MIL-STD-2045 in DFDL – Experience and Recommendations

Status of This Document

Grid Working Document (GWD)

Document Change History

2014-06-23 Created.

Copyright Notice

Copyright © Open Grid Forum (2014). Some Rights Reserved. Distribution is unlimited.

Abstract

Modeling a MIL-STD-2045 header in DFDL v1.0 is not possible without the addition of new capabilities. There are many related military-standard binary data formats which are similar, and so cannot be modeled in DFDL.

This document describes the challenges of modeling MIL-STD 2045 in DFDL, the semantics of the required new properties and property values, and several additional recommended properties and property values that help make modeling this data standard much easier.

The bit order feature has been implemented as part of the Daffodil open-source DFDL project.

Contents

[1. Introduction 4](#_Toc391384403)

[2. The bitOrder Property 5](#_Toc391384404)

[2.1 The bitOrder 'mostSignificantBitFirst' 5](#_Toc391384405)

[2.2 The bitOrder 'leastSignificantBitFirst' 5](#_Toc391384406)

[2.3 Restriction: bitOrder can only Change on Byte Boundaries 6](#_Toc391384407)

[3. Byte Order and Bit Order 7](#_Toc391384408)

[4. US-ASCII-7-Bit-Packed Character Set Encoding 8](#_Toc391384409)

[5. The byteOrder 'littleEndianAtomic16Bit' 9](#_Toc391384410)

[6. Additional Recommendations 10](#_Toc391384411)

[6.1 Presence Indicators 10](#_Toc391384412)

[6.2 Recommended Feature: dfdl:occursCountKind='prefixed' 12](#_Toc391384413)

[6.3 Repeat Indicators 12](#_Toc391384414)

[6.4 Recommended Feature: dfdl:occursCountKind="repeatUntil" 15](#_Toc391384415)

[6.5 Strings with Optional Delimiters 15](#_Toc391384416)

[6.6 Recommended Feature: dfdl:lengthKind='fixedLengthOrTerminated' 17](#_Toc391384417)

[6.7 Value Elements 17](#_Toc391384418)

[6.8 Recommended Feature: Representation Types: Solving the Value-Elements Problem 19](#_Toc391384419)

[7. Other Limitations 20](#_Toc391384420)

[7.1 Calculated Values 20](#_Toc391384421)

[7.2 Hidden Groups 20](#_Toc391384422)

[8. TDML Extension to support bitOrder 21](#_Toc391384423)

[9. Security Considerations 24](#_Toc391384424)

[10. Glossary 25](#_Toc391384425)

[11. Contributors 26](#_Toc391384426)

[12. Intellectual Property Statement 27](#_Toc391384427)

[13. Disclaimer 28](#_Toc391384428)

[14. Full Copyright Notice 29](#_Toc391384429)

[15. References 30](#_Toc391384430)

# Introduction

There are a number of military standard binary data formats. The definitions of these formats are not generally available to the public. However, MIL-STD 2045 is a publicly available standard for a binary header used in conjunction with many other binary data formats and it illustrates most of the modeling complexities of the general family of MIL-STD binary formats.

Modeling the MIL-STD-2045 header, as well as the related binary data formats, is not possible without the addition of some additional new capabilities for DFDL. These include:

* character encoding US-ASCII-7-BIT-PACKED
* property: bitOrder with values leastSignificantBitFirst and mostSignificantBitFirst
* property: byteOrder – new value littleEndianAtomic16Bit

There are several other constructs in MIL-STD-2045 which occur very frequently, and while not impossible to model, they are burdensome to model. These will be examined, and new DFDL properties or property values are suggested that would make this much more convenient, and perhaps improve performance of DFDL processors parsing/unparsing this data.

Finally, the TDML test system was enhanced in a way that makes dealing with bit ordering easy.

# The bitOrder Property

This new property controls the order of bits within a byte. That is, it defines the notion of which bit, that is, the bit with what numeric significance, is considered to be first.

There are two values for the proposed new dfdl:bitOrder property: mostSignificantBitFirst, and leastSignificantBitFirst.

## The bitOrder 'mostSignificantBitFirst'

In this bit order, the least-significant bit of byte N is adjacent to the most-significant bit of byte N+1.

