf of" flow by "helpers" getting tokens to be used by other devices.

17.6.1 AuthRequest Service Structure

The structure of the AuthRequest service primitive is shown in Table 17-X1. The terminology and symbology used in this table are explained in Clause 5.6.

Table 17-X1. Structure of AuthRequest Service Primitives

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Access Token Request	U	U(=)		
Result(+)			S	S(=)
Access Token Response			U	U(=)
Result(-)			S	S(=)
Error Type			M	M(=)
Error Details			U	U(=)

17.6.2 Argument

This parameter shall convey the information for the AuthRequest confirmed service request. For future extensibility, this parameter is a construct consisting of multiple request types, each defined to be optional. All service invocations shall have at least one of these options present. Note that there is only one request option defined at this time, the "Access Token Request".

17.6.2.1 Access Token Request

This parameter specifies the individual parameters for the request for an access token bound to a specific client for a given authorization scope to be used for a particular audience.

Table 17-X2. Structure of 'Access Token Request' Parameters

Parameter Name	Req	Ind	Datatype
Client	M	M(=)	Unsigned
Audience	M	M(=)	Sequence of Integer
Scope	U	U(=)	BACnetAuthorizationScope
User ID	U	U(=)	Unsigned
User Role	U	U(=)	Unsigned
Extension	U	U(=)	BACnetExtension

The 'Client' parameter specifies the device instance of the client device that the token is to be bound to. Only a device authenticating with this instance can successfully use the token.

The 'Audience' parameter indicates where the token can be used. This contains a list of devices and/or groups represented as signed integer values and is required to contain at least one value. Positive numbers are device instances and negative numbers are groups, e.g., group 5 is represented as -5. Group number 1 is reserved to mean "all devices".

The 'Scope' parameter indicates the requested standard authorization scope for the given audience. The server can return a different set of authorizations than those that were requested. If the ‘Scope’ parameter is absent, the client is requesting the server to return the "default scope" configured for the combination of ‘Client’, ‘Audience’, ‘User ID’, and ‘User Role’. If such a default does not exist, the server shall return an error.

The optional 'User ID' parameter indicates which user the token should be bound to. The indication of "user" is determined by the client device when the access token is used. The authentication of human users is discussed in Clause 17.3.

The optional 'User Role' parameter indicates which role the token should be bound to. The indication of "role" is determined by the client device when the access token is used. The authentication of human users is discussed in Clause 17.3.

The optional 'Extension' parameter is for proprietary or experimental extensions. It contains a non-empty CharacterString that contains a URI controlled by the organization defining the extension. The URI is for type identification purposes only and is not required to be dereferenceable. Extensions are not assumed to be interoperable and servers receiving an extension type they do not understand shall reject the token request.

17.6.3 Result(+)

The 'Result(+)' parameter shall indicate that the service request succeeded. A successful result includes the following parameters. For future extensibility, this parameter is a construct consisting of multiple response types, each defined to be optional. All Result(+) service responses shall have at least one of these options present, and it shall correspond to the option present in the service request. Note that there is only one response option defined at this time, the "Access Token".

17.6.3.1 Access Token Response

This parameter, of type BACnetAccessToken, specifies the requested access token. The scope of the token might be less (contain fewer permissions) than that of the request. See Clause 17.4.7 for the definition of the members of this construct.

17.6.4 Result(-)

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

17.6.4.1 Error Type

This parameter shall be used to report a service execution errors that result in a failure to grant the request.

The 'Error Class' and 'Error Code' to be returned for specific situations are as follows:

Situation	Error Class	Error Code
The device is not configured to grant access tokens.	SERVICES	OPTIONAL_FUNCTIONALITY_NOT_SUPPORTED
The specified client has no policy.	SERVICES	UNKNOWN_CLIENT
The specified audience has no policy.	SERVICES	UNKNOWN_AUDIENCE
The server does not have any policies for the requested audience.	SERVICES	UNKNOWN_AUDIENCE
The server does not have any policies for the requested client.	SERVICES	UNKNOWN_CLIENT
The server does not recognize the requested scope.	SERVICES	UNKNOWN_SCOPE

A requested scope was not provided, and the server does not have a configured default scope matching the client, user, role, and audience.	SERVICES	NO_DEFAULT_SCOPE
The server does not have a policy that allows the requested combination of parameters	SERVICES	NO_POLICY

17.6.4.1 Error Description

This optional parameter can be used to report a human readable description of the error condition.

17.6.5 Service Procedure

After verifying the validity of the request, the responding BACnet-user shall attempt to grant the request based on the parameters provided. Note that there is only one possibility defined at this time, the "Access Token Request".

17.6.5.1 Access Token Request Procedure

This request is used to get an access token that will grant an authorization scope to a client for a set of target devices.

Note that this is only a "request" for a token. The authorization server decides which parts of this request will be granted and which parts will need to be modified. Following OAuth practice, if a client requests "too much scope", the authorization server will return a reduced scope token rather than returning an error. This is true even if the return scope has no privileges at all. However, if the client asks for any other aspect (audience, user, client, extension, etc.), that can't be granted, the authorization server will return an error.

The 'client' field is not optional and must be filled out. Since this service does not require security, and the client can't know that its request will flow through a path that maintains the 'Authentication' data attribute, there is no guaranteed way for the authorization server to know who the client is. Typically, this field will just contain the instance of the client itself, but it could be different if a "helper" is requesting the token "on behalf of" another client that will use the token.

The 'audience' list will typically contain the one device that the client is wishing to talk to. If the client knows it is talking to a group, it will include that in the audience. If the client needs to talk to all devices, it can request "Group 1", thus asking for the requested scope in "all devices". Again, it is up to the authorization server to decide if that is granted. It is typically best if the client asks for the minimum audience required and lets the authorization server decide to broaden its applicability if appropriate, based on its knowledge of what the client needs to do. However, the authorization server shall also be guided by the doctrine of least privilege, so broadening will likely be rare.

The 'scope' field is optional. The absence of the 'scope' fields indicates that the token requestor does not know the specific scope that is being requested and is therefore requesting that the authorization server return the "default scope" that matches the client, audience, user, and role (noting that user and role are also optional parameters). This can be useful for implementing simple policies for simple client devices that do not need to, or cannot, manage multiple tokens per target.

The server can return a scope in the token that is different from the requested scope. A token with a different scope might succeed if the authorization server intentionally issued the different scope based on its knowledge of the scopes in use in the target device and the expected range of operations of the client.

Since the client can look at the token's 'audience' as part of its strategy for picking a token from its cache for a given audience, the authorization server is not allowed to turn a requested device into a group if the client did not ask for a group to begin with. In other words, clients are either "group-aware" or not. Returning a group instead of a device could confuse a non-group-aware client.

When a request is denied, either by the return of a Result(-) or a reduction of scope, the specific behavior of the authorization server is a local matter. However, since such denial or reduction is either the result of an attempt of an unauthorized action (i.e., an attack) or, more likely, the result of a configuration change in a legitimate device that now needs a new policy to perform its legitimate function, some kind of notification to a human operator or authorization management system is required. The manner of such notification is a local matter with the exception that, since clients will likely retry the request periodically, it is

required that authorization servers provide features to rate limit repetition of identical notifications or provide a feature to silence specific notifications for a period of time to allow field devices to be reconfigured. The manner of notification shall be indicated on the PICS. See Annex A.

1 New BACnet/SC Options

135-2020*cp*-2 BACnet/SC Options to Support Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 defines new data options for authentication and authorization information that can accompany encapsulated NPDUs.

Additionally, a new destination option is defined to indicate that the SC peer is "auth aware".

[Change Table AB-3, p. 1386]
[Note to reviewer: These clause numbers start at "X2" to reduce the chance of errors since the assigned values will very likely start at 2]

Table AB-3 BVLC Header Options

Header Option Type	Numeric Header Option Type	Description
Secure Path	1	See Clause AB.2.3.1
Hello	*x2*	*See Clause AB.2.3.X2*
Source	*x3*	*See Clause AB.2.3.X3*
Hint	*x4*	*See Clause AB.2.3.X4*
Token	*x5*	*See Clause AB.2.3.X5*
Proprietary Header Option	31	See Clause ~~AB.2.3.2~~ *AB.2.3.[X5+1]*

[Add new Clauses AB.2.3.X2-X6, p. 1386]

AB.2.3.X2 Hello Header Option

The 'Hello' header option specifies the BACnet/SC protocol version and the capabilities of a connecting peer.

The 'Hello' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Must Understand' = 0, 'Header Data Flag' = 1, 'Header Option Type' = x2
Header Length	2-octets	Length of 'Header Data' field, in octets. Shall be 1
Header Data	1-octet	Required to include:
Capabilities	1-octet	8 reserved bit flags. Shall each be 0.

This header option, if present, shall be a destination option in the 'Destination Options' parameter of the BVLC messages 'Connect Request' and 'Connect Accept'.

This header option shall be included with the 'Connect Request' and 'Connect Accept' messages and shall only be included with those messages. An SC connection shall record the capabilities of the connecting peer and this information will remain for the lifetime of the connection. If the capabilities of a peer change, that peer shall drop the connection so that a new Hello Option can be sent upon reconnection.

Note that the presence of the 'Hello' option during the connect sequence implicitly indicates the capability of being "auth-aware", as defined by Clause 17.2.1. Therefore, there no explicit bit is defined for that capability.

AB.2.3.X3 Source Header Option

The 'Source' header option specifies the identity information for the sender of a message. The Auth Path field of this option specifies the trustworthiness of the device identity in the Device field.

