GWD-I
Category: INFORMATIONAL
GGF Data Format Description Language Working Group

Martin Westhead, EPCC, University of Edinburgh
et. al….

10th September 2003

Data Format Description Language – XML representation

Status of This Memo

This memo provides information to the Grid community regarding the specification of a Data Format Description Language. The specification is currently an early draft which does not represent a consensus within the group. Distribution is unlimited.

Abstract

XML provides an essential mechanism for transferring data between services in an application and platform neutral format. However it is not well suited to large datasets with repetitive structures, such as large arrays or tables. Furthermore, many legacy systems and valuable data sets exist that do not use the XML format. The aim of this working group is to define an XML-based language, the Data Format Description Language (DFDL), for describing the structure of binary and character encoded (ASCII/Unicode) files and data streams so that their format, structure, and metadata can be exposed. This effort specifically does not aim to create a generic data representation language. Rather, DFDL endeavors to describe existing formats in an actionable manner that makes the data in its current format accessible through generic mechanisms.

This document defines the XML representation and associated API for the DFDL structural description language. The semantics for this structural representation are given in the associated document “Data Format Description Language – structural description”.

Contents

1. Introduction

DFDL aims to provide the following:

- a way to describe the structure of a binary sequence (e.g. file or bit stream)
- a way to attach semantic labels to features of that structure
- one or more ontologies of such semantic labels.

This document will define the realisation of the DFDL structural description language (described in the document "DFDL structural description") in XML and APIs.

Pieces to this document:

1. XML Schema conventions
2. Core representation of structural description language – representing the various operators in XML
3. Structural description document – arrangement of a description document itself with examples
4. XML view of described data
5. XML Schema – schema for the description document
6. Access API

Note: this core language does not contain any primitive types other than "bit". However, some of the types defined in the Primitive Types Ontology are used to help motivate some of the examples.

Need to make sure its clear which bits are SDL and which are imported from the basic ontologies.

2. XML Schema representation conventions

- Type name as element name
- Element and attribute names follow W3C capitalization convention
  - Initial letter is lower case
  - Upper case is used to separate words e.g. fooBar
- All metadata representations in attributes

3. Core representation

There follows a discussion of the various elements in the language and their correspondence to the operators in the SDL formal spec.

3.1 Representation of Types

The schema defines an element:

<type name="foo" varName="bar/>

Since many things in dfdl are equivalent to a type, for example all of the operators in the SDL return a type. Thus, the XML Schema is based heavily on the use of substation groups. All the different operations are included in a substitution group with the <type/> tag. Plus, additional types added in ontologies can be added to the substitution group to get transparent extensibility.

This means that here are currently two different styles representing types (structured binary sequence sets) in DFDL.

1. the type name as the element name

<foo byteOrder="bigEndian" varName="temperature">

2. we have a generic element name with a compulsory attribute *name*:

<type name="foo" varName="temperature">

This second form has two variants:

<newType name="foo"/>
This is used to define a new type and

<paramType name="typeName"/>

when "typeName" is a formal parameter that must be substituted for a real type using <setParameter>.

The reason for having so many different forms is to make it easier to introduce identity constraints within the XML schema for matching the names. There is probably more work to do in this area and we may decide to simplify the picture with a little use experience.

Within this schema we introduce one concrete type <bit> which is the basis for constructing all the others.

3.2 Assignment

This takes two forms for the different styles. Firstly assuming our new type *foo* has been defined as an element in an included schema definition we can use:

```
foo:=bah

<define>
    <foo/>
    <toBe>
      <bar/>
    </toBe>
</define>
```

Alternatively if *foo* is not defined as an element we can provide a definition inline:

```
foo:=bah

<define>
    <newType name="foo"/>
    <toBe>
      <bar/>
    </toBe>
</define>
```

3.3 Sequence

```
[a; b; c]

<sequence>
```

```
    <a/>
    <b/>
    <c/>
</sequence>
```

Concatenation

```
A::B

<concatenate>
  <A/>
  <B/>
</concatenate>
```

3.4 Repetition

- Fixed
- *
- +
- ?

```
char.5

<repeat number=”5”>
  <char/>
</repeat>
```

```
char.*

<repeat number=”unbounded”>
  <char/>
</repeat>
```

```
char.+

<repeat number=”oneOrMore”>
  <char/>
</repeat>
```

```
char.?

<repeat number=”zeroOrOne”>
  <char/>
</repeat>
```

3.5 Parameters

```
array( type, size ) := type.size

<define>
   <array>
      <parameter name=”type”/>
      <parameter name=”size”/>
   </array>
    <is/>
      <repeat number=”$size”>
         <dfdlType name=”$type”/>
      </repeat>
</define>

array(float, 10)
```

```
<array>
    <setParameter name=”type” value=”float”/>
    <setParameter name=”size” value=”10”/>
</array>
```

NB for this to work we are assuming that <type name=”foo”/> is exactly equivalent to <foo/>.