For example consider these 16 bytes of data in hexadecimal: 65 FA 00 00 E5 01 00 50 A8 61 00 47 00 00 59 2F, in byte positions 1 to 16. With dfdl:bitOrder 'mostSignificantBitFirst', the data might be presented like this:

Byte 1 Byte 2 Byte 3 Byte 4 | Hex | Bit Offset

------------------------------------|----------|----

01100101 11111010 00000000 00000000 | 65FA0000 | :000000

11100101 00000001 00000000 01010000 | E5010050 | :000020

10101000 01100001 00000000 01000111 | A8610047 | :000040

00000000 00000000 01011001 00101111 | 0000592F | :000060

In the above, the first byte and last byte are highlighted in yellow. The bytes and bits are numbered from left to right with bit position 1 on the top left. The five bits at positions 6 to 10 (counting with the first bit at position 1) are underlined, and contain 10111.

## The bitOrder 'leastSignificantBitFirst'

MIL-STD-2045 stores all data least-significant-bit first meaning that within any given byte of data, the bit considered to be first is the least significant bit. So in any data which spans two or more bytes of data the most significant bit of byte N is considered positionally adjacent to the least significant bit of byte N+1.

The best way to visualize dfdl:bitOrder 'leastSignificantBitFirst' is to display the bits and bytes right to left, and number the bytes and bits starting on the right. Using the same 16 bytes as above:

Byte 4 Byte 3 Byte 2 Byte 1 | Hex | Bit Offset

------------------------------------|----------|----

00000000 00000000 11111010 01100101 | 0000FA65 | :000000

01010000 00000000 00000001 11100101 | 500001E5 | :000020

01000111 00000000 01100001 10101000 | 470061A8 | :000040

00101111 01011001 00000000 00000000 | 2F590000 | :000060

The yellow highlighted bytes above are the first and last bytes of this data. This way of writing the data keeps the place-value of bits within a binary number in left-to-right order which is the ordinary way people display numbers. The drawback of this presentation is that the code points of text strings will appear from right-to-left, that is, backwards for many languages.

It is important to understand that the bit-order of bits does not affect the numeric significance of bits within a byte, but only how the bits are assigned to positions for purposes of indexing them. Hence, if the data above were stored in a file, the first byte of the file contains the integer 0x65 or 101 decimal regardless of the setting of dfdl:bitOrder.

However, if considering the five bits at bit positions 6 to 10 (counting with the first bit at position 1), which span the boundary of the first and second data bytes, then the bits to be included depend very much on the dfdl:bitOrder. When the dfdl:bitOrder is 'leastSignificantBitFirst', then the five bits consist of bits 6, 7, and 8 from the first byte, and bits 1 and 2 from the second byte. These bits are underlined in the example above and contain 10011.

## Restriction: bitOrder can only Change on Byte Boundaries

It is a schema definition error if the bit order is changed except on a byte boundary.

An important composition principle in DFDL is that if two kinds of data can each be described in DFDL then their concatenation can be described. If the bit order differs between two concatenated data formats, then they cannot meaningfully both describe part of the same data byte; hence, it is disallowed to change bit order except on a byte boundary.

# Byte Order and Bit Order

Byte order and bit order are separate concepts. However, of the four possible combinations, only the following 3 are known to exist as data formats.

1. bigEndian with mostSignificantBitFirst
2. littleEndian with mostSignificantBitFirst
3. littleEndian with leastSignificantBitFirst (Used by MIL-STD-2045)

The final combination: bigEndian with leastSignificantBitFirst, is reserved in case of future need.

The first two combinations are discussed in the DFDL specification. This document describes the third.

This example appears in the DFDL specification (Section 13.7.1.4.1).

Positions:  
00000000 01111111 11122222  
12345678 90123456 78901234  
Bits:  
01011010 10010010 00000000

Hex values  
 5 A 9 2 0 0

As in the DFDL specification, let us examine the 13 bits beginning at position 2, in the context where dfdl:byteOrder is 'littleEndian' and dfdl:bitOrder is 'leastSignificantBitFirst' and dfdl:binaryNumberRep is 'binary'.