When passed to or from the network layer, a 'Source' option is conveyed as data attribute with the encapsulated NPDU. It is designated as a "every segment" attribute. See Clause 17.3.2 for its definition and usage.

The 'Source' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 1, 'Header Data Flag' = 1, 'Header Option Type' = x3
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data	7-octets	Required to include:
Auth Path	1-octets	1 if never left chain of auth-ware devices, 0 otherwise
Device	3-octets	Device instance number, with most significant octet first, 4194303 if unknown
User ID	2-octets	User identifier, 0 if unknown. See Clause 17.4.5.
User Role	1-octet	User role, 0 if unknown. See Clause 17.4.5.

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

AB.2.3.X4 Hint Header Option

The 'Hint' header option provides information about what authorization is required for a failed operation. See Clause 17.4.9 for meaning and usage.

When passed to or from the network layer, a 'Hint' option is conveyed as data attribute with the encapsulated NPDU. It is designated as a "first segment" attribute.

A 'Hint' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 0, 'Header Data Flag' = 1, 'Header Option Type' = x4
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data	3-N octets	Required to include:
Scope	Variable	The required scope represented as the ASN.1 production of a BACnetAuthorizationScope

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

AB.2.3.X5 Token Header Option

The 'Token' header option conveys an access token to authorize a protected operation. See Clause 17.4.7 for meaning and usage.

When passed to or from the network layer, a 'Token' option is conveyed as data attribute with an encapsulated NPDU. It is designated as a "first segment" attribute.

A 'Token' header option consists of the following fields.

Header Marker	1-octet	'More Options' = 0 or 1, 'Every Segment' = 0, ' Header Data Flag' = 1, 'Header Option Type' = x5
Header Length	2-octets	Length of 'Header Data' field, in octets.
Header Data	N octets	Required to include:
Token	Variable	The ASN.1 production of a BACnetAccessToken

This header option, if present, shall be a data option in the 'Data Options' parameter of BVLC messages conveying an NPDU.

[Renumber Clause AB.2.3.2 to be at the end of the AB.2.3.x series, p. 1386]

~~AB.2.3.2~~ *AB.2.3.[X5+1]* Proprietary Header Option

2 New Device properties

135-2020*cp*-3 Device Object Properties to support Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 defines new properties for the Device object for configuring and monitoring the behavior of authorization clients and servers.

[Change Table 12-13, p. 222]

…	…	…
Authorization_Server	*BACnetAuthorizationConfiguration*	*O*[s]
Authorization_Status	*BACnetAuthorizationStatus*	*O*[s]
Authorization_ACL	*BACnetARRAY of BACnetAuthorizationPolicy*	*O*
Authorization_Groups	*BACnetARRAY of Unsigned*	*O*
Authorization_Cache	*BACnetARRAY of BACnetAccessToken*	*O*[t]
Authorization_Scopes	*BACnetARRAY of BACnetAuthorizationScopeDescription*	*O*
…	…	…

[s] *required for secure devices that perform server-side authorization*
[t] *required for secure devices that contain client tokens that are pre-configured by some local means*

[Add new clauses 12.11.X1-X6, p. 231]

12.11.X1 Authorization_Server

This property, of type BACnetAuthorizationServer, contains information about the authorization server that controls the authorization policies for this device.

The ‘auth-server’ field indicates the device instance of the authorization server for this device. Authorization clients can use this to know where to request an access token for this device using the AuthRequest service.

The ‘signing-key-1’ and ‘signing-key-2’ fields contain the public keys that can be used to validate the signature in access tokens. Only one key is required for operation and either field can be used for a single key. The ‘key-id’ field in the token selects which key is to be used. This allows for key rotation, if desired. For example, if the authorization server wishes to rotate keys, one of these fields can be updated with the future key, then the authorization begins using that key, and then the previous key can be removed.

12.11.X2 Authorization_Status

This read-only property, of type BACnetAuthorizationStatus provides information about this device’s current security posture and recent authentication and authorization activity.

The ‘posture’ field indicates the current security posture of the device. The enumerated values have the following meaning:

Enumeration Value	Meaning
open	no authorization is in effect
proprietary	proprietary authorization is in effect (temporary situation)
configured	everything looks good and is properly secured
misconfigured-partial	something is wrong, but things are mostly working
misconfigured-total	nothing works, may need reset to factory defaults

The ’error’ field is used when a known error condition can be indicated for a ‘misconfigured’ posture. If the error is known to be caused by a particular property of an object, then that shall be indicated in the ‘error-source’ field. If there are human readable

details of the error condition available, they shall be indicated in the ‘error-details’ field. If no error condition is in effect, or the reason is unknown, these fields shall be absent.

There are four variably sized lists of events that can be us used for diagnostics. The intent of the four lists is to be helpful to humans. The selection of which events to include, if any, is a local matter. When present, the events shall be in timestamp order, oldest record first.

Authentication events shall consist of the following fields:

Field Name	Data Type	R/O	Description
timestamp	BACnetDateTime	R	When the event occurred
peer	BACnetAuthenticationPeer	R	The peer’s address, identity, and capabilities
source	BACnetAuthenticationSource	R	From the "Source" data attribute
decision	BACnetAuthenticationDecision	R	allow or deny reason
decision-details	CharacterString	O	human readable details for the decision

Authorization events shall consist of the following fields:

Field Name	Data Type	R/O	Description
timestamp	BACnetDateTime	R	When the event occurred
address	BACnetAddress	R	SNET/SADR of originator
source	BACnetAuthenticationSource	O	From the "Source" data attribute, if provided
token	BACnetAccessToken	O	the access token, if provided
decision	BACnetAuthorizationDecision	R	allow or deny reason
decision-details	CharacterString	O	human readable details for the decision

The effective size of this property shall be dynamically adjusted based on the capabilities of the device reading this property. For example, if the reading device does not support segmentation, then the event lists shall be limited to that which can be conveyed in the response without segmentation. The ‘token’ field, while useful for diagnostics, can be omitted from BACnetAuthorizationEvent to reduce size.

12.11.X3 Authorization_ACL

This property, of type BACnetARRAY of BACnetAuthorizationPolicy, is a collection of pre-configured local policies containing the same information that would be delivered by access tokens across the wire. If no token is provided by a client device, this device can consult this local policy list to see what authorization has been configured for the client device.

The local policy capability supports earlier non-auth-aware client devices, both secure and non-secure, by effectively pre-sending authorization information to the resource server side, to be used when the client doesn't/can't send one on its own. This supports non-secure clients, secure clients that pass through non-secure network, and original BACnet/SC clients that are "Secure Path" capable but don't use access tokens. The primary usefulness for this is for resource servers that want to protect themselves with standard mechanisms but still want to be able to interact with earlier clients.

This property can be used by non-secure target devices. The only restriction for use by non-secure target devices is that the ‘method’ field is be limited to ‘any-method’, the ‘origin’ field can’t use ‘direct-connect’, and ‘user-id’ and ‘user-role’ must be absent.

See Clause 17.4.2 for a detailed description of the contents of a policy entry.

12.11.X4 Authorization_Scopes

This read-only property, of type BACnetARRAY of BACnetAuthorizationScopeDescription, is a way of describing the non-standard scopes that are in use in this device. It shall be present in devices that require the use of non-standard scopes. It shall not be present in devices that only use standard scopes.

Its presence allows an authorization server to read this information and populate its user interface with meaningful scope descriptions for the user to pick from.

The ‘name’ field is a short string restricted to the format that can be used as an OAuth "scope-token". See RFC 6749 Section A.4. The intent of the ‘name’ field is for interoperation with OAuth; therefore, the names should be meaningful to the extent possible.

The ’description’ field provides the human-readable description of the scope identifier. The description should be sufficiently detailed to provide guidance to the user as to its meaning. However, considering that it is expected to be used in user interfaces, possibly in tables or drop-down choices, it shall not contain newline characters and is not intended to be a lengthy description.

12.11.X5 Authorization_Cache

This read-only property, of type BACnetARRAY of BACnetAccessToken, is a list of access tokens that are in use by the client functions within this device. The tokens in this list have either been acquired by this device with the AuthRequest service or have been configured into this device by some local means.

This list is for diagnostic purposes. It is not required to contain every token that is in use. The management of this list is a local matter. BACnet clients reading this list should be aware that the items in the array can shift and change at any time.

12.11.X5 Authorization_Groups

This property, of type BACnetARRAY of Unsigned, indicates the list of group numbers that this device is a member of. The ‘audience’ field of an access token can specify one of these group numbers as an alias for this device.

3 New Data Structures

135-2020*cp*-4 Data Structures to support Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 defines new data structures used for configuring, monitoring, and conveying authentication and authorization information.