3.6 Labels

Labels are just attributes:

```
myType<varName=”pressure”, units=”Pa”>

<myType varName=”pressure” units=”Pa”>

<dfdlType name=”myType” varName=”pressure” units=”Pa”>
```

3.7 Data References

A data reference is an XPath reference over the XML view of the data (see section on XML views). We use the # character to warn the implementation that there is a data reference coming. For example:

```
variableArray(type) := [int<varName=”size”;
                         array(type, β(“../size[@varName=’size’]”))]

<define>
    <variableArray”>
        <parameter name=”type”/>
    <variableArray”/>
    <toBe/>
      <sequence>
        <integer varName=”size”/>
        <array>
            <parameter name=”type” value=”type”/>
            <parameter name=”size” value=”../size[@varName=’size’]” />
        </array>
      </sequence>
    </toBe>
</define>
```

3.8 Conditionals

By reference

```
myUnion := [int; (“..\int”, α(0):float, α(1):complex)]

<define>
    <myUnion/>
    <toBe>
      <sequence>
         <int/>
         <switch value=”#../int”>
             <case value=”0”>
                 <float/>
             </case>
             <case value=”1”>
                 <complex/>
             </case>
         </switch>
      <sequence/>
```

```
    </toBe>
</define>
```

By Pattern

```
[( ‘a’ | ‘b’ ); ‘x’]

<sequence>
    <either>
      <char>a</char>
      <char>b</char>
    </either>
</sequence>
```

3.9 Character sets

In the SDL document we used set builder notation to define sets and ranges of values. We need to be able to express the same properties in XML. Fro this we need:

- inclusion – build a set from specific elements
- exclusion – exclude specific elements from a set
- ranges – define a range of elements to be included or excluded

Simple inclusion:

```
<define>
  <comma/>
  <toBe>
    <char>,</char>
  </toBe>
</define>
```

More complex inclusion

```
<define>
  <digit/>
  <toBe>
    <either>
      <char>0</char>
      <char>1</char>
      <char>2</char>
      <char>3</char>
      <char>4</char>
      <char>5</char>
      <char>6</char>
      <char>7</char>
      <char>8</char>
      <char>9</char>
    </either>
  </toBe>
</define>
```

Ranges:

```
<define>
  <lowerCase/>
  <toBe>
    <range>
      <char>a</char>
      <char>z</char>
    </range>
  </toBe>
</define>
```

Exclusion

```
<define>
  <lowerCaseConsonants/>
  <toBe>
    <exclude>
      <either>
        <char>a</char>
        <char>e</char>
        <char>i</char>
        <char>o</char>
        <char>u</char>
      </either>
      <from>
        <lowerCase/>
      </from>
    </exclude>
  </toBe>
</define>
```

3.10 Pointers
3.11 Transformations

4. Structural description document

Proposed document form

```
<dfdl>

    <definitions>
    <!-- definitions of new composed types -->
    </definitions>

    <description>
    <!-- description of the digital entity in question -->
    </description>

</dfdl>
```

So the document has a root element <dfdl> which has two children <definitions> in which any supplemental composite definitions are placed and <description> which contains the description of the digital entity that we are actually describing.

Question: do we really need to separate these two?

5. XML view of described data

The basic principle here will follow BFD and DataBinX in placing the data inside the corresponding dfdl elements:

```
<float>3.142</float>
```

However some thought needs to go in to the representation of more complex structures. In particular we need to have sequence indexes arranged so that we can reference the data with respect to them.

6. Access API

The proposal is that any DFDL structure would be represented by an object. All objects would inherit methods from a common root object with a set of common methods. The behaviour of the methods would depend on the actual class of the object.

The common methods would be split into the following categories:

1. Reflection
2. Navigation
3. Primitive access
4. Array access

6.1 Reflection

The reflection methods would allow discovery of the object's dfdl type, plus possibly other information about it.

```
String      type();
```

6.2 Navigation

This would include methods for navigating through the tree representation of the data it would include methods for navigating the tree and stepping along an individual sequence for example

```
dfdlObject  childAt(int index);
dfdlObject  parent();
dfdlObject  nextSibling();
dfdlObject  previousSibling();
int         numberOfChildren();
```

6.3 Primitive access

This set of methods is for accessing the data as primitives. Any new type will need to have a definition within the ontology of the behaviour of each of these methods when called on an object corresponding to that type. It may, of course, return an error.

```
byte        getAsByte();
short       getAsShort();
int         getAsInt();
long        getAsLong();
char        getAsChar();
float       getAsFloat();
double      getAsDouble();
boolean     getAsBoolean();
String      getAsString();
```

There would also be corresponding methods for setting the value of the object

```
void    set (byte value);
void    set (short value);
void    set (int value);
void    set (long value);
void    set (char value);
void    set (float value);
void    set (double value);
void    set (boolean value);
void    set (String value);
```