In this case, the bit positions are assigned differently. Below the bytes are shown left-to-right:

Positions:  
00000000 11111110 22222111  
87654321 65432109 43210987  
Bits:  
01011010 10010010 00000000

Hex values  
 5 A 9 2 0 0

The bits of interest are highlighted above. If we redisplay this same data, but reversing the order of the bytes to right-to-left, then we get:

Positions:  
22222111 11111110 00000000  
43210987 65432109 87654321  
Bits:  
00000000 10010010 01011010

Hex values  
 0 0 9 A 5 A

The above shows more clearly that we are looking at a contiguous region of bits containing 00100100101101, or the value 0x92D.

When dfdl:bitOrder is 'leastSignificantBitFirst', and dfdl:byteOrder is 'littleEndian', then the place value of the i-th bit (where the first bit is I = 1) is 2^(i-1).

# US-ASCII-7-Bit-Packed Character Set Encoding

All character strings in MIL-STD-2045 are stored in 7-bit ASCII. Unlike many uses of ASCII, these are not stored one character code per byte, but rather, all bits are used, so that in 7 bytes of data, 8 character code points can be stored. Any given code point can span a byte boundary, and start with any bit alignment; hence, the dfdl:bitOrder of the data is relevant to properly decoding a character string.

For example the bits below represent a 3-bit unsigned integer containing value 7, followed by the ASCII string 'ABC' followed by the ASCII DEL character (character code 0x7F). The bits are written right-to-left:

1111111 1000011 1000010 1000001 111

DEL C B A 7

In the above example, if we number the bits from the right, starting with position 1, the character code for 'A' occupies bits 4 to 10. If we divide the data above into bytes with vertical bars we must start on the right to get:

char: DEL C B A

bits: 01111111 | 1000011 1 |000010 10 | 00001 111

byte: 7F 87 1C 0F

byte pos: 4 3 2 1

# The byteOrder 'littleEndianAtomic16Bit'

In formats related to MIL-STD-2045 some (but not all) integers are stored in a mixed byte order. This byte order is called "little endian with atomic size 16-bit" in Wikipedia[[1]](#footnote-2). In this format the integer always has a multiple of 16-bits as its length. The data is stored littleEndian, but the units are 16-bit words, not individual bytes. Each 16-bit word is stored as a pair of bytes; however, this pair is stored big-endian.

For example, consider the integer 0x1A2B3C4D. In each byte order this is stored as the given byte sequence:

* bigEndian: 1A, 2B, 3C, 4D
* littleEndian: 4D, 3C, 2B, 1A
* littleEndianAtomic16Bit: 3C, 4D, 1A, 2B

This new enum for dfdl:byteOrder would only be usable for types xs:int, xs:long, xs:unsignedInt, xs:unsignedLong, xs:integer, xs:nonNegativeInteger, and xs:decimal. Furthermore this new byte order would only be allowed when dfdl:binaryNumberRep is 'binary'.

As with the other enum values for dfdl:byteOrder, it is necessary to determine the byte order at parse time, or via external variable bindings.

# Additional Recommendations

There are a number of areas where modeling MIL-STD-2045 data is difficult, but not impossible using DFDL. These topics include:

* Presence indicators
* Repeat indicators
* Strings
* Value elements

## Presence Indicators

Individual data elements, and groups of data elements can be preceded by a single-bit presence indicator. If the value of this bit is1 then the data element or group of data elements, is present.

These presence indicators should not appear in the final infoset after parsing. The presence or absence of the associated element item indicates presence or absence of the element in the infoset.

In the DFDL schema for MIL-STD-2045, presence indicators are modeled as single-bit unsigned integers, and used with dfdl:occursCountKind 'expression'. The integer, with value 0 or 1, gives the number of occurrences of an element.

The resulting DFDL schema construct, including hiding the presence bit in a hidden group ends up looking like:

<xs:sequence>

...

... *within a sequence where field 'unit\_name' appears with an FPI*

...