[Add the following productions to Clause 21.6, p. 882, in alphabetical order]

```
BACnetAuthorizationPolicy ::= SEQUENCE {
        not-before      [0] BACnetDateTime OPTIONAL,        -- local time
        not-after       [1] BACnetDateTime OPTIONAL,        -- local time
        client          [2] Unsigned,                       -- "who" this policy applies to
        origin          [3] BACnetAuthenticationOrigin,     -- where the client is located with respect to the target
        method          [4] BACnetAuthenticationMethod,     -- how the client authenticated
        user-id         [5] Unsigned OPTIONAL,              -- the user-id this policy applies to, absent for "any"
        user-role       [6] Unsigned OPTIONAL,              -- the user-role this policy applies to, absent for "any"
        scope           [7] BACnetAuthorizationScope,       -- the granted permissions
        extension       [8] BACnetExtension OPTIONAL
        }

BACneAuthenticationOrigin ::= ENUMERATED {-- these are hierarchical:
        any-network     (0),        -- any-network will also allow local network or direct connections
        local-network   (1),        -- local-network will also allow direct connections
        direct-connect  (2)         -- direct-connect only allows direct connections
        }

BACnetAuthenticationMethod :: ENUMERATED {      -- these are hierarchical:
        any-method      (0),        -- any-method will also allow secure-path and authenticated
        secure-path     (1),        -- secure-path will also allow authenticated
        authenticated   (2)         -- authenticated will only allow authenticated
        }

BACnetAuthorizationScope ::= SEQUENCE {
        standard        BIT STRING {  -- see standard scopes in Clause 17.4.4
                view            (0),    -- for viewing confidential information, not needed for public information
                adjust          (1),
                control         (2),
                override        (3),
                config          (4),
                bind            (5),
                install         (6),
                auth            (7),
                infrastructure  (8),
                reserved-9      (9),
                reserved-10     (10),
                reserved-11     (11),
                reserved-12     (12),
                reserved-13     (13),
                reserved-14     (14),
                reserved-15     (15),
                reserved-16     (16),
```



```
                reserved-17     (17),
                reserved-18     (18),
                reserved-19     (19),
                reserved-20     (20),
                reserved-21     (21),
                reserved-22     (22),
                reserved-23     (23)
                }
        extended        [0] SEQUENCE OF CharacterString OPTIONAL
        }

BACnetAccessToken ::= SEQUENCE {
        issuer          [0] Unsigned,                   -- device instance of authorization server issuing the token
        issued          [1] BACnetDateTime,             -- when the token was issued
        audience        [2] SEQUENCE OF Integer,        -- negative numbers are groups; group 1 is "all",
        policy          [3] BACnetAuthorizationPolicy,  -- contains the actual authorization information and client binding
        key-id          [4] Unsigned                    -- 1=signing-key-1, 2=signing-key-2
        signature       [5] OCTET STRING                -- digital signature of all the preceding ASN encoded octets
        }

BACnetAuthorizationConfiguration ::= SEQUENCE {
        auth-server     [0] Unsigned,               -- instance of authorization server for this device, 4194303=unconfigured,
        signing-key-1   [1] OCTET STRING OPTIONAL,  -- SubjectPublicKeyInfo in binary DER, absent=unconfigured
        signing-key-2   [2] OCTET STRING OPTIONAL,  -- SubjectPublicKeyInfo in binary DER, absent=unconfigured
        groups          [3] SEQUENCE OF Unsigned    -- groups that this device is a part of
        }

BACnetAuthorizationScopeDescription ::= {
        name            CharacterString,            -- usable as OAuth/JWT scope token
        description     CharacterString
        }

BACnetAuthorizationStatus ::= SEQUENCE {
        posture                 [0] BACnetAuthorizationPosture,
        error                   [1] Error OPTIONAL,
        error-source            [2] BACnetObjectPropertyReference OPTIONAL,
        error-details           [3] CharacterString OPTIONAL,
        authentication-success  [4] SEQUENCE OF BACnetAuthenticationEvent,
        authentication-failure  [5] SEQUENCE OF BACnetAuthenticationEvent,
        authorization-success   [6] SEQUENCE OF BACnetAuthorizationEvent,
        authorization-failure   [7] SEQUENCE OF BACnetAuthorizationEvent
        }

BACnetAuthorizationPosture ::= ENUMERATED {
        open                    (0),
        proprietary             (1),
        configured              (2),
        misconfigured-partial   (3),
        misconfigured-total     (4)
        }

BACnetAuthenticationEvent ::= SEQUENCE {
        timestamp               [0] BACnetDateTime,
        peer                    [1] BACnetAuthenticationPeer,
        source                  [2] BACnetAuthenticationSource,
        decision                [3] BACnetAuthenticationDecision,
```



```
        decision-details        [4] CharacterString OPTIONAL
        }

BACnetAuthenticationPeer ::= SEQUENCE {
        host            BACnetHostNPort,
        device          Unsigned,  -- 4194303 if unknown
        auth-aware      Boolean,
        router          Boolean,
        hub             Boolean
        }

BACnetAuthenticationSource ::= SEQUENCE {
        auth-path       Boolean,
        device          Unsigned,
        user-id         Unsigned,
        user-role       Unsigned
        }

BACnetAuthenticationDecision ::= ENUMERATED {
        allow-match     (0),
        deny-mismatch   (1),
        deny-non-router (2)
        }

BACnetAuthorizationEvent ::= SEQUENCE {
        timestamp       [0] BACnetDateTime,
        address         [1] BACnetAddress,  -- SNET and SMAC
        source          [2] BACnetAuthenticationSource OPTIONAL,
        token           [3] BACnetAccessToken OPTIONAL,
        decision        [4] BACnetAuthorizationDecision
        decision-details [5] CharacterString OPTIONAL
        }

BACnetAuthorizationDecision ::= ENUMERATED {
        allow-by-token          (0),
        allow-by-local-policy   (1),
        deny-no-token-or-policy (2),
        deny-not-before         (3),
        deny-not-after          (4),
        deny-target-device      (5),
        deny-target-group       (6),
        deny-target-application (7),
        deny-client-device      (8),
        deny-client-method      (9),
        deny-scope              (10),
        deny-extension          (11),
        deny-issuer             (12),
        deny-stale              (13),
        deny-revoked            (14),
        deny-signature          (15),
        deny-other              (16)
        }

BACnetExtension ::= SEQUENCE {
        type    CharacterString,                -- extension type (URI)
        data    [0] ABSTRACT-SYNTAX.&Type
```


}

[Change Clause 21.6, p. 882]

21.6 Base Types

...

BACnetServicesSupported ::= BIT STRING {

...

-- Remote Device Management Services

device-communication-control (17),
confirmed-private-transfer (18),
confirmed-text-message (19),
reinitialize-device (20),
-- who-Am-I (47),
-- you-Are (48),

-- Security Services

-- auth-request (x),

...

-- Services added after 2016

confirmed-audit-notification (44), -- Alarm and Event Service
audit-log-query (45), -- Object Access Service
unconfirmed-audit-notification (46), -- Alarm and Event Service
who-Am-I (47), -- Remote Device Management Service
~~you-Are (48) -- Remote Device Management Service~~
you-Are (48), -- Remote Device Management Service

[note to reviewer: add comma to the above line]

-- Services added after 2020

auth-request (x)

...

}

...

[Change Clause 21.2, p. 858]

BACnetConfirmedServiceChoice ::= ENUMERATED {

...

-- Remote Device Management Services

device-communication-control (17),
confirmed-private-transfer (18),
confirmed-text-message (19),
reinitialize-device (20),

-- *Security Services*

auth-request (x),

...

-- Services added after 2016

-- confirmed-audit-notification (32) see Alarm and Event Services
-- audit-log-query (33) see Object Access Services

-- *Services added after 2020*

-- auth-request (x) see Security Services

...


```
BACnet-Confirmed-Service-Request ::= CHOICE {
...
-- Remote Device Management Services
        device-communication-control    [17]  DeviceCommunicationControl-Request,
        confirmed-private-transfer      [18]  ConfirmedPrivateTransfer-Request,
        confirmed-text-message          [19]  ConfirmedTextMessage-Request,
        reinitialize-device             [20]  ReinitializeDevice-Request,

-- Security Services
        auth-request                    [x] AuthRequest-Request,
...
-- Services added after 2016
        -- confirmed-audit-notifications [32]  see Alarm and Event Services
        -- audit-log-query              [33]  see Object Access Services
        }

-- Services added after 2020
        -- auth-request                 [x] see Security Services
 ...

BACnet-Confirmed-Service-ACK  ::= CHOICE {
...

-- Remote Device Management Services
        confirmed-private-transfer      [18]  ConfirmedPrivateTransfer-ACK,

-- Security Services
        auth-request                    [x]   AuthRequest-ACK,
...
```

[Add new Clause 21.2.5, p. 866]

21.2.X Authentication and Authorization Services

```
AuthRequest-Request ::= CHOICE {                -- CHOICE of the "sub service" allows future extensibility
        token-request       [0] SEQUENCE {
                client              [0] Unsigned,                       -- client device instance to bind the token to
                audience            [1] SEQUENCE OF Integer,            -- target device(s) and/or group(s).
                scope               [2] BACnetAuthorizationScope,       -- requested scope
                user-id             [3] Unsigned,                       -- 0 if unknown, see 17.4.5
                user-role           [4] Unsigned,                       -- 0 if unknown, see 17.4.5
                extension           [5] BACnetExtension OPTIONAL
                }
        }

AuthRequest-ACK ::= CHOICE
        token-response     [0] BACnetAccessToken
        }
```

[Change Clause 21.4, p 872]

```
...
BACnet-Error ::= CHOICE {
...
-- Remote Device Management Services
```



```
        device-communication-control    [17]  Error,
        confirmed-private-transfer      [18]  ConfirmedPrivateTransfer-Error,
        confirmed-text-message          [19]  Error,
        reinitialize-device             [20]  Error,

-- Security Services
        auth-request                    [x]   AuthRequest-Error,
...
-- Services added after 2016
        -- confirmed-audit-notification [32]  see Alarm and Event Services
        -- audit-log-query              [33]  see Object Access Services
        }

-- Services added after 2020
        -- auth-request                 [x]   see Security Services
...
        }
```

