Sam Utley

16 August 2015

The ALERT2 Application Layer Protocol Specification, section 2.2.3 “Value Format/Length (FL)”, is currently written in such a way that suggests it is possible to encode unsigned and signed integers, floating point values, and a few timestamp formats using 1 to 15 bytes. Section 2.2.4 “Report Value (V)” goes on to limit that scope by specifying the following valid combinations, to be supported by application layer decoders:

* 1, 2, and 4 byte unsigned integers (0x11, 0x12, 0x14)
* 1, 2, and 4 byte signed integers (0x21, 0x22, 0x24)
* 4 and 8 byte IEEE floating point numbers (0x34, 0x38)

This document will proposes

1. More clearly documenting the current and newly proposed data types, either through examples or by providing reference to publicly available standards describing the data type
2. The addition of 3 byte signed integer (0x23), as used by MSR type 4
3. The addition of 3 byte unsigned integer (0x13), for symmetry
4. The addition of 8 byte unsigned integer (0x18), commonly used for noting Unix/POSIX time and IPv4 addresses
5. The addition of 8 byte signed integer (0x28), for symmetry
6. The addition of 2 byte floating point value (0x32)
7. The addition of 3 byte floating point value (0x33)
8. The addition of a CHAR value format, I propose the currently reserved type 4, allowing one to transmit an array of up to 15 characters

## 3-byte Unsigned and Signed Integer Formats (0x13 and 0x23)

The 3-byte unsigned integer value format will be indicated using a format/length designator of 0x13. The format can represent an integer value between 0 and 16777215.

|  |  |
| --- | --- |
| Value16 | Value10 |
| 00 00 00 | 0 |
| 7F FF FF | 8388607 |
| FF FF FF | 16777215 |

The 3-byte signed integer value format will be indicated using a format/length designator of 0x23. This format is used for stage in the MSR Type 4 report. This format represents an integer between -8388608 and 8388607 and is stored using two’s compliment with the most significant bit indicating sign.

|  |  |
| --- | --- |
| Value16 | Value10 |
| 80 00 00 | -8388608 |
| FF FF D6 | -42 |
| 7F FF FF | 8388607 |

## 8-byte Unsigned and Signed Integer Formats (0x18 and 0x28)

The 8-byte unsigned integer value format will be indicated using a format/length designator of 0x18. In the C programming language, it is commonly referred to as UINT64.

The 8-byte signed integer value format will be indicated using a format/length designator of 0x28. In the C programming language, it is commonly referred to as INT64.

## FP2, 2-byte Floating Point Value Format (0x32)

The FP2, 2-byte floating point value format, will be indicated using a format/length designator of 0x32.

I would like to recognize that there is an IEEE 754 notation, commonly referred to as half-precision or binary16, which specifies a 2-byte floating point number. I have generated and reviewed the list of values that can be represented using the half-precision format. This format does have a larger dynamic range than the format I will propose below (FP2). However I firmly believe that the half-precision format is sub-par, when compared to FP2, for the purposes of succinctly reporting environmental measurement data such as stage and air temperature. The half-precision format quickly loses precision as the magnitude of the value departs from zero and provides more precision than is commonly needed for values close to zero. The ALERT2 application layer document seems to suggest that precisions of 0.1, 0.01, and 0.001 are common. The half-precision format is unsuitable for 0.001 at magnitudes greater than 2.0, 0.01 at magnitudes greater than 16.0, and 0.1 at magnitudes greater than 128.0.