<xs:sequence

dfdl:hiddenGroupRef="

vmfdfdl:gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_reference\_message\_data\_group\_unit\_name\_32\_1\_2\_1\_FPI

"/>

<xs:element name="unit\_name" minOccurs="0" maxOccurs="1"

dfdl:occursCountKind="expression"

dfdl:occursCount="{ ../vmfdfdl:unit\_name\_FPI }">

<xs:complexType>

<xs:group ref="vmfi:tString64"/>

</xs:complexType>

</xs:element>

...

... *more fields and groups with and without presence and*

... *repeat indicators*

...

</xs:sequence>

The above uses this global group definition which contains the FPI field which is referenced from the dfdl:occursCount expression above:

<!-- hidden group definition (which must be global; hence, the huge name -->

<xs:group name="

gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_reference\_message\_data\_group\_unit\_name\_32\_1\_2\_1\_FPI

">

<xs:sequence>

<!-- this FPI element name is referenced from the occursCount of the element -->

<xs:element name="unit\_name\_FPI" type="vmfi:tFPI"/>

<!-- output value calc details omitted -->

</xs:sequence>

</xs:group>

In the above, notice the huge long names that are used to insure global uniqueness of the group definitions. These groups are necessary if the FPI field is to be hidden from the infoset.

## Recommended Feature: dfdl:occursCountKind='prefixed'

Much like dfdl:lengthKind prefixed, a dfdl:occursCountKind of 'prefixed' would allow the optional elements in MIL-STD-2045 and related binary format schemas to be greatly simplified.

## Repeat Indicators

Individual fields and groups of fields can repeat. Repeating fields or groups always appear once. (If zero repeats are possible, a field or group presence indicator must be used.)

The appearance of a second and subsequent fields is expressed by way of a field repeat indicator (FPI) or group repeat indicator (GPI). Both are single bits. When the repeat indicator is 1, the field or group will have an additional occurrance. When the repeat indicator is 0, then the occurrance is the last one.

Implementation of this behavior in DFDL is possible using discriminators and dfdl:occursCountKind 'implicit' or 'parsed'.

...

... *within a sequence where field 'control\_release\_marking' appears with an FPI and FRI*

...

<xs:sequence

dfdl:hiddenGroupRef="

vmfdfdl:gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_control\_release\_marking\_19\_1\_1\_FPI

"/>

<xs:element name="control\_release\_marking" minOccurs="0" maxOccurs="unbounded"

dfdl:lengthKind="implicit" dfdl:occursCountKind="implicit">

<xs:complexType>

<xs:sequence>

<xs:sequence>

<xs:annotation>

<xs:appinfo source="http://www.ogf.org/dfdl/">

<dfdl:discriminator>

{

if (dfdl:occursIndex() = 1)

then

../vmfdfdl:control\_release\_marking\_FPI= 1

else

../vmfdfdl:control\_release\_marking[dfdl:occursIndex()-1]/vmfdfdl:control\_release\_marking\_FRI = 1

}

</dfdl:discriminator>

</xs:appinfo>

</xs:annotation>

</xs:sequence>

<xs:sequence

dfdl:hiddenGroupRef="

vmfdfdl:gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_control\_release\_marking\_19\_1\_1\_FRI

"/>

<xs:element name="value" type="vmfdfdl:tIntField" dfdl:length="9"/>

</xs:sequence>

</xs:complexType>

</xs:element>

...

... *more fields and groups with and without presence and*

... *repeat indicators*

...

In the above, the discriminator expression is the most interesting and complex part. It tests the FRI bit of the previous instance. This is depending on the DFDL processor to forward speculate the existance of another instance of the repeating field, the discriminator then decides whether or not the additional instance actually exists.

The above code uses the two group definitions below which must exist in order to hide these bits from the final infoset:

<xs:group name="gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_control\_release\_marking\_19\_1\_1\_FRI">

<xs:sequence>

<xs:element name="control\_release\_marking\_FRI" type="vmfi:tFRI"/>

</xs:sequence>

</xs:group>

<xs:group name="gh\_mil\_std\_2045\_application\_header\_message\_handling\_group\_control\_release\_marking\_19\_1\_1\_FPI">

<xs:sequence>

<xs:element name="control\_release\_marking\_FPI" type="vmfi:tFPI"/>

</xs:sequence>

</xs:group>

## Recommended Feature: dfdl:occursCountKind="repeatUntil"

The idiom for repetition of fields and groups in MIL-STD-2045 is so prevalent in military-standard binary formats that it should be expressable directly via a new enum for the dfdl:occursCountKind property.