[Add new codes to the Error production, interspersing new items in alphabetical order]

```
Error ::= SEQUENCE {
        ...
        error-code          ENUMERATED { -- see below for numerical order
                              ...
                              adjust-scope-required                (n),
                              ...
                              auth-scope-required                  (n+1),
                              ...
                              bind-scope-required                  (n+2),
                              ...
                              config-scope-required                (n+3),
                              ...
                              control-scope-required               (n+4),
                              ...
                              extended-scope-required              (n+5),
                              ...
                              incorrect-client                     (n+6),
                              ...
                              install-scope-required               (n+7),
                              ...
                              insufficient-scope                   (n+8),
                              ...
                              no-default-scope                     (n+9),
                              ...
                              no-policy                            (n+10),
                              ...
                              override-scope-required              (n+11),
                              ...
                              unknown-audience                     (n+12),
                              ...
                              unknown-client                       (n+13),
                              ...
                              unknown-scope                        (n+14),
                              ...
                              view-scope-required                  (n+15),
                              ...
```



```
-- numerical order reference
                                -- see other                          (0),
                                ...
                                -- see adjust-scope-required         (n)
                                -- see auth-scope-required           (n+1)
                                -- see bind-scope-required           (n+2)
                                -- see config-scope-required         (n+3)
                                -- see control-scope-required        (n+4)
                                -- see extended-scope-required       (n+5)
                                -- see incorrect-client              (n+6)
                                -- see install-scope-required        (n+7)
                                -- see insufficient-scope            (n+8)
                                -- see no-default-scope              (n+9)
                                -- see no-policy                     (n+10)
                                -- see override-scope-required       (n+11)
                                -- see unknown-audience              (n+12)
                                -- see unknown-client                (n+13)
                                -- see unknown-scope                 (n+14)
                                -- see view-scope-required           (n+15)
                            }
        ...
        }

...
WritePropertyMultiple-Error ::= SEQUENCE {
        error-type                      [0]   Error,
        first-failed-write-attempt      [1]   BACnetObjectPropertyReference
        }

AuthRequest-Error ::= SEQUENCE {
        error           Error,
        error-details   CharacterString OPTIONAL
        }
...
```


4 Error Codes

135-2020*cp*-5 Error Codes to support Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 defines new error situations that need new error code definitions.

[Change Clause 18.5, p. 793]

18.5 Error Class - SECURITY

...

BAD_SIGNATURE - The signature in a secure message is incorrect. ~~(Reserved for future use)~~

...

SOURCE_SECURITY_REQUIRED - The operation requested requires that the source secure or encrypt the request. ~~(Reserved for future use)~~

...

UNKNOWN_AUTHENTICATION_TYPE - The authentication method in a secure message is unknown to the receiving device. ~~(Reserved for future use)~~

[Add new error codes to Clause 18.5, Error Class - SECURITY, p. 793, in alphabetical order]

INCORRECT_AUDIENCE - The audience specified in an access token does not match the recipient.

INCORRECT_CLIENT - The client specified in an access token does not match the sender.

INSUFFICIENT_SCOPE - The protected resource requires an authorization scope that was not provided.

VIEW_SCOPE_REQUIRED - The protected resource requires the ‘view’ scope.

ADJUST_SCOPE_REQUIRED - The protected resource requires the ‘adjust’ scope.

CONTROL_SCOPE_REQUIRED - The protected resource requires the ‘control’ scope.

OVERRIDE_SCOPE_REQUIRED - The protected resource requires the ‘override’ scope.

CONFIG_SCOPE_REQUIRED - The protected resource requires the ‘config’ scope.

BIND_SCOPE_REQUIRED - The protected resource requires the ‘bind’ scope.

INSTALL_SCOPE_REQUIRED - The protected resource requires the ‘install’ scope.

AUTH_SCOPE_REQUIRED - The protected resource requires the ‘auth’ scope.

EXTENDED_SCOPE_REQUIRED - The protected resource requires a non-standard scope.

[Add new error codes to Clause 18.6, Error Class - SERVICES, p. 795, in alphabetical order]

UNKNOWN_AUDIENCE - There is no policy defined for the given audience.

UNKNOWN_CLIENT - There is no policy defined for the given client.

UNKNOWN_SCOPE - The requested scope is not recognized for the target device.

NO_DEFAULT_SCOPE - There is no default scope for the requested client and audience.

NO_POLICY - There is no policy defined for the requested operation.

5 PICS changes

135-2020*cp*-6 PICS statements to support Authentication and Authorization capabilities

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 defines new capabilities for devices that need to be indicated on a PICS statement.

[Add new section to Annex A, p.]

ANNEX A - PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT (NORMATIVE)

...

Authentication Options:

If this product is an authentication server, describe the methods by which administrators are notified of denied authorization requests and the options for filtering, pacing, and silencing such notifications:

_____

If this product is an authorization client,

- *Can it use the AuthRequest service to obtain its own tokens dynamically? ____*
- *Can it be statically configured with access tokens? ____ If so, how many? _____*

If this product is an authorization target, how many local polices can be configured? ___

6 Clause 3 changes

135-2020*cp*-7 New definitions for Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 uses new abbreviations.

[Add new abbreviations to Clause 3.3, p. 17]

3.3 Abbreviations and Acronyms Used in this Standard

...
ACL *Access Control List*
...
VPN *Virtual Private Network*
...
VLAN *Virtual Local Area Network*
...

7 BIBBs and Profiles

135-2020*cp*-8 New BIBBs and Profiles for Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 describes new services and behaviors that need to be described with BIBBs and Profiles.

[Add new Clause K.X, p. 1117]

K.X Authentication and Authorization BIBBs

K.X.1 BIBB - Dynamic Token Retrieval - A (AA-DT-A)

The A device is an authorization client that is capable of getting its own access tokens from an authorization server. When a protected operation fails with an authorization error code like CONFIG_SCOPE_REQUIRED, a device claiming conformance to AA-TR-A will attempt to retrieve an access token for the required scope from the authorization server configured into its Authorization_Server property. See Clause 17.

BACnet Service	Initiate	Execute
AuthRequest	x	

Devices claiming conformance to this BIBB are interoperable with devices claiming conformance to AA-AS-B and AA-RS-B.

K.X.2 BIBB - Static Token Configuration - A (AA-ST-A)

The A device is an authorization client that accepts configuration of access tokens by a helper tool. The manner of such configuration is a local matter. The capacity of such configuration shall be claimed on the PICS. The device shall support a minimum capacity to hold two access tokens. The device shall support the Authorization_Cache property with a minimum capacity of two tokens. However, it is recommended that the Authorization_Cache have the capacity to indicate the same number of tokens that can be configured. See Clause 17.

Devices claiming conformance to this BIBB are interoperable with devices claiming conformance to AA-RS-B.

K.X.3 BIBB - Resource Server - B (AA-RS-B)

The B device is an auth-aware resource server that has protected operations that require authorization. The B device has an Authorization_Server property that supports both of the signing keys and a group membership list with at least two members. The Authorization_ACL property shall be present and support at least four local policies. Support for all optional policy fields except ‘extension’ is required. See Clause 17.

The B device shall process access tokens presented to it by an authorization client.

The B device shall generate appropriate authorization error codes like CONFIG_SCOPE_REQUIRED. If the device supports non-standard scopes, then the device shall also generate "Hint" data attribute for the error responses and the Authorization_Scopes property shall be present.

Devices claiming conformance to this BIBB are interoperable with devices claiming conformance to AA-DT-A and AA-ST-A.

K.X.4 BIBB - Non-secure Resource Server - B (AA-NRS-B)

The B device is a non-secure device that nonetheless supports authorization policies to create an "allow list" for protected operations to defend against misconfigured devices. The Authorization_ACL property shall be present and support at least four local policies. Support for all optional policy fields except ‘extension’, ‘user-id’, and ‘user-role’ is required. The ‘method’ field is restricted to containing only the ‘any-method’ value. See Clause 17.

K.X.5 BIBB - Authorization Server - B (AA-AS-B)

The A device is an authorization server that is capable of issuing access tokens to authorization clients or their helpers. See Clause 17.

BACnet Service	Initiate	Execute
AuthRequest		x

Devices claiming conformance to this BIBB are interoperable with devices claiming conformance to AA-DT-A.

[Add new Clause L.X, p. 1143]

L.X Authentication and Authorization Profiles

The following table indicates which BIBBs shall be supported by the device types of this family, for each interoperability area.

Data Sharing

B-AS
DS-RP-B

Alarm & Event Management

B-AS

Scheduling

B-AS

Trending

B-AS

Device & Network Management

B-AS
DM-DDB-B
DM-DOB-B
DM-DCC-B

Authentication & Authorization

B-AS
AA-AS-B

L.X.1 BACnet Authorization Server (B-AS)

The B-AS is an authorization server that issues access tokens to authorization clients or their helpers. The server has a persistent database of access policies configured to meet the installation’s needs. The server has a User Interface suitable for configuring the authorization policies for the tokens that it issues. Support for all optional policy fields except ‘extensions’ is required. See Clause 17.

8 Examples

135-2020*cp*-9 Examples for Authentication and Authorization

Rationale

The BACnet Authentication and Authorization framework defined in Clause 17 describes new services and behaviors. Examples are created to provide clarity to the various interactions' patterns and scenarios.

[Add new Annex XX, p. 14xx]

ANNEX XX - EXAMPLES OF AUTHENTICATION AND AUTHORIZATION (INFORMATIVE)
(This annex is not part of this standard but is included for informative purposes only.)

