bacnet experts

PolarSoft® Inc.
914 South Aiken Ave
Pittsburgh PA 15232-2212
412-683-2018 voice
info@polarsoft.com
www.polarsoft.com

# Change of Value in BACnet

David Fisher

10-Feb-2017

TUTORIAL

## Contents



# Change of Value in BACnet

10-Feb-2017

David Fisher

## Introduction

Many BACnet clients have the need to monitor the moment-to-moment value of an object property in some object in some device. This is easy to do using the ReadProperty service in BACnet. But often values don't change frequently, or it's unpredictable when they will change. That forces the client to read the value perhaps over and over even though the value hasn't actually changed. What's worse, is that when there are a lot of values that need to be monitored, the client produces more and more traffic on the network in order to reread these values, even though many of them haven't changed either. Generally the problem doesn't scale well in larger systems.

To help mitigate these situations, BACnet includes a concept that is called *change of value* (COV). The idea is that a client that is interested in a value, tells the server that owns the object property whose value is needed, that it is interested in knowing any time the value changes by a certain amount from the last time it was reported to the client. This is called a *subscription* and works much like a magazine subscription. Once subscribed, the server worries about detecting whether the value has changed, and when it changes by more than the subscription's designated amount, then a special *COVnotification* message is sent to the subscribing client that includes the new value. This technique can substantially reduce the amount of network traffic under normal circumstances.

BACnet supports several types of COV. This paper discusses each type, how it works, and issues that may occur when implementing and using COV in general.

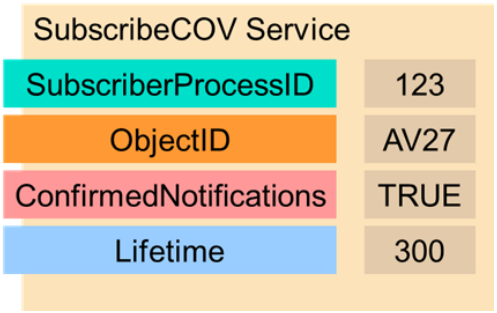
## COV Types

In the scheme of COV there are two types of devices:

- Some device that can detect changes of value for one or more objects (a server) and
- Some device(s) that are interested in knowing when these values change (a client)

BACnet provides two kinds of approaches to solve this problem, and devices are free to not implement them at all, or to implement one approach, or both.

The first approach is called *Subscription*. The client device sends a message called SubscribeCOV to the server. The subscription identifies what object is of interest in the server device, how long the subscription should last, and how changes of value should be reported back to the client.




When a change is detected, the server must notify the client that the change has occurred. Clients are allowed to group notifications together so that potentially different processes in the client can manage them, e.g. maintenance notifications vs. lifesafety notifications.

When the client issues a subscription it provides a "process ID" to the server. This is a number that presumably has significance to the client.

In the example above the processID is 123. The client is interested in the object AV-27. Because it's a standard object type (AV) the standard defines which property is the one that will be checked for changes (Present_Value in this case). The amount by which the Present_Value must change before a notification is the COV_Increment property of the AV object. When the change occurs the subscription says that the server should issue a ConfirmedCOVNotification. The subscription should last for 300 seconds.

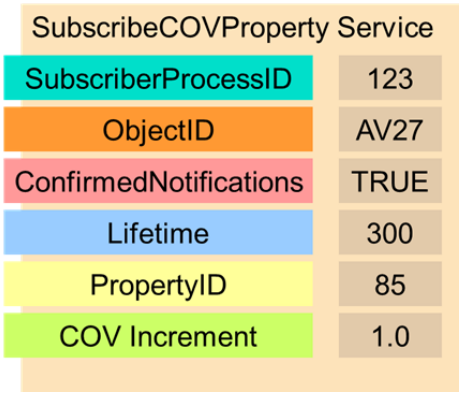
If the client is still interested in receiving notifications after 300 seconds then it must resubscribe, so typically the resubscription occurs sometime around (lifetime - resubscription). If there is no resubscription then after lifetime elapses the subscription is canceled and freed for use by others. It is possible to provide a lifetime of zero which means "indefinite lifetime". In this case the subscription is never canceled automatically. The popular opinion is that indefinite lifetimes should be discouraged. The reason for having a lifetime is that if the client "goes away" for example is turned off or disconnected for some period of time, the server will continue to try to deliver notifications to it. This causes a lot of extra work for the server, and extra bandwidth, that serves no purpose. Having a tighter lifetime assures that the server stops resending if the client subscription lapses.