I propose that the ALERT2 application layer adopt a 2-byte floating point value format known as FP2. FP2 utilizes 1 bit for indicating the sign of the value, 2 bits for exponent, and 13 bits for encoding the mantissa and for flagging the special cases of ±infinity and NaN. FP2 is a previously established format, already recognized by some software packages focused on environmental data ingest, including those used for decoding GOES DCS data, and has been used by Campbell Scientific for over 30 years. The following two tables outline the bit pattern of FP2 and how they are interpreted.

|  |  |  |
| --- | --- | --- |
| Name | Bit | Description |
| Sign (S) | 15 (msb) | Specifies the sign of the value. 0 = positive, 1 = negative. |
| Exponent (E) | 14 and 13 | Specifies the magnitude of the negative decimal exponent. |
| Mantissa (M) | 12 to 0 (lsb) | Specifies the magnitude of the 13 bit mantissa, 0 to 8191 |

|  |  |  |  |
| --- | --- | --- | --- |
| S | E | M | FP2 Value Is = |
| 0 | 0 | 8191 | + infinity |
| 1 | 0 | 8191 | - infinity |
| 1 | 0 | 8190 | NaN |
| 0 or 1 | 0 to 3 | 0 to 7999 | (-1 ^ S) \* (10 ^ -E) \* M |

|  |  |
| --- | --- |
| Value16 | Value10 |
| 1F 3F | 7,999 |
| C4 D2 | -12.34 |
| 9F FE | NaN |

As shown above, FP2 has a range of -7,999 to 7,999. It has the ability to precisely represent each integer, unlike the IEEE 754 half-precision data type. The following table shows the achievable precision for a given range.

|  |  |
| --- | --- |
| Range | Max Precision |
| -7.999 to 7.999 | 0.001 |
| -79.99 to 79.99 | 0.01 |
| -799.9 to 799.9 | 0.1 |
| -7,999 to 7,999 | 1 |

## FP3, 3-byte Floating Point Value Format (0x33)

The FP3, 3-byte floating point value format, will be indicated using a format/length designator of 0x33.

I was unable to find an IEEE standard for a 3-byte floating point value.

I propose that the ALERT2 application layer adopt a 3-byte floating point value format referred to as FP3. FP3 utilizes 1 bit for indicating the sign of the value, 3 bits for exponent, and 20 bits for encoding the mantissa and for flagging the special cases of ±infinity and NaN. The following two tables outline the bit pattern of FP3 and how they are interpreted.

|  |  |  |
| --- | --- | --- |
| Name | Bit | Description |
| Sign (S) | 23 (msb) | Specifies the sign of the value. 0 = positive, 1 = negative. |
| Exponent (E) | 22 to 20 | Specifies the magnitude of the negative decimal exponent. |
| Mantissa (M) | 19 to 0 (lsb) | Specifies the magnitude of the 20 bit mantissa, 0 to 1048575 |

|  |  |  |  |
| --- | --- | --- | --- |
| S | E | M | FP2 Value Is = |
| 0 | 0 | 1048575 | + infinity |
| 1 | 0 | 1048575 | - infinity |
| 1 | 0 | 1048574 | NaN |
| 0 or 1 | 0 to 7 | 0 to 999,999 | (-1 ^ S) \* (10 ^ -E) \* M |

|  |  |
| --- | --- |
| Value16 | Value10 |
| 0F 42 3F | 999,999 |
| A0 04 D2 | -12.34 |
| 8F FF FE | NaN |

As shown above, FP3 has a range of -999,999 to 999,999. The following table shows the achievable precision for a given range.

|  |  |
| --- | --- |
| Range | Max Precision |
| -0.0999999 to 0.0999999 | 0.0000001 |
| -0.999999 to 0.999999 | 0.000001 |
| -9.99999 to 9.99999 | 0.00001 |
| -99.9999 to 99.9999 | 0.0001 |
| -999.999 to 999.999 | 0.001 |
| -9999.99 to 9999.99 | 0.01 |
| -99999.9 to 99999.9 | 0.1 |
| -999999 to 999999 | 1 |

## Character Array Value Format (4)

I propose that we add a new value format that indicates an array of characters follow. Format/length values of 0x41 to 0x4F will indicate that 1 to 15 characters follow. A common configuration of decode software would be to treat the characters as ASCII / Extended ASCII and store them as a null terminated string. Transmission of the null character would not be required, but would also not be illegal. If the character array has a length of 15 (0x4F), the null termination character would be placed at the end of the decoded data (byte 16).