These are examples of the various flows of data used in authentication and authorization. In many cases, only the interesting items are shown for clarity. For example, "Secure Path" option is assumed. The absence of any required items should not be considered significant.

The following abbreviations are used in the examples:

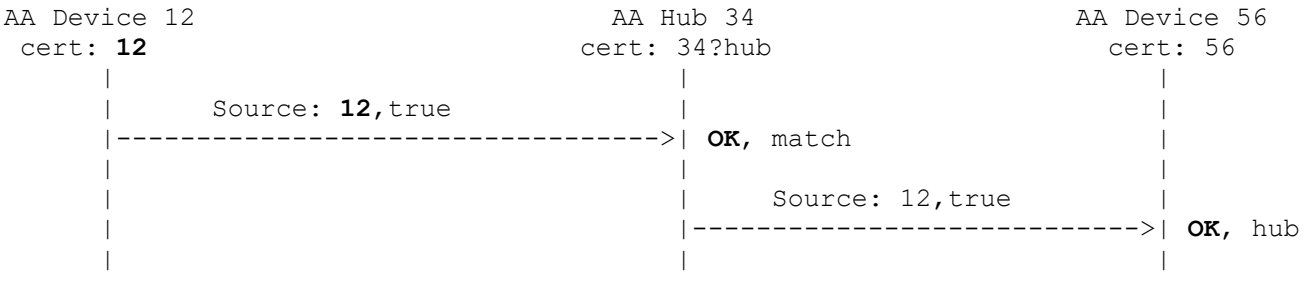
AA	an auth-aware device, see Clause 17.2.1
non-AA	a non-auth-aware device, see Clause 17.2.1
cert	this shows just the "..." part of the "bacnet://..." Subject Alternative Name URI
n/a	not applicable to the example; may or may not be present and its value is not important

XX.1 Authentication Examples

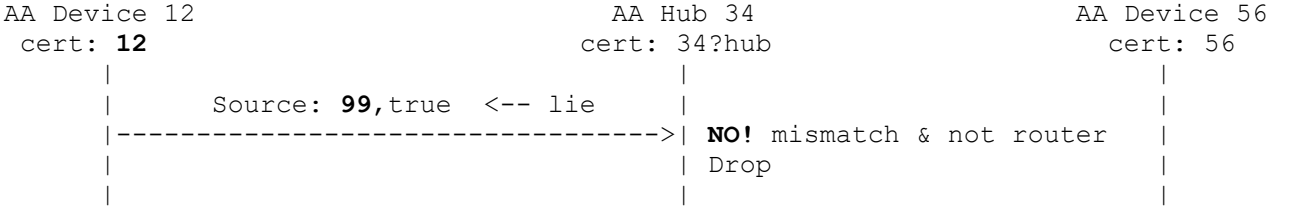
The following examples show the flow of information for authenticating a client to a target, either directly or relayed through trusted intermediaries.

XX.1.1 AA to AA interactions

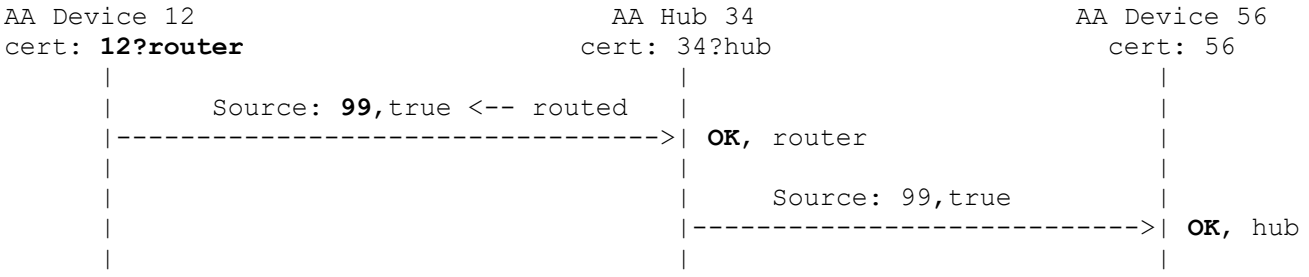
AA hub receives matching source from device --> success
AA destination receives matching source from hub --> success



AA hub receives mismatched source from device --> failure

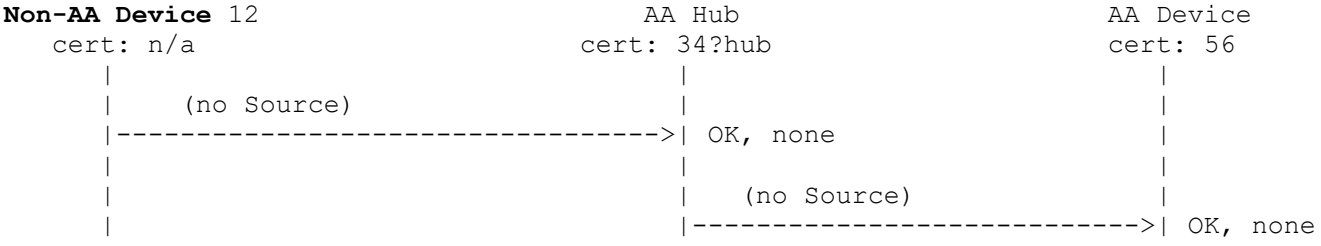


AA Hub receives mismatched source from router --> success

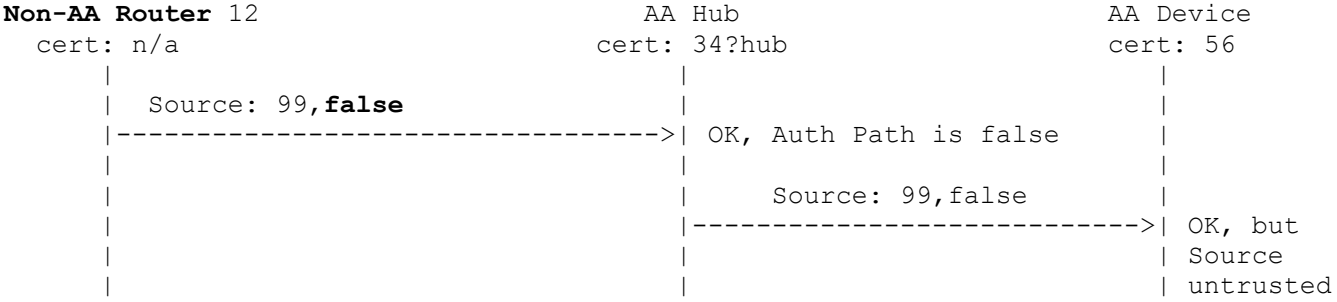


XX.1.2 non-AA to AA interactions

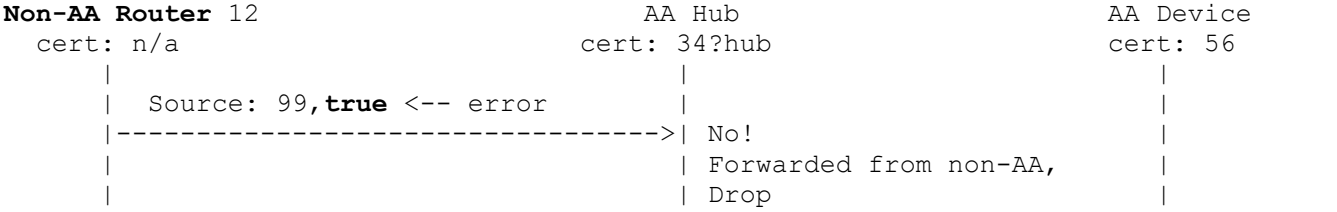
AA hub receives plain NPDU from non-AA device --> success, forwards
AA device receives plain NPDU from AA hub --> success (but no authorization)



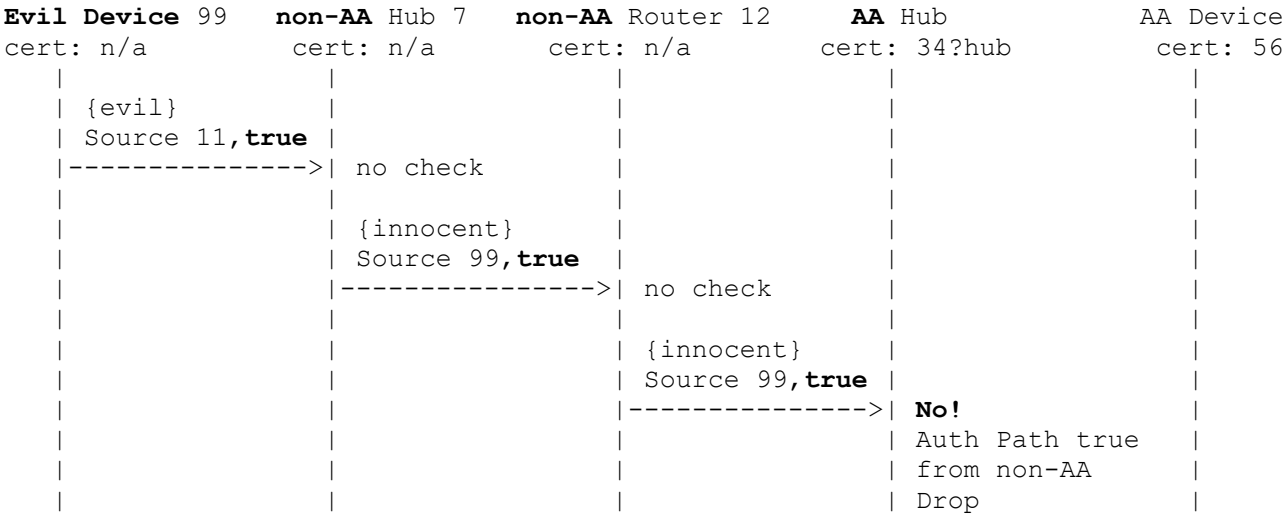
AA hub receives routed Auth Path false from non-AA router --> success, forwards
AA destination receives Auth Path false from hub --> success, but untrusted Source