The "ConfirmedNotifications" specifies whether the server sends the notification as a Confirmed or Unconfirmed service. Confirmed services require a positive handshake from the client when the notification is received, otherwise it will be retried numerous times (Nretries) after a suitable timeout period (APDU_Timeout). On the other hand Unconfirmed services are just sent once, and don't require a handshake, and are not retried. In either case the COV notification is unicast to a specific device.

This concept we've described so far is often referred to as "COV" and you'll notice that it is:

- subscription based
- targets a specific object
- implies the property that is monitored for change based on object type
- implies a change amount that comes from the target object's COV_Increment property
- provides an explicit destination process in the client

Sometimes a client may be interested in a different property than the one you would normally think of for an object type. For example, for an AV normally we are interested in Present_Value, but suppose that we want to be informed if instead the Object_Name is changed? Or if the High_Limit is changed? In these case the subscription needs to provide the property ID of the alternate property to monitor. The client can use the SubscribeCOVProperty service instead of SubscribeCOV to do this:

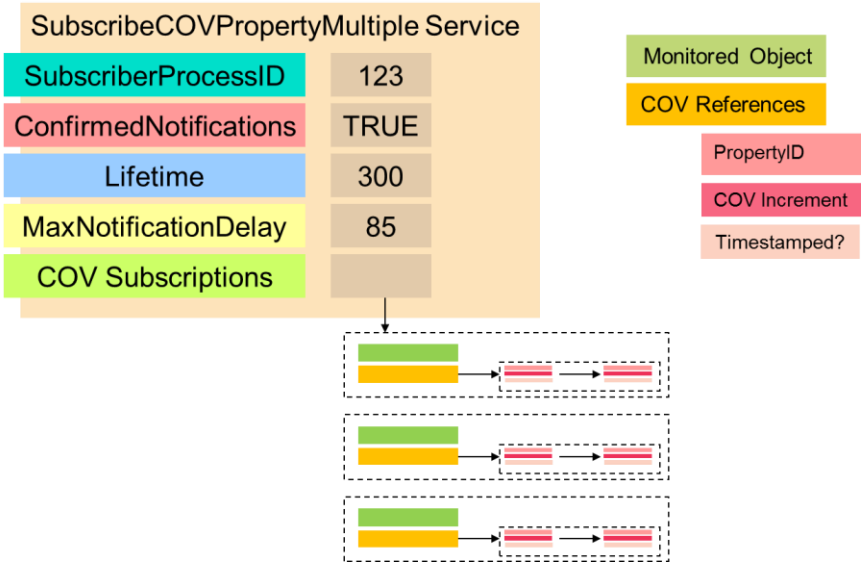



This is very similar to SubscribeCOV but includes the PropertyID. In cases when the propertyID is a numeric-valued property, such as High_Limit, the COV_Increment is provided as well. Note that if we want to override the normal COV_Increment in an object we can use SubscribeCOVProperty also for the Present_Value.

This concept is often called "COVP".

A common scenario is that a specific client device actually wants to subscribe for multiple different objects, and possibly specific properties, in the same device. While it's certainly possible to accomplish this using COV or COVP, it greatly increases the complexity and burden on the client because each individual subscription must be tracked and supervised (resubscribed). The 135-2016 BACnet standard introduces the idea of a *COV property multiple* or "COVM." The SubscribeCOVPropertyMultiple service is used to request this kind of multiple object-property subscription:




This is very similar to the other types of subscription in that a SubscriberProcessID, ConfirmedNotifications option and Lifetime are provided when subscribing. However you'll notice that the service is provided with a list of objects to be monitored, and that each of those objects also specifies its own list of *COV References*. Each reference is a PropertyID for the object property to monitor for changes as well as a COV Increment for REAL-datatyped properties only. There is also an option to request that changes include a timestamp or not.