Note the **getAsString**(); and the **set**(String value); are particularly relevant

6.4 Array access

This set of methods corresponds to the previous set except that they are accessing to and from arrays of primitive values.

```
byte[]      getAsByteArray();
short[]     getAsShortArray ();
int[]       getAsInArrayt();
long[]      getAsLongArray ();
char[]      getAsCharArray ();
float[]     getAsFloatArray ();
double[]    getAsDoubleArray ();
boolean[]   getAsBooleanArray ();
String[]    getAsStringArray ();
```

There would also be corresponding methods for setting the value of the object

```
void    set (byte[] value);
void    set (short[] value);
void    set (int[] value);
void    set (long[] value);
void    set (char[] value);
void    set (float[] value);
void    set (double[] value);
void    set (boolean[] value);
void    set (String value);
```

Note the **getAsString**(); and the **set**(String value); are particularly important since these represent the alpha and beta functions in the formal SDL.

7. Security Considerations

There are no security considerations that we are aware of at this time.

Appendix 1 – XML Schema

Discussion of structure of schema…

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Martin Westhead
(EPCC) -->
<!--W3C Schema generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<xs:schema targetNamespace="http://www.dfdl.org/2003/dfdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.dfdl.org/2003/dfdl" elementFormDefault="qualified">
   <xs:element name="type">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="typeOfType">
               <xs:attribute name="name" type="xs:string" use="required"/>
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:element name="repeat" substitutionGroup="type">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="type"/>
         </xs:sequence>
         <xs:attribute name="number" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:string"/>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <xs:element name="sequence" substitutionGroup="type">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="type" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="switch" substitutionGroup="type">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="case" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="type"/>
                  </xs:sequence>
                  <xs:attribute name="value" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="value" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="concatenate" substitutionGroup="type">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="type" minOccurs="2" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="either" substitutionGroup="type">
      <xs:complexType>
         <xs:sequence>
```

```
         <xs:element ref="type" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="define">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="type"/>
         <xs:element name="toBe">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="type"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:key name="defineParamKey">
      <xs:selector xpath=".//parameter"/>
      <xs:field xpath="@name"/>
   </xs:key>
   <xs:keyref name="defineParamTypeRef" refer="defineParamKey">
      <xs:selector xpath=".//paramType"/>
      <xs:field xpath="@name"/>
   </xs:keyref>
</xs:element>
<xs:element name="dfdl">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="definitions" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="define" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="description" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="type"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:key name="typeNameKey">
      <xs:selector xpath=".//newType"/>
      <xs:field xpath="@name"/>
   </xs:key>
   <xs:keyref name="typeNameRef" refer="typeNameKey">
      <xs:selector xpath=".//type"/>
      <xs:field xpath="@name"/>
   </xs:keyref>
</xs:element>
<xs:element name="range" substitutionGroup="type">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="type" minOccurs="2" maxOccurs="2"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="exclude" substitutionGroup="type">
   <xs:complexType>
```

```
         <xs:sequence>
            <xs:element ref="type"/>
            <xs:element name="from">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="type"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="newType" substitutionGroup="type">
      <xs:annotation>
         <xs:documentation>Used to define a new type on the
fly</xs:documentation>
      </xs:annotation>
      <xs:complexType mixed="true">
         <xs:sequence>
            <xs:element name="parameter" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string" use="required"/>
         <xs:attribute name="varName" type="xs:string"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="paramType" substitutionGroup="type">
      <xs:annotation>
         <xs:documentation>Used to allow parameterisation of types (but allow
matching constraints)</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="parameter" minOccurs="0" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required"/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="name" type="xs:string"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="bit" type="typeOfType" substitutionGroup="type"/>
   <xs:complexType name="typeOfType" mixed="true">
      <xs:choice>
         <xs:element name="setParameter" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:attribute name="name" type="xs:string" use="required"/>
               <xs:attribute name="value" type="xs:string" use="required"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="parameter" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:attribute name="name" type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:choice>
      <xs:attribute name="varName" type="xs:string"/>
   </xs:complexType>
</xs:schema>
```

Author Information

Martin Westhead, M.Westhead@epcc.ed.ac.uk, EPCC, University of Edinburgh. James Clerk Maxwell Building, Mayfield Road, Edinburgh EH9 3JZ, UK.

Glossary

DFDL – Data Format Description Language

Intellectual Property Statement

The GGF 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 GGF Secretariat.

The GGF 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 GGF Executive Director.

Full Copyright Notice

References

BinX http://www.epcc.ed.ac.uk/gridserve/WP5/Binx/
HDF http://hdf.ncsa.uiuc.edu/HDF5
BDF/SAM http://collaboratory.emsl.pnl.gov/docs/collab/sam
XDR http://www.faqs.org/rfcs/rfc1014.html
DFDL web pages http://forge.gridforum.org/projects/dfdl-wg/