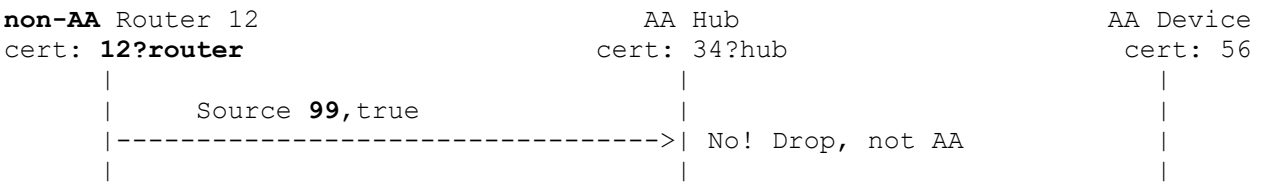
AA hub receives Auth Path true from non-AA router --> failure!



An evil device trying to sneak an Auth Path true through a non-AA path will fail, eventually. It is temporarily a lie in transit, but it will be dropped at the first AA peer because there is no innocent way for this to happen. Since AA peers will set Auth Path to false before sending to a non-AA peer, the only way it can be sent as true by a non-AA peer is a hack upstream.

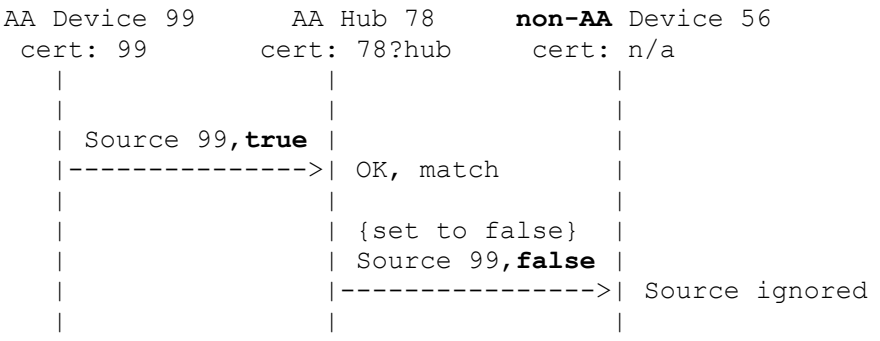


A non-AA peer with a new cert is still a non-AA peer. It can't/shouldn't route Auth Path true. Just because a non-AA peer has been given a new cert does not make it Auth Aware.

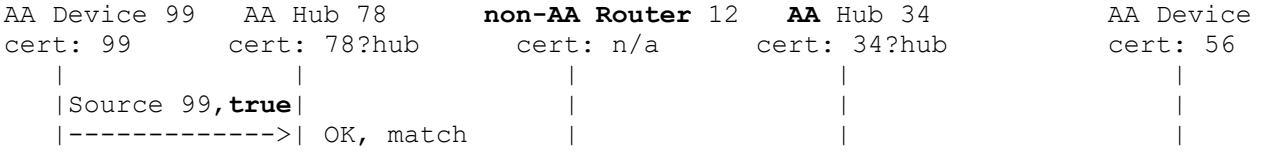


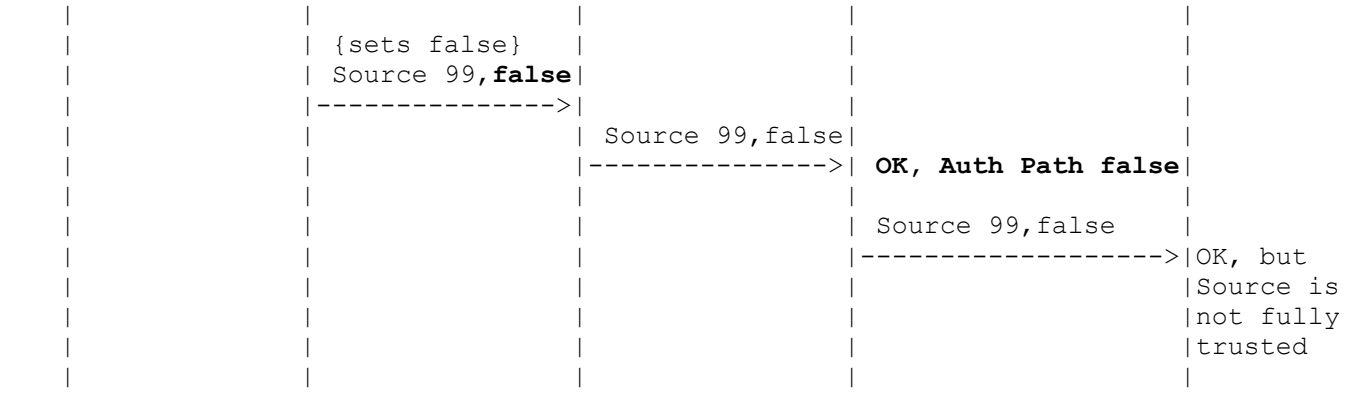
XX.1.3 AA to non-AA interactions

AA hub changes Auth Path to false on the way to non-AA end device. The presence of the Source option is meaningless since this is the final destination, but the hub does not remove it because the non-AA device might be a router to a downstream AA device that wants Source even with Auth Path false.



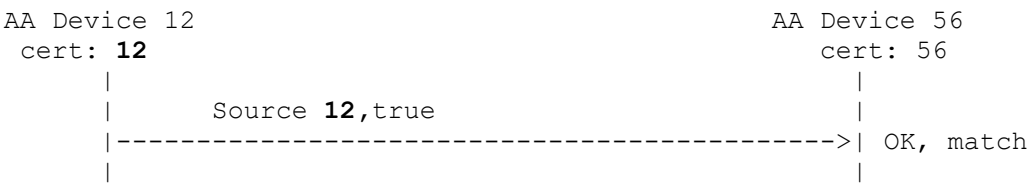
AA hub changes Auth Path to false on the way to non-AA router who forwards it on without knowing anything about it.
The next AA hub accepts it from the non-AA router because Auth Path is false.
The final destination gets Source with Auth Path false. This might work with its policies.





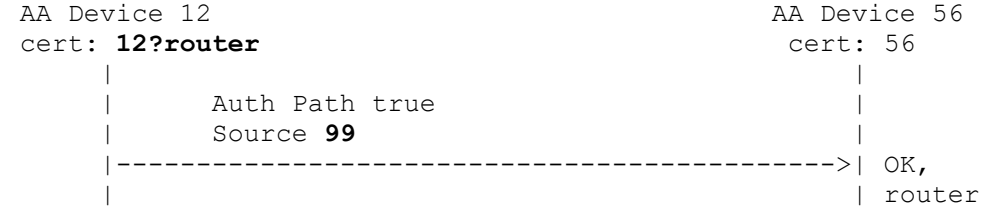
XX.1.4 Direct Connect AA to AA interactions

AA peer sends matching Source --> success.



AA non-routing peer sends mismatched Source --> failure.

AA non-routing peer sends mismatching Source --> success



XX.1.5 Direct Connect AA to non-AA interactions

AA peer sends whatever it wants to a non-AA peer --> "success", options ignored.
In Direct Connect, the AA sending peer *could* choose to not send Auth Path and Source, knowing that the receiving peer will ignore them. But for stack simplicity, following the rules without exception puts the options there even if they are not needed.


```
AA Device 12                                    non-AA Device 56
 cert: 12                                          cert: n/a
      |                                              |
      |      Source 12,false                         |
      |--------------------------------------------->| OK, ignored
      |                                              |
```

AA non-routing peer sends mismatched Source --> still success, ignored.

```
AA Device 12                                    non-AA Device 56
 cert: 12                                          cert: n/a
      |                                              |
      |      Source 99,false                         |
      |--------------------------------------------->| OK, ignored
      |                                              |
```

AA non-routing peer sends mismatching Source --> still success, still ignored

```
  AA Device 12                                  non-AA Device 56
 cert: 12?router                                    cert: 56
      |                                              |
      |      Source 99,false                         |
      |--------------------------------------------->| OK, ignored
      |                                              |
```

XX.1.6 Direct Connect non-AA to AA interactions

Non-AA peer sends Auth Path false to AA peer --> success, but not fully trusted

```
non-AA Router 12                                 AA Device 56
 cert: n/a                                          cert: 56
      |                                              |
      |      Source 99,false                         |
      |--------------------------------------------->| OK, Auth Path is false
      |                                              | Source not fully trusted
```

Non-AA router sends Auth Path true to AA device --> failure.
There is no way for the Auth Path to be originated from a non-AA device and no legitimate way for Auth Path true to be sent through a non-AA router.

```
non-AA Router 12                                AA Device 56
 cert: n/a                                          cert: 56
      |                                              |
      |      Source 99,true                          |
      |--------------------------------------------->| No! Auth Path is true
      |                                              | from non-AA peer
      |                                              | Drop
```