This enum value would cause the dfdl:occursCount property to be interpreted as a path expression producing a boolean value relative to the current array position. If the expression value is true then an additional occurrance is expected (when parsing).

The behavior of this new dfdl:occursCountKind property is much like the *'do...until'* constructs of many programming languages.

## Strings with Optional Delimiters

All text strings use an optional-delimiter strategy. The strings all have a fixed maximum length, which is a multiple of the character size (7 bits). If the string consumes all of the available length, then no delimiter is required. If the string is shorter than the maximum, then a delimiter, the DEL character (code point 0x7F) is found.

Parsing of strings using this convention is possible using dfdl:lengthKind 'pattern' and a regular expression. Removing the DEL from the string data is difficult, however, because it requires that we use a separate calculation using dfdl:inputValueCalc. A regular expression with lookahead for the delimiter does not solve the problem, because the DEL character must be consumed by the parsing.

A significant complexity is introduced because a string, which is normally thought of as a simple type, must have a complex type, because the representation involves a sequence of a hidden part that uses dfdl:lengthKind 'pattern', and a non-hidden part that uses dfdl:inputValueCalc to trim the optional DEL character from the hidden string. Hence, all strings in MIL-STD-2045 are complex types, the actual string value is obtained from the child element named 'value'.

The implementation of strings is shown below, this example is for strings of maximum length 64 (characters). A pair of groups like this must be created for each string length in the data.

..<!-- hidden group for strings of length 64 -->

<xs:group name="gh\_String64Rep">

<xs:sequence>

<xs:element name="\_\_rep" type="xs:string"

dfdl:lengthKind="pattern"

dfdl:lengthPattern="[^\x7F]{0,63}\x7F|[^\x7F]{64}"

dfdl:outputValueCalc="{

if (fn:string-length(../vmfi:value) lt 64)

then

fn:concat(../vmfi:value, dfdl:string('%DEL;'))

else

fn:substring(../vmfi:value, 1, 64)

}"

xmlns:vmfi="urn:vmfDFDLInternal"/>

</xs:sequence>

</xs:group>

<!-- this group is non-hidden and used for all strings of length 64 -->

<xs:group name="tString64">

<xs:sequence>

<xs:sequence dfdl:hiddenGroupRef="vmfi:gh\_String64Rep"/>

<!-- this value element is the string value -->

<xs:element name="value" type="xs:string"

dfdl:inputValueCalc="{

if ((fn:string-length(xs:string(../vmfi:\_\_rep)) = 64)

and fn:not(fn:ends-with(../vmfi:\_\_rep, dfdl:string('%DEL;'))))

then

xs:string(../vmfi:\_\_rep)

else

fn:substring(../vmfi:\_\_rep, 1, fn:string-length(../vmfi:\_\_rep) - 1)

}"

xmlns:vmfi="urn:vmfDFDLInternal"/>

</xs:sequence>

</xs:group>

Strings of Length 64 Implementation

Each use of a string of length 64 has schema like the example below:

<xs:element name="unit\_name" .... >

<xs:complexType>

<xs:group ref="vmfi:tString64"/>

</xs:complexType>

</xs:element>

In the case of this example, the value of this string would be obtained by the expression unit\_name/value. See also the discussion of Value Elements in another section.

## Recommended Feature: dfdl:lengthKind='fixedLengthOrTerminated'

The string idiom used in MIL-STD-2045 is prevalent enough that it should be directly supported, and the suggested name for this new length kind enum value would be 'fixedLengthOrTerminated'.

The dfdl:length property would give the maximum length of the field, but the dfdl:terminator property would specify the terminator which must be found if the string is less than the maximum length.

This would both simply the schema, and eliminate the need for a string delimited in this manner to be represented as a complex type.

An alternative to this would be some means of hiding a complex type that serves as the representation type for an element of simple type. See the appendix: Representation Types.