The MaxNotificationDelay is always less than Lifetime and specifies how many seconds may elapse before notifications of changes to properties, that were specified as Timestamped=TRUE, may be delayed. The idea is to allow the device to queue notifications of timestamped changes for a while in order to reduce the number of notifications that need to be sent.

## Unsubscribed COV

But there is a second approach to COV. It's not so often used here in the US, but interestingly is almost exclusively used in Japan. This is called *Unsubscribed COV*.

The idea is that the server is configured to "know" that specific values are of general interest to peer devices, and that when those values change the server should just go ahead and send a COVnotification even though there is no explicit subscription. Usually such devices have an enable/disable for this feature as well as a maximum rate (e.g. no more than X notifications per minute). Although the server could send these unicast to specific devices, that requires configuration in the server of who the device(s) are. So usually this kind of COVNotification is broadcast. It's a really bad idea to issue such a broadcast to the global network, so when it's used it should be only broadcast on the local segment, where presumably the peer devices are located.

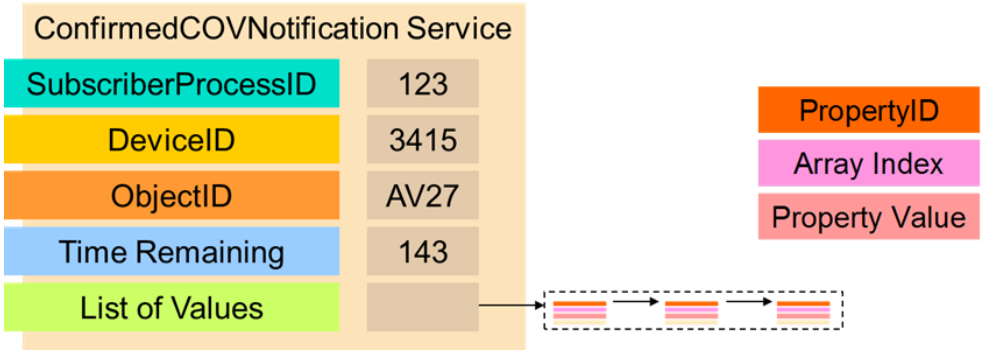
Because the unsubscribed COVnotification might be broadcast, that introduces a new problem. The subscriber ProcessID you'll recall is unique to the client that issued the subscription. However if we send a COVnotification as a broadcast then the processID MUST have the same meaning to all receiving clients.

The original idea was that processIDs were unique to a SITE but over time this has changed to unique to a client device. This breaks the original assumption.

Today we reserve the special process ID of 0 to mean "unsubscribed". When a server sends an unsubscribed COVnotification it MUST only use processID zero. Many clients treat this specially and will only look at incoming COVnotifications with processID==0 as unsubscribed. This concept is called "COVU".

## How COV Works

When COVNotifications arrive they look like this:



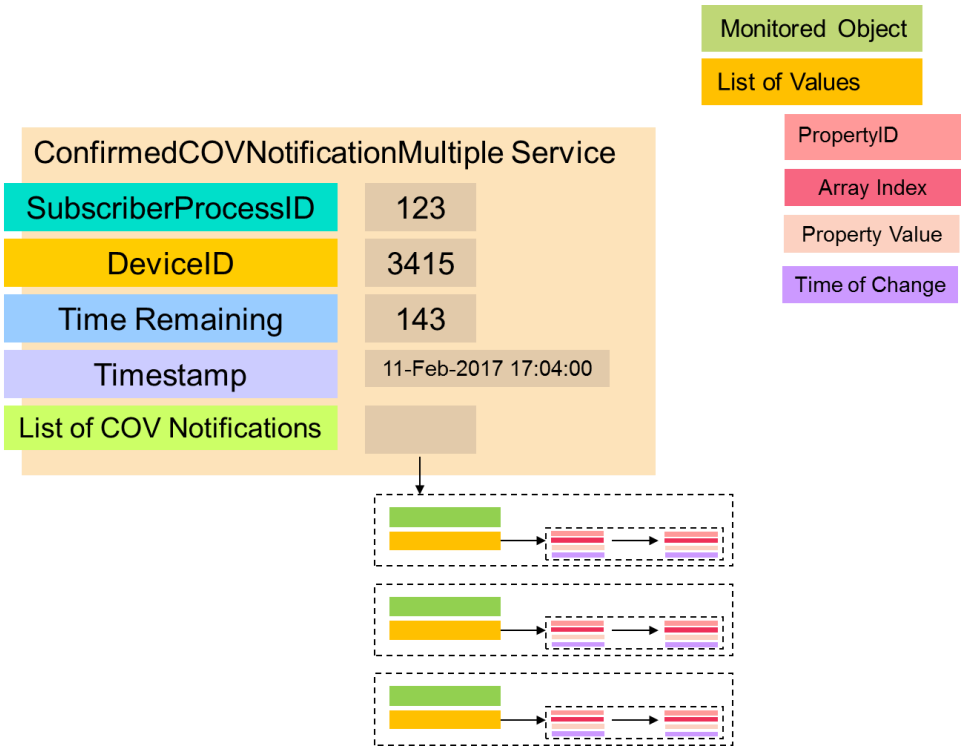

The SubscriberProcessID comes from the original subscription request, or it's zero if the notification is unsubscribed. The DeviceID is the device instance of the sending device. This is useful for unsubscribed cases because you don't necessarily know who the sending device is yet. The TimeRemaining is how long the existing subscription will last (from now). The List of Values is a list of blocks that each contain a propertyID, optionally an array index if the property is an array, and the property value that has changed.

You might ask, why do I get more than one value? The answer is that standard object types have a standard collection of values you get back, usually Present_Value and Status_Flags. But some kinds of objects are more complex. For example the Global_Group object you'll recall has a Present_Value that is a collection of multiple objects and their values.

When a Global_Group sends even an unsubscribed COVNotification, it will contain potentially many values one from each of its members. Without segmentation, or in the case of Unsubscribed broadcast which can't use segmentation, it's possible that the list of values is too big to fit in a single COVNotification. In that case, multiple COVNotifications can be sent, each one containing a subset of the group member's values.

When COVM subscriptions are used the process is similar. Instead of ConfirmedCOVNotification service, there is a ConfirmedCOVNotificationMultiple service. However since there isn't a single object as in ConfirmedCOVNotification, there are multiple objects each with potentially multiple properties and multiple values.




Notice that the service includes an optional parameter Timestamp that conveys the timestamp for the last change in the notification. It's only included if one or more of the subscribed property subscriptions specified Timestamp?=TRUE.

The List of COV Notifications includes a list of objects and for each one also a list of values for those properties that are subscribed to. Notice that a given value may also include a Time of Change timestamp if the corresponding property subscription had Timestamp?=TRUE.

There is a subtle capability that is also provided by SubscribeCOVPropertyMultiple. The server that issues ConfirmedCOVNotificationMultiple may *aggregate together* notifications of individual property changes into the same notification *as long as the subscribing client and processID are the same* and *those subscriptions were for COVM notifications*. Another subtle point is that the number of property change notifications in any given ConfirmedCOVNotificationMultiple may be less than the total number in the subscription. This is because only those properties that have changed are reported. This is very convenient for servers that don't implement segmentation, or whose subscribing client doesn't, because if there are too many changes to fit in a single ConfirmedCOVNotificationMultiple message, then multiple messages can be sent, each containing as many change notifications as will fit.

This is a considerable improvement over regular COV subscriptions, even with Global Groups, because only the changed values need be sent.

Of course, COVM can also specify UnconfirmedCOVNotificationMultiple and the same rules as COVU apply with respect to the Subscriber ProcessID when these messages are locally broadcast.