Additional Suggested DFDL Features to Simplify Modeling of MIL-STD-2045

Status of This Document

Grid Working Document (GWD)

Document Change History

2014-08-18 Revised - removed suggested features for which alternatives have been proposed, changed repeat-until scheme to stop-indicator scheme.

2014-07-23 Created - Material split out from prior document for separate review and consideration.

Copyright Notice

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

Abstract

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

Contents

[Additional Suggested DFDL Features to Simplify Modeling of MIL-STD-2045 1](#_Toc396139720)

[1. Introduction 3](#_Toc396139721)

[2. Recommendations 4](#_Toc396139722)

[2.1 Repeat Indicators 4](#_Toc396139769)

[2.2 Recommended Feature: dfdl:occursCountKind="stopIndicator" and dfdl:occursStopIndicatorType. 7](#_Toc396139770)

[2.3 Value Elements 7](#_Toc396139829)

[2.4 Recommended Feature: Representation Types: Solving the Value-Elements Problem 9](#_Toc396139830)

[3. Other Limitations 10](#_Toc396139831)

[3.1 Calculated Values 10](#_Toc396139832)

[3.2 Hidden Groups 10](#_Toc396139833)

[4. Security Considerations 11](#_Toc396139834)

[5. Glossary 12](#_Toc396139835)

[6. Contributors 13](#_Toc396139836)

[7. Intellectual Property Statement 14](#_Toc396139837)

[8. Disclaimer 15](#_Toc396139838)

[9. Full Copyright Notice 16](#_Toc396139839)

[10. References 17](#_Toc396139840)

# 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 is difficult. There are several constructs in MIL-STD-2045 which occur very frequently, and while not impossible to model, they are burdensome to model. 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.

# Recommendations

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

* Repeat indicators
* Value elements

## 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 before the repeating field/group.)

The appearance of a second field and subsequent fields is expressed by way of a field repeat indicator (FRI) or group repeat indicator (GRI). 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\_control\_release\_marking\_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\_control\_release\_marking\_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\_control\_release\_marking\_FRI">

<xs:sequence>

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

</xs:sequence>

</xs:group>

<xs:group name="gh\_control\_release\_marking\_FPI">

<xs:sequence>

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

</xs:sequence>

</xs:group>

## Recommended Feature: dfdl:occursCountKind="stopIndicator" and dfdl:occursStopIndicatorType.

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.

The stop indicator type is a QName of a type which must be derived from xs:boolean. True means the occurrence is the last. False means it is not. If the opposite sense is needed then the DFDL Boolean properties of the type can be used to compensate.

The parser would work a bit like it does for 'stopValue' - it keeps parsing speculatively until it finds an occurrence which indicates the end of the array - the difference being that in this case it is added to the infoset.

The oddity about this is that it applies to arrays only (meaning more than 1 possible occurrence). It does not work with optional elements, as there is no way to represent zero occurrences.

Hence it can not be used with minOccurs of '0'. It should be a schema-definition-error if such usage occurs.  
  
The data grammar of section 9 becomes:   
  
SimpleNormalRep = LeftFraming StopIndicator PrefixLength SimpleContent RightFraming   
ComplexNormalRep = LeftFraming StopIndicator PrefixLength ComplexContent ***ElementUnused*** RightFraming   
  
StopIndicator = SimpleContent

## 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, some fields require quite complex expressions using dfdl:inputValueCalc, so for the sake of uniformity, all fields are implemented (currently) as having 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.

# 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](#a_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

|  |  |
| --- | --- |
| [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. |
| [DFDLCharset] | Michael J Beckerle, DFDL-Specific Character Set Encodings, Open Grid Forum 2014  Forthcoming |
| [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/>) |
| [TDML] | Michael J. Beckerle – Test Data Markup Language. (<http://redmine.ogf.org/dmsf_files/13238>) |