## Value Elements

An element that might logically be a simple type may have representation complexity that requires the corresponding DFDL schema element declaration to be of complex type. This creates a situation where elements do not directly express a value, but have a child element named 'value' (by convention) which contains the value.

Hence, the infoset of a MIL-STD-2045 header looks like this:

<mil\_std\_2045\_application\_header>

<version>

<value>3</value>

</version>

<originator\_address\_group>

<urn>

<value>207</value>

</urn>

<unit\_name>

<value>UNITA</value>

</unit\_name>

</originator\_address\_group>

<recipient\_address\_group>

<urn>

<value>3</value>

</urn>

</recipient\_address\_group>

<message\_handling\_group>

<umf>

<value>2</value>

</umf>

<vmf\_message\_identification\_group>

<fad>

<value>15</value>

</fad>

<message\_number>

<value>99</value>

</message\_number>

</vmf\_message\_identification\_group>

<operation\_indicator>

<value>1</value>

</operation\_indicator>

<retransmit\_indicator>

<value>0</value>

</retransmit\_indicator>

<message\_precedence\_codes>

<value>2</value>

</message\_precedence\_codes>

<security\_classification>

<value>0</value>

</security\_classification>

<originator\_dtg\_group>

<year>

<value>4</value>

</year>

<month>

<value>2</value>

</month>

<day>

<value>28</value>

</day>

<hour>

<value>15</value>

</hour>

<minute>

<value>27</value>

</minute>

<second>

<value>55</value>

</second>

</originator\_dtg\_group>

<acknowledgement\_request\_group>

<machine\_acknowledge\_request\_indicator>

<value>1</value>

</machine\_acknowledge\_request\_indicator>

<operator\_acknowledge\_request\_indicator>

<value>0</value>

</operator\_acknowledge\_request\_indicator>

<operator\_reply\_request\_indicator>

<value>0</value>

</operator\_reply\_request\_indicator>

</acknowledgement\_request\_group>

</message\_handling\_group>

</mil\_std\_2045\_application\_header>

This added tier of element nesting might be transformed away by a post-processing pass over the resulting DFDL infoset, but the inverse transformation which restores these value elements so as to allow unparsing is also required.

Strictly speaking, not all the above illustrated fields of the MIL-STD-2045 header require a value element. However, all fields that have either a FPI or an FRI or both require a value element, so for the sake of uniformity, even fields that are required and non-repeating have a value element.

## Recommended Feature: Representation Types: Solving the Value-Elements Problem

Any long term solution to the Value Elements problem must allow a complex type to be used as a hidden representation for an element of simple type. One such approach is illustrated by example here.

Consider this DFDL fragment with typothetical new property dfdl:representationType:

<xs:element name="myString" type="xs:string"

dfdl:representationType="tns:tString64"/>

<xs:complexType name="tString64Rep">

<xs:sequence>

<xs:element name="\_\_rep" type="xs:string"

dfdl:lengthKind="pattern"

dfdl:lengthPattern="[^\x7F]{0,63}\x7F|[^\x7F]{64}"

dfdl:outputValueCalc="{

if (fn:string-length(../vmfi:value) lt 64)

then

fn:concat(../vmfi:value, dfdl:string('%DEL;'))

else

fn:substring(../vmfi:value, 1, 64)

}"

xmlns:vmfi="urn:vmfDFDLInternal"/>

<!-- this value element is the string value -->

<xs:element name="value" type="xs:string"

dfdl:inputValueCalc="{

if ((fn:string-length(xs:string(../vmfi:\_\_rep)) = 64)

and fn:not(fn:ends-with(../vmfi:\_\_rep, dfdl:string('%DEL;'))))

then

xs:string(../vmfi:\_\_rep)

else

fn:substring(../vmfi:\_\_rep, 1, fn:string-length(../vmfi:\_\_rep) - 1)

}"

xmlns:vmfi="urn:vmfDFDLInternal"/>

</xs:sequence>

</xs:complexType>

The above suggests that an element of simple type can have a complex representation that is hidden. The child of the complex type with distinguished name 'value' provides the value of the element.