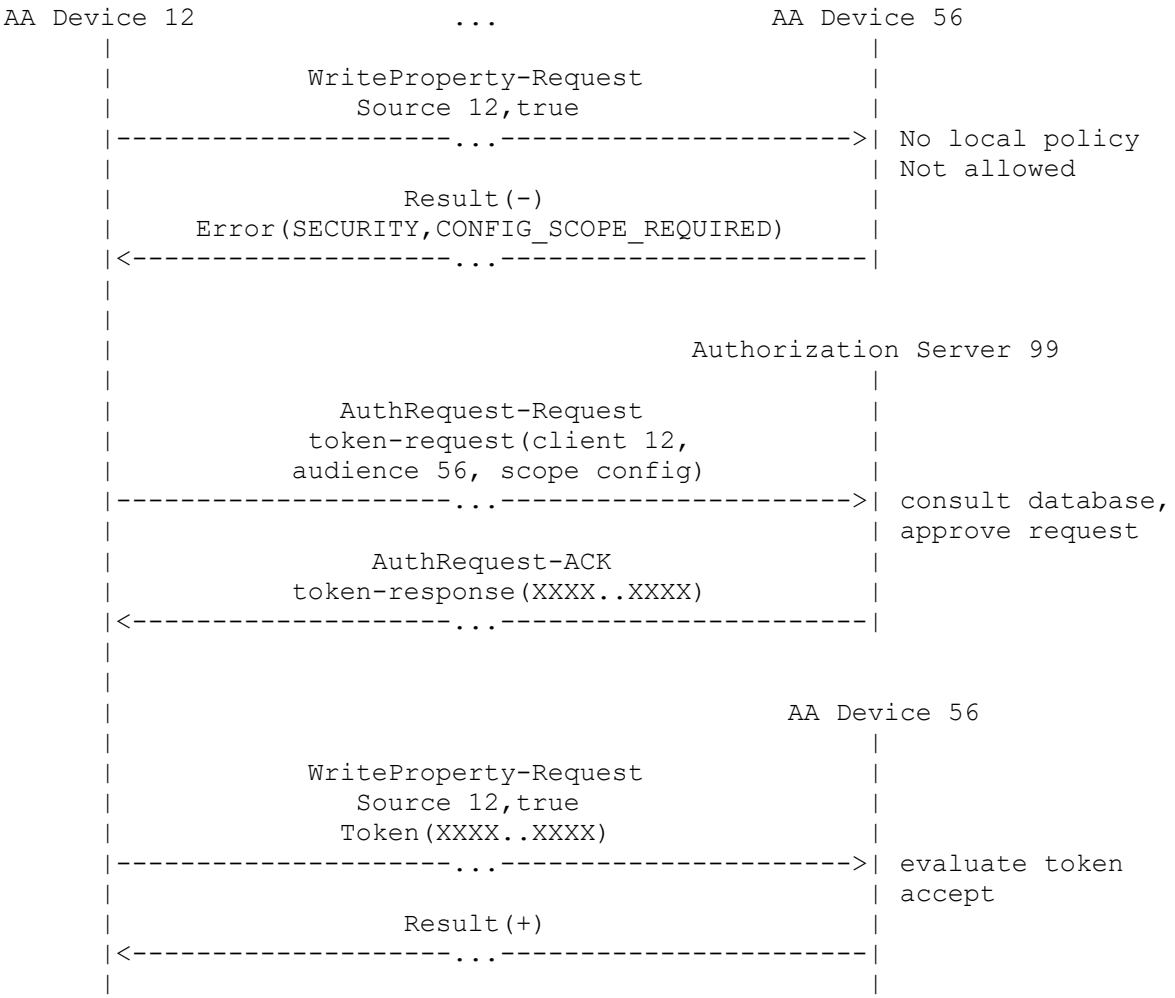
Non-AA router with "router" certificate is still rejected --> failure.
Giving a non-AA device a new certificate does not make it Auth Aware.

XX.2 Authorization Examples

The following examples show the flow of information for authorizing the operations of an authenticated client by the target device. In these diagrams, an ellipsis in the message arrows indicates that it is not material to the example whether the path is direct or relayed through hubs and routers.

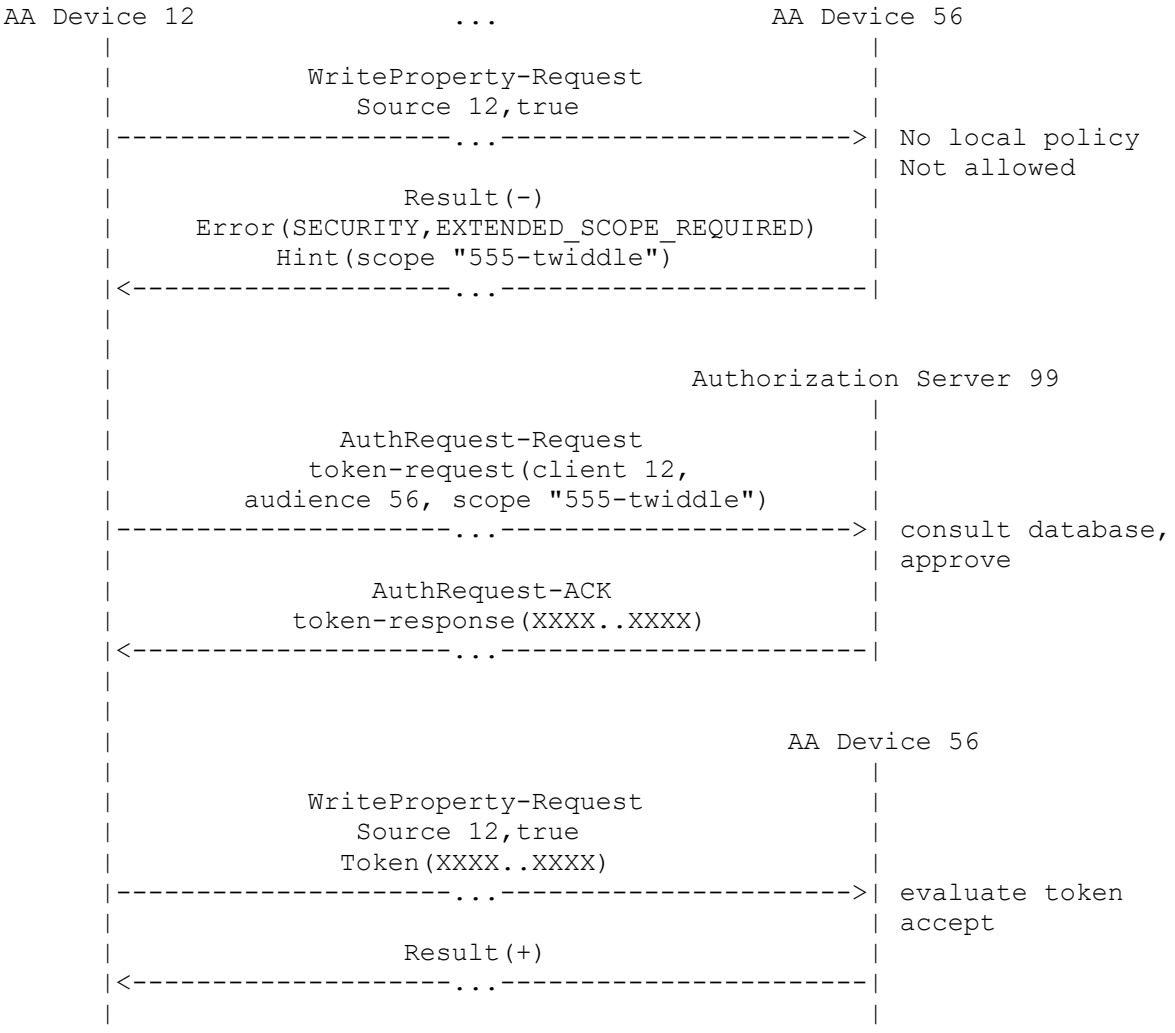
XX.2.1 Typical Token request flow

The client device does not have a local policy configured at the target. So, when a protected operation is attempted, it is denied. The client uses the error code to know what kind of access token to request for the protected operation. The client requests a token with that scope and is granted a token. The client then retries the protected operation, this time presenting the token, and the operation succeeds.



