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

Source: **Edwin W. Moore on BACnet-L (14-Nov-2007)**

Question: *Is it possible to NOT support all 'n' states in the Present Value of a Multi-state Value object? For example, say you ONLY want to use the integers 3, 4, and 5 to represent the state of a specific object. Values 1 and 2 do NOT have meaning for this object (but they do for other objects you intend on supporting). Is it possible to NOT USE the values 1 and 2 (essentially 'skip' them)? So, you would support THREE distinct states, represented by the values 3, 4, and 5. If so, how do you implement this?.*

Answer: **David Fisher**

Yes this is possible. It's up to your object from moment-to-moment to determine which states are in use, and you set the value as you need to to whatever state it's in at the time. If you are always in state 3, 4 and 5 who's to say that your device is not working "properly"?

The issue comes in the handling of the State_Text array, and in handling writes to the Present_Value (if it's writable).

Let's assume you are talking about an MSV object that is intended to be writable. It's State_Text[0] reports a value of 5, which means that potentially there are 5 valid states.

Could be in your application, states 1 and 2 only can be used under certain circumstances (optional hardware installed, time of day, mode, etc.). It doesn't matter why 1 and 2 are not available at this moment, but they aren't.

Case 1 - somebody writes a value of 1 (or 2) to Present_Value

The way to handle this is to return a VALUE_OUT_OF_RANGE error. IMHO it would be more informative to say WRITE_ACCESS_DENIED but the way that 15.9.1.3.1 is written describes this as caused by the entire property not being currently writable. Purists will be upset by such a usage of the error code.

Case 2 - somebody tries to read State_Text[1] or State_Text[2] (or write them if they are writable).

There are several ways to handle this. If you don't want to support the 1 and 2 array slots at all in this scenario, then you should return a READ_ACCESS_DENIED error. This makes the most sense to clients and operators. However, regrettably this option is not mentioned in 15.5.1.3.1 (an error in my opinion). If one is a purist, then one could return INVALID_ARRAY_INDEX although the description for this case is again misleading and overly specific.

Frank Shubert observes:

… I have understood the MI,MO and MV always to support consecutive values only.

The standard reads: The logical state of the input shall be one of 'n' states, where 'n' is the number of states defined in the Number_Of_States property.

QUESTION

Sounds, as if gaps would not be allowed. If you are right and if gaps would be allowed, this would also allow implementations of e.g. error-codes, etc. in multi-state objects, interesting...

Marco Polet observes:

It's also my understanding that MV,MI or MO have to have consecutive values only. We, Priva, had also problems with mapping our own proprietary Multi states in the control software to BACnet multi-state values. But from the standard we also understood that multi-states have to start by 1 and have n consecutive state.

It's however possible to deny writing to certain values of a Multi-state. This is one of our questions in the pass to this group. We also implemented david's suggestion by returning WRITE_ACCES-DENNIED for states that can't be written, but only given as a result of a process.

Answer: David Fisher

I should have discussed this case more completely. The issue of reading the whole State_Text array is important. I see three solutions:

1. The "invalid" array slots return empty strings.
   This is easy to implement and in my view it is consistent with the intent. If you take this approach, then reading a specific slot that is "invalid" should not return an error, but also return the empty string.
2. The "invalid" array slots return a complete string
   This has the bad feature that you must provide a description for unused slots, but in Eddie's example use case, the slots aren't unused, just not valid at the moment. So this might make more sense in Eddie's use case. You could still return an error if someone writes that state, even if the slot is used. Just because my MV has five states, and each one is named, doesn't mean that it is OK RIGHT NOW to write to it.
3. The reading of the whole array (when there are gaps) returns an error.
   I suppose you could make a case for this, but to me this is not a good solution. A good client must be able to recover from an inability to read the whole array and fall back to reading it slot by slot anyway. However some <<overly rigid>> clients might only do this for SegmentationNotSupported errors so other error codes might not trigger them to read slot by slot.

The concept of discontinuous multi-state values is very common in BAS, so IMHO the standard should allow this kind of thing, as it is actually used anyway. So our interpretation of multi-state objects has always been that they represent "a maximum of N states", where some of the values in the range 1..N may be invalid. The only drawback to this approach is that one must read the whole array in order to see what states are valid, and imply from the strings returned which ones might be OK. This is no more work than if you require all states to be usable, you would still have to read the whole array. As Marco says, even if they all appear to be usable, they may not under every circumstance be writable states from moment-to-moment depending on the application.