A few observations:

1. The infoset can still be validated because the representation type's child elements do not appear in the infoset.
2. The DFDL schema for the representation type itself (tString64) is visible to the DFDL implementation as any complex type definition.

This mechanism has some similarities to the dfdl:prefixLengthType property, which hides an implicit type definition from the infoset.

# Other Limitations

In the course of implementing MIL-STD-2045, the following additional limiations were found to be troublesome:

## Calculated Values

* dfdl:inputValueCalc only allowed on scalar elements (minOccurs 1, maxOccurs 1)
* dfdl:inputValueCalc only allows on local element decls (not on types, not on global elements)

The above imply there is no way to create a logical simple type from a complex type representation if the element is optional or array. This is an additional source of requirement for additional element tiers which are needed to satisfy the DFDL schema requirements but are not logically necessary in the resulting infoset. These restrictions seem arbitrary given how easy it is to work around them by inserting another tier of elements.

## Hidden Groups

* No ability to nest hidden groups adjacent to their point of use such that their names are localized.

A syntax where a hidden group could be directly embedded in the schema would eliminate the need for most long global names.

This requires the use of globally unique names for each hidden group. These verbose names make debugging the schema much more difficult.

# TDML Extension to support bitOrder

The TDML test data markup language is extended to support testing of data that uses the dfdl:bitOrder property. Two new capabilities were added:

document parts with bytes written in right-to-left increasing order

document parts with least-significant-bit-first bit order.

The combination of these two capabilities allow direct translation of examples from the MIL-STD-2045 specification into TDML test cases.

For example, the TDML fragment below illustrates the first part of the example in Table B-I of the MIL-STD-2045 specification

|  |
| --- |
| <document bitOrder="LSBFirst">  <documentPart type="bits" byteOrder="RTL">Version XXXX 0011</documentPart>  <documentPart type="bits" byteOrder="RTL">FPI XXX0 XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">GPI for Originator Address XX1X XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">FPI for URN X1XX XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">URN X0000000 00000000 01100111 1XXX XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">FPI for Unit Name 1XXX XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">Unit Name (UNITA) U X101 0101</documentPart>  <documentPart type="bits" byteOrder="RTL"> 0XXX XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL"> N XX10 0111</documentPart>  <documentPart type="bits" byteOrder="RTL"> 01XX XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL"> I XXX1 0010</documentPart>  <documentPart type="bits" byteOrder="RTL"> 100X XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL"> T XXXX 1010</documentPart>  <documentPart type="bits" byteOrder="RTL"> 0001 XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL"> A XXXX X100</documentPart>  <documentPart type="bits" byteOrder="RTL"> 1111 1XXX</documentPart>  <documentPart type="bits" byteOrder="RTL"> DEL XXXX XX11</documentPart>  <documentPart type="bits" byteOrder="RTL">GPI for Recip. Addr Group XXXX X1XX</documentPart>  <documentPart type="bits" byteOrder="RTL">GRI for R\_ONE XXXX 0XXX</documentPart>  <documentPart type="bits" byteOrder="RTL">FPI for URN XXX1 XXXX</documentPart>  <documentPart type="bits" byteOrder="RTL">URN XXXX00000 00000000 00000000 011X XXXX</documentPart>  </document> |

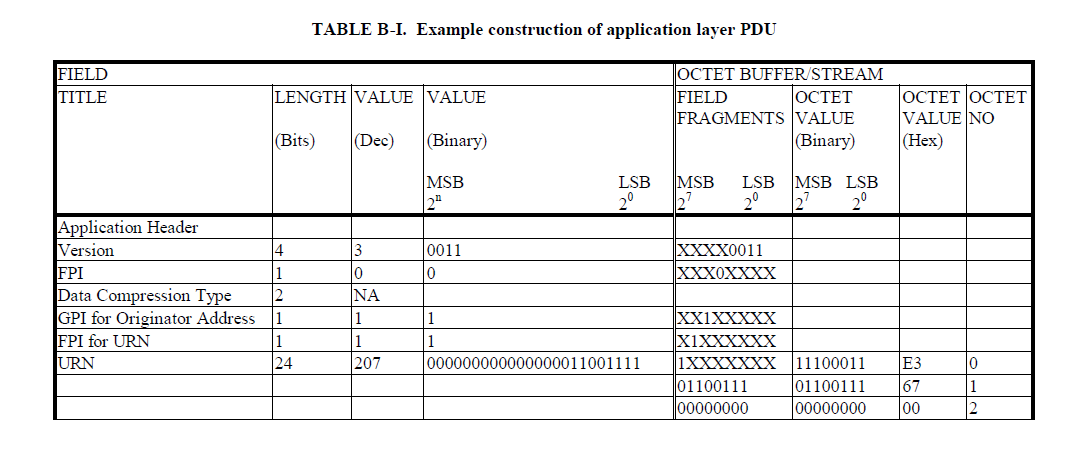
Table : TDML for part of MIL-STD-2045 header. Specifies the bytes E3 67 00 80 55 67 92 1A FC 77 00 00 00.

