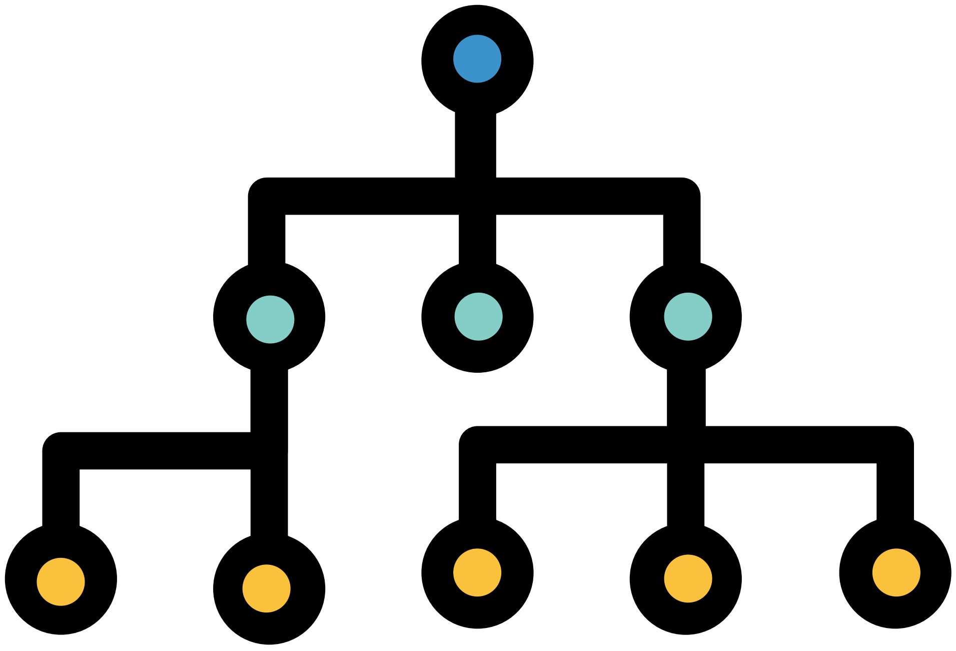
Structured View Object

David Fisher

14-Jul-2021



TUTORIAL

Contents

Structured View Object

14-Jul-2021

David Fisher

Introduction

We tend to think about BACnet objects in terms of their physicalities: analog-input-1 is a temperature sensor, binary-output-2 is a fan and so forth. But often in building automation systems (BAS) the real physical equipment is represented instead by a collection of BACnet objects that all together represent and control the equipment itself. A single smoke detector is of course a point by itself, but it is also a member of the collection of all smoke detectors, smoke detectors on floor 3, the building fire system and the air handler control for floor 3. Each of these collections is a view of some particular aspect of the BAS. When we look at a particular object's properties, it's not always clear what the object is for and how it fits into the larger scheme of the BAS.

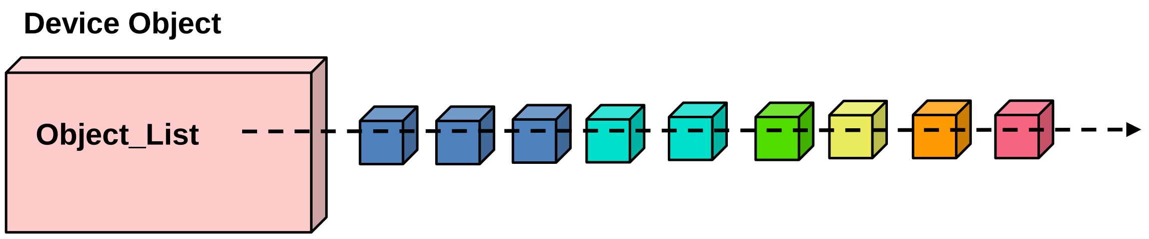
In actual use, BACnet devices and their objects are often organized into hierarchies for control and operational purposes. Many times, these hierarchies cut across device and physical boundaries to represent "logical" groupings that are conceptual, and of course groups contain other groups etc.