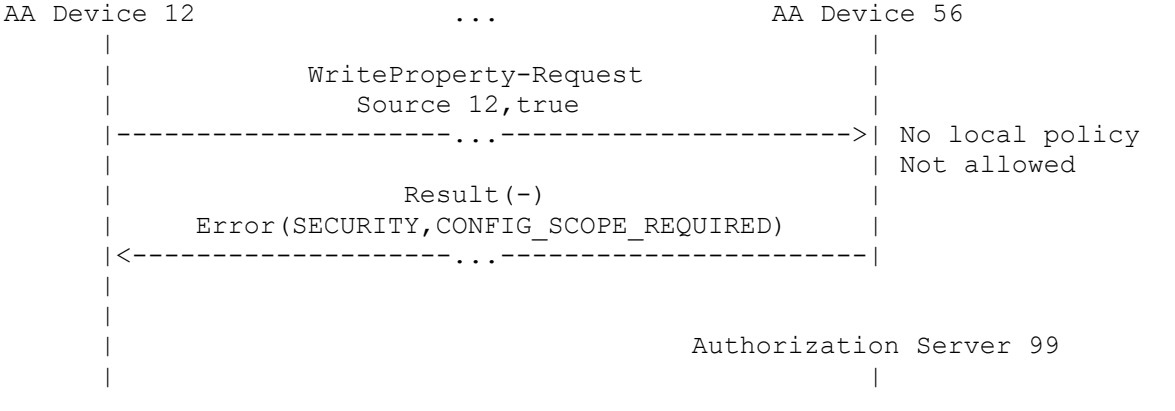
XX.2.1 Extended Scope

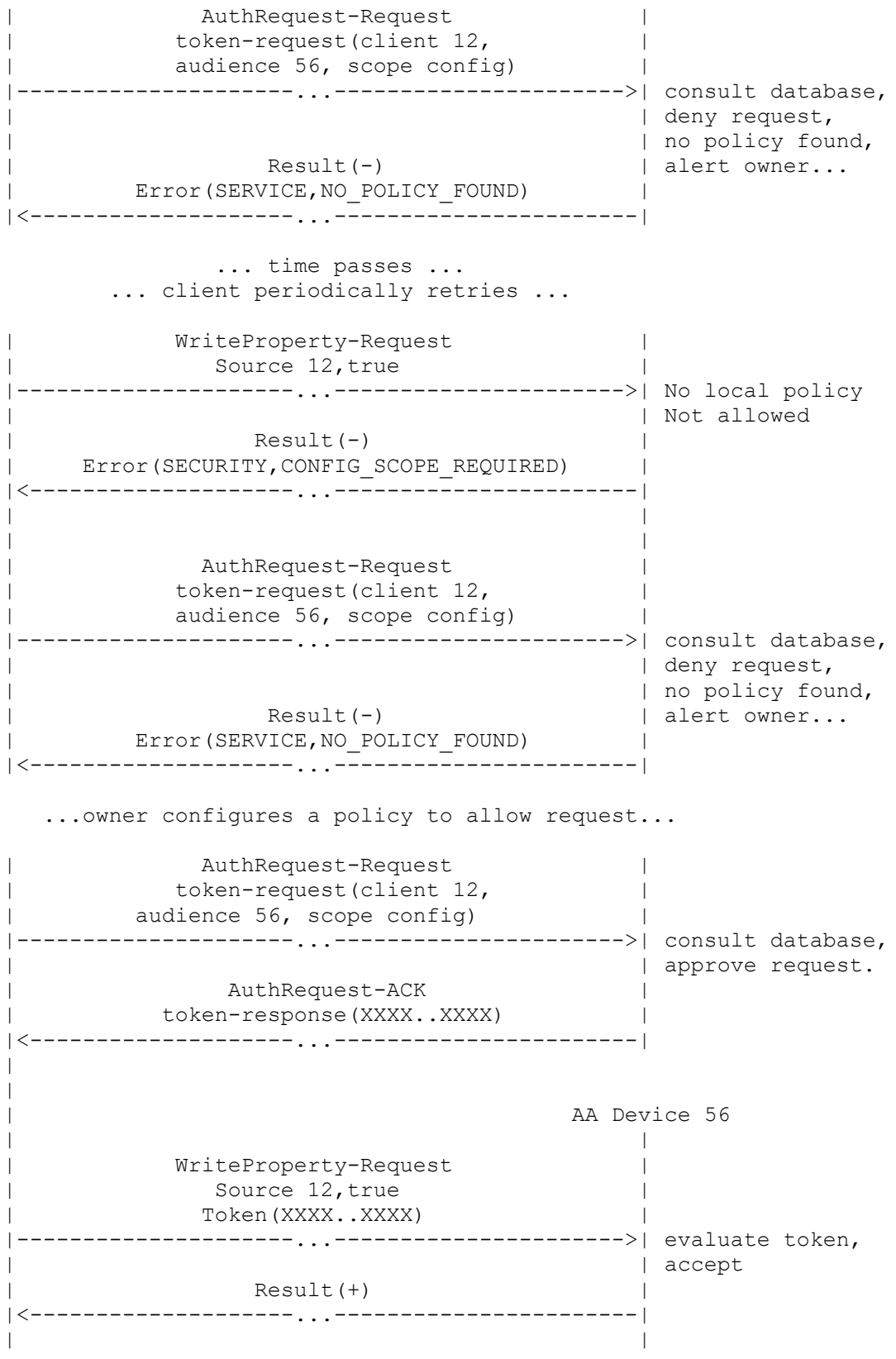
The target device requires an extended (non-standard) scope identifier for a protected operation. It returns that requirement in a Hint data attribute.



XX.2.1 No Policy Exists, Resolved with Authorization Server

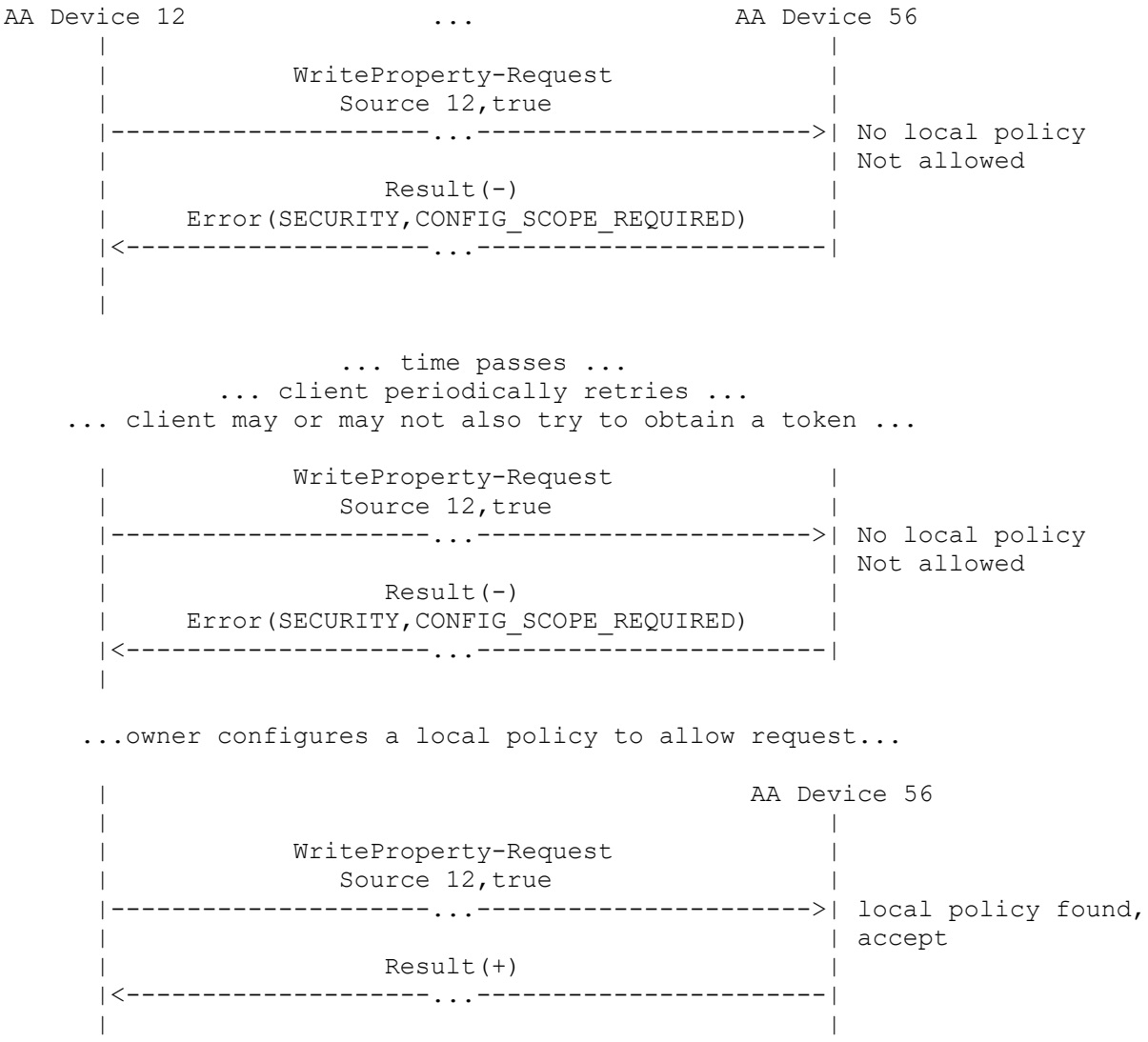
The Authorization server does not have a policy for a requested token, so there is a delay while the owner configures a policy to allow the token to be issued. Client also retries original operation in case a local policy has been created.





XX.2.1 No Policy Exists, Resolved with Local Policy

The resource server does not have a policy for a requested operation, so there is a delay while the owner configures a local policy in the Authorization_ACL property to allow the operation to succeed.



[Add a new entry to **History of Revisions**, p. 1429]

(This History of Revisions is not part of this standard. It is merely informative and does not contain requirements necessary for conformance to the standard.)

HISTORY OF REVISIONS

. . .	. . .	**. . .**
1	X	**Addendum *xx* to ANSI/ASHRAE Standard 135-2020** Approved by ASHRAE on MONTH DAY, 20XX; and by the American National Standards Institute on MONTH DAY, 20XX. 1. **Addition of Authentication and Authorization** **2. BACnet/SC Options to Support Authentication and Authorization** **3. Device Object Properties to support Authentication and Authorization** **4. Data Structures to support Authentication and Authorization** **5. Error Codes to support Authentication and Authorization** **6. PICS statements to support Authentication and Authorization capabilities.** **7. New definitions for Authentication and Authorization** **8. New BIBBs and Profiles for Authentication and Authorization** **9. Examples for Authentication and Authorization**

