Context Tagging Within BACnet®

By Duffy O'Craven

At the bits and bytes level, data in BACnet packets on the wire obey some very specific rules so the receiver can glean the intent of the sender. Most of these rules are simple, and the code for any BACnet protocol stack thoroughly exercises most cases. However, the occurrence of certain conditions could execute incorrect code. This article sheds light on one such situation.

Those of you who envision data octets on the wire in BACnet are probably familiar with an arrangement such as:

Context number	Class: L/V/T Length, Value, or Type, etc.
data	

and for length of 5 or higher

Context number	1 1 0 1 Length uses extension bytes
Extended Length	
data	

Implementation of a decoder for this arrangement, to decompose the content to its constituent parts, might look like:

```
void show_head_unsigned(unsigned int offset, int tagval)
{
   int len = pif_get_byte(offset-1)&0x07;
   unsigned long value = get_bac unsigned(offset, len);
   // get_bac_unsigned() internally handles all length cases
}
```

Actually, a special case exists in the specification, in clauses 20.2.1.2 and 20.2.1.3.1, ever since its earliest days, specifying a different arrangement for context-specific tag numbers 15 or higher:

1 1 1 1	Class; L/V/T Length, Value, or Type, etc.
Context number	
data	

and for length of 5 or higher with a context-specific tag number 15 or higher:

1 1 1 1	1 1 0 1 Length uses extension bytes
Context number	
Extended Length	
data	

In this case an additional intervening byte exists between the L/V/T octet and the start of Extended Length or start of data. `offset-1` and `offset` do not have the same relationship, and the call to `get_bac_unsigned()` will fail to decode the content correctly. The code excerpt shown above didn't see it coming and isn't prepared for it.

Many implementations of BACnet encoders and decoders have probably been developed without any cognizance of this arrangement. Programmers had no reason to write the code for these special cases. No instances of context-specific tag number 15 or higher are declared anywhere in the BACnet specification. Until now.

A proposal recently considered by the Objects and Services working group of Standing Standards Project Committee 135, *BACnet®—A Data Communication Protocol for Building Automation and Control Networks* (WG-OS) advocates declaring six additional context tagged choices in the BACnetPropertyStates production in Clause 21 for data types used in properties within the standard. Three of these properties' data types were overlooked in the original declaration of the BACnetPropertyStates production, and three others are property data types added in addenda to the standard since 1995:

- action [15] BACnetAction;
- maintenance [16] BACnetMaintenance;
- notify-type [17] BACnetNotifyType;
- silenced-state [18] BACnetSilencedState;
- life-safety-operation [19] BACnetLifeSafetyOperation; and
- file-access-method [20] BACnetFileAccessMethod.

This would be the first occurrence in the standard of a production in Clause 21 declaring a context tag number 15 or higher.

Some BACnet protocol encoders and decoders can be expected to have errors in their implementations, regarding a context tag number 15 or higher.

Nonetheless, this rather innocuous and superfluous change in the standard may be the best time to bring that to light, rather than awaiting the day some other amendment to extend a complex production such as BACnetEventParameter or BACnetNotificationParameters also eventually forces the use of a context tag number 15 or higher.

All of this is very rigorous, very predictable—unless you don't see it coming and aren't prepared for it. For example, to correctly code the previous situation, simply make the modification shown in bold:

```
void   show_head
unsigned(unsigned int
offset, int tagval)
{
   int len = pif_get_
   byte(offset-1)&0x07;
```

See Context Tagging, Page B46

Context Tagging, From Page B9

```
    int tag4bits = pif_get_byte(offset-
    1)&0xF0;
    unsigned long value;
    // get_bac_unsigned() internally
    handles all length cases
    value = get_bac_unsigned(offset + (15
    == tag4bits ? 1:0), len);
}
```

There is no need to await the fate of the BACnetPropertyStates production amendment. The different arrangement for context-specific tag numbers 15 or higher is already in the standard and has been ever since its earliest days.

To implement a correct and complete BACnet decoder, the code for these special cases must be provided. Even if there are no instances of a context tag declaration 15 or higher used in standard properties, the use of a context tag number 15 or higher has always been permissible in a proprietary property value. The proper encoding for it is and always has been defined.

This particular situation, an additional intervening byte after the Length octet, is particularly pernicious, because, if this situation is ignored, not just the interpretation of the current item of data is disrupted, but the length of this item and start of the next item is miscalculated. Everything that follows in the decode of the packet will be garbled.

Shifting everything by 1 byte, or pulling the length out of the wrong byte brings on a world of trouble. This could even cause the decode for the next item to start in the middle of something in such a way that it appears valid, but is mistaken, yet no fault or flaw is reported. Tracking down that error will point in a completely wrong direction.

The implementer of a BACnet sending encoder is likely to notice this issue and make correct provision for it, since although a context tag number 15 can fit in 4 bits, any number higher than that takes 5, 6, 7 or 8 bits. Finding where to place the larger value will send the implementer scurrying to the fine print of the standard for the correct encoding, since it can't fit in 4 bits. A receiving decoder implementer, on the other hand, might never expect a context-specific tag number 15 or higher, never make provision for it, and never be able to decode any packet containing it. One must anticipate it to code for it.

In the data of Confirmed or UnconfirmedEventNotification-Request, Confirmed or UnconfirmedCOVNotification-Request, and in ReadPropertyMultiple-ACK are the most likely places for this situation to occur. Can you really live with a garbled decode of those packets on the receiving end when this situation occurs? The best time to check through and revise implementation code, if any correction is needed, is now.

Duffy O'Craven is a software consultant for Quinda in Toronto.