The Structured View (SV) object can be used for all of these purposes. In essence, an SV is a definition of such a group and its members that also includes information about the structure, purpose and relationships between BACnet-based objects. If we think of BACnet objects like files, SV objects are like folders that organize other folders and files together, as well as containing information about what kind of folder it is. SV objects may also reference other SV objects as their members allowing complete hierarchies to be represented.

Although at first glance SV objects may seem similar to Group and Global Group object types, there are significant differences. The Group objects exist as a means of organizing the values of the group members' points, so that they can be fetched using a simple reference to the Present_Value of the group object. SVs on the other hand are not directly concerned with values at all. Instead, they are focused on representing the structure of the groups and their relationships.

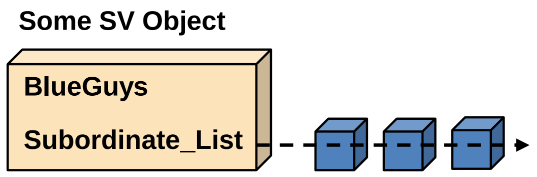
Objects in a Device

We are used to thinking about BACnet devices as having a collection of objects of various types that includes the Device object itself.

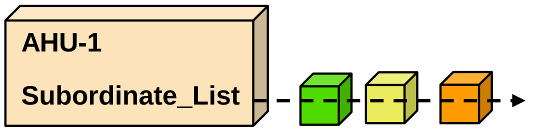


We can't really imply anything about the structure of a device's objects, or their relationships to eachother, just by looking at the Object_List. That's because the *view* which the Object_List provides is limited to containing all of the objects in the device.

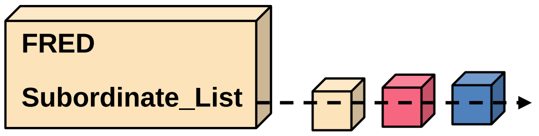
But we can use an SV object to organize, let's say, all of the "blue objects" together under one SV object:



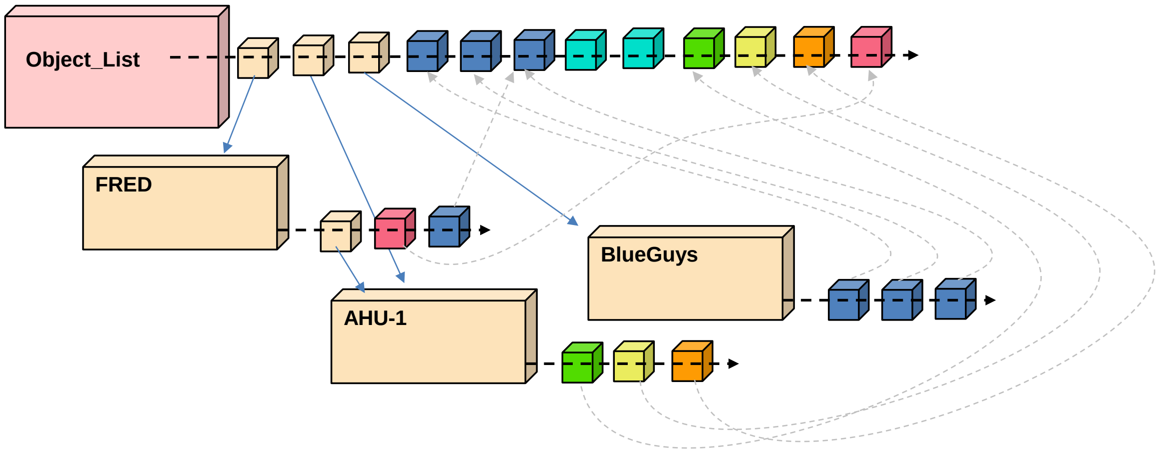
It might be that some of these objects are used to control an air handler "AHU-1":



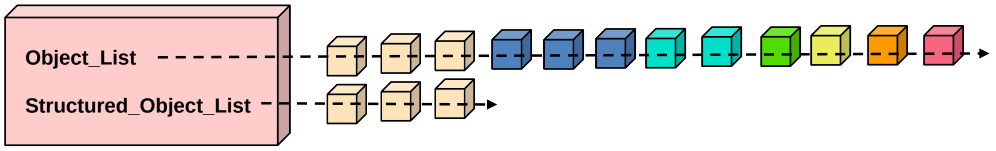
Maybe there is another group "FRED" that includes one of the BlueGuys and the red guy and AHU-1:



The point here is that the groupings can be arbitrary and whatever structure makes sense can be used. In these examples, all of the SV objects would also be added to the device's Object_List. We call the SVs "views" because each one defines a particular view (grouping) of other objects. Now our "map" of this device is quite a bit more complex:



As a BACnet client, how can we find these SV objects? It's easy enough to look through the Object_List for a device and pick out the SV objects. But first of all, not all of the SV objects have a "top level" significance. It could be, for example, that AHU-1 really only makes sense as a child of FRED. The Device object has another key property called Structured_Object_List. This <u>only</u> contains "top level" SVs:

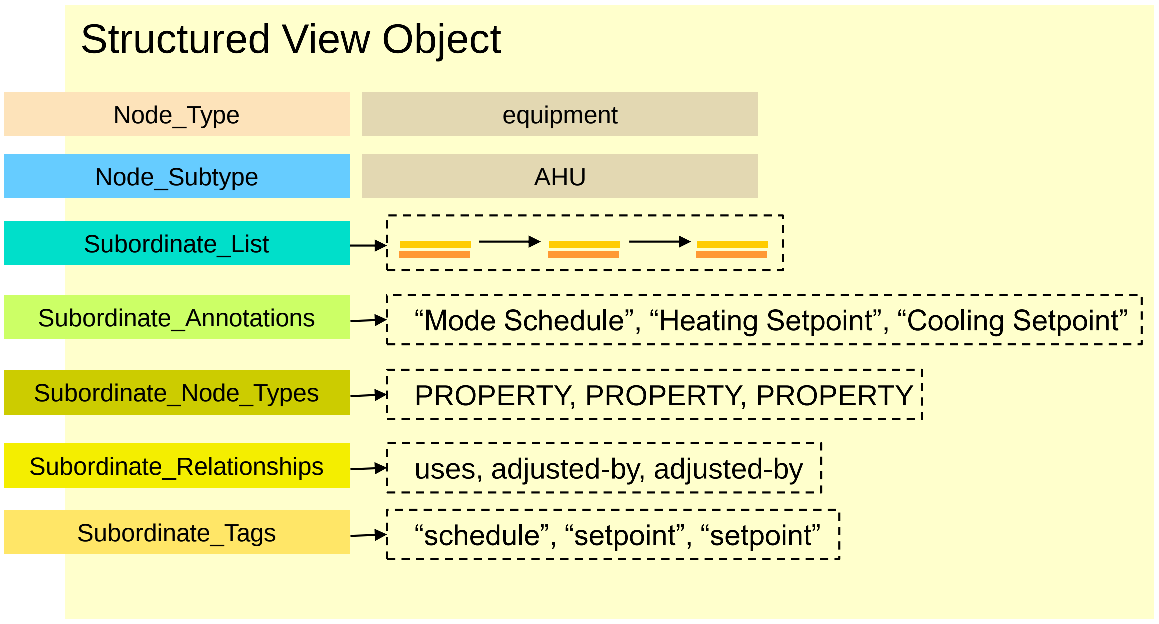


SV Object in Detail

Conceptually, SV objects are used to create hierarchical groupings of other objects. While these are useful in their own right, we can also designate some objects as "top level" and presumably directly related to the device itself. So far we've looked at how SV objects can be used to group together objects and made the distinction that this is different from Group (GR) and Global Group (GG) objects that are for grouping property values. But how does any of this help in the structuring of these groups, and how does it help to determine the relationships and purpose between the group members?