In the above, the tdml:document element has an attribute named 'bitOrder' which has values 'MSBFirst' (the default), or 'LSBFirst'. This attribute appearing on the tdml:document element is just a convenience. It is equivalent to specifying the same attribute on each tdml:documentPart element that does not have its own 'bitOrder' attribute.

The tdml:documentPart elements also each carry an attribute named 'byteOrder' with values 'LTR' (the default) meaning “left to right”, and 'RTL' meaning “right to left”. With value 'RTL' the data is interpreted as bytes numbered starting from the right.

This data example uses type='bits', and so any character other than 0 or 1 is ignored. Hence, we are able to conveniently insert labels (remembering to avoid characters 1 and 0 in the text) and we use X to represent a bit placeholder that will be supplied in a subsequent tdml:documentPart.

This same convention is used in the MIL-STD-2045 specification to illustrate example data:



The combination of properties of bitOrder 'LSBFirst' and byteOrder 'LTR' is disallowed.

The byteOrder attribute can also be used on tdml:documentPart elements of type 'byte'.

# Security Considerations

Only data parsing is discussed in this document. Data serialization, or 'unparsing', raises one critical data security issue which is that when writing data, the contents of all of the data must be specified so as to prevent unused parts of the data being used for unintended purposes.

The DFDL standard (see [DFDL]) specifically provides for this by requiring that a fill byte be specified, the contents of which are used to fill in any unused bits or bytes of the output data.

# Glossary

Byte – 8 bits of data, also called an Octet.

DFDL - Data Format Description Language

MIL-STD-2045 – MIL Standard 2045 47001D with Change 1. See [MILSTD2045] in references.

TDML - Test Data Markup Language

# Contributors

Michael J. Beckerle (Corresponding Author)

Tresys Technology

Columbia, MD

USA

Email: mbeckerle@tresys.com

# Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

# Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

# Full Copyright Notice

Copyright (C) Open Grid Forum (2014). Some Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included as references to the derived portions on all such copies and derivative works. The published OGF document from which such works are derived, however, may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing new or updated OGF documents in conformance with the procedures defined in the OGF Document Process, or as required to translate it into languages other than English. OGF, with the approval of its board, may remove this restriction for inclusion of OGF document content for the purpose of producing standards in cooperation with other international standards bodies.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

# References

[MILSTD2045] CONNECTIONLESS DATA TRANSFER APPLICATION LAYER STANDARD, MIL-STD-2045-47001D w/CHANGE 1, 23 June 2008 (available publicly from US Dept. of Defense at <http://assistdocs.com/>)

[Daffodil] <https://opensource.ncsa.illinois.edu/confluence/display/DFDL/Daffodil%3A+Open+Source+DFDL>

[DFDL] Michael J Beckerle, Steven M Hanson, Alan W Powell. Data Format Description Language (DFDL) v1.0 Specification. Open Grid Forum. (<http://redmine.ogf.org/dmsf/dfdl-wg>)

Forthcoming Update: GFD-P-R.207 (2014)

Obsolete: GFD-P-R.174. January 2011.

[TDML] Michael J. Beckerle – Test Data Markup Language. (<http://redmine.ogf.org/dmsf_files/13238>)

1. http://en.wikipedia.org/wiki/Endianness [↑](#footnote-ref-2)