We know that all BACnet objects provide Object_Name properties that give names to individual objects. If we look at the Subordinate_List in any SV object we can find out the object identifiers of each group member and could subsequently then go and read those objects' Object_Names. This would provide a picture of the SV object name, and the names of its constituents (subordinates). The problem with that approach is that the name of a subordinate's object is not necessarily informative outside of the context of the SV it may be a subordinate of. For example, we might have an analog-input (AI) object that is measuring "leaving air temperature". But the name of the object might be something generic like "Temperature". The semantics of that object name may be invisible in the AI object itself. For those reasons, SV objects allow us to define a name for each group member. BACnet calls these *annotations*. The SV object allows each subordinate to have such an annotation using the Subordinate_Annotations property. This also allows the annotation to be different from the underlying referenced object's name and unique to the context of that structured view. There are other characteristics that can be defined for the SV and its subordinates that we'll discuss below.

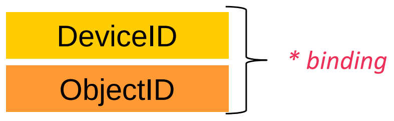
Here is a big picture of the SV object's special properties:


Structured View Object
Node_Type
equipment
Node_Subtype
AHU
Subordinate_List
Subordinate_Annotations
"Mode Schedule", "Heating Setpoint", "Cooling Setpoint"
Subordinate_Node_Types
PROPERTY, PROPERTY, PROPERTY
Subordinate_Relationships
uses, adjusted-by, adjusted-by
Subordinate_Tags
"schedule", "setpoint", "setpoint"

Each SV is a *node* within a hierarchical tree of SV objects. Every node (i.e. SV object) can be assigned a Node_Type. The collection of standard node types is: unknown, system, network, device, organizational, area, equipment, point, collection, property, functional, other, subsystem, building, floor, section, module, tree, member, protocol, room, zone. Based on the Node_Type, a client can get hints about the intended purpose of that SV.

The standard node types are intentionally general-purpose in nature. More specificity can be provided using the Node_Subtype property which is a simple character string. In the example above, the Node_Type is "equipment" and the Node_Subtype specifically says that it is an AHU (air handlinig unit).

We've talked about the Subordinate_List as a list of object identifiers, but actually the list contains *bindings* to objects in (possibly) other devices. A binding in this context means a BACnetDeviceObjectReference that identifies the device that contains that object, and which object it is.



Notice that means that one device may contain an SV object that references objects in other devices. This is convenient for supervisory controllers that have more memory resources to contain SV objects that group together objects from supervised (i.e. subordinate) devices.

It is also possible for the SV to define the Node_Type for each of the subordinate objects. In the example above, we show all of these types as being "property", but they could use any of the standard Node_Types. So, an SV that represents one building might itself have a Node_Type of "building" while its subordinates might have Node_Types of "floor".

A valuable piece of information that SV can provide is the *relationship* between the SV and each of its subordinates. In the example above you'll note that the Mode Schedule subordinate uses the value from that referenced subordinate, whereas the Heating Setpoint and Cooling Setpoint are adjusted-by the algorithm that owns the SV object. This is an example of another usage for SV objects: as an outward-facing interface to an internal control program. There are many standard relationships defined: unknown, default, contains, contained-by, uses, used-by, commands, commanded-by, adjusts, adjusted-by, ingress, egress, supplies-air, receives-air, supplies-hot-air, receives-hot-air, supplies-cool-air, receives-cool-air, supplies-power, receives-power, supplies-gas, receives-gas, supplies-water, receives-water, supplies-hot-water, receives-hot-water, supplies-cool-water, receives-cool-water, supplies-steam, receives-steam.

As a final refinement, each subordinate can also have characterstring *tags* that presumably can be searched to further characterize the subordinates.

So, What is it all For?

When a device contains a fully populated collection of SV objects, it provides a lot of information to BACnet clients that can aide in the automated formatting and presentation of hierarchical information about the objects in a device and its subordinates. The Node_Type can provide hints for a UI in terms of how to display the subordinate items, in much the way that in Windows File Explorer, the type of file (or folder) affects the icon displayed next to the name. One can easily imagine using the SV information to populate a tree graph or other visualization of the structure of objects.

Legal Stuff

Contact the Author

David Fisher

president, PolarSoft® Inc.
368 44th Street
Pittsburgh PA 15201-1761 USA
+1-412-683-2018 voice
dfisher@polarsoft.com
https://polarsoft.com