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 – Basic Structures Ontology

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.

Copyright Notice

Copyright © Global Grid Forum (2002). All Rights Reserved.

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 ontology of basic structures.

Contents

1. Purpose of this Ontology

This document provides the second of two basic ontologies. This is the ontology of basic structures that builds on the ontology of primitive types.

2. XML Schema additions

The following elements were added to the schema:

- textInteger – text representation of integers
- textFloat – text representation of floating point numbers
- nullTermString – null terminated string format
- complex-32 – complex number composed of two 32 bit floats
- complex-64 – complex number composed of two 64 bit floats
- array – a one dimensional array parameterized by size and type
- array-2d – a two dimensional array parameterized by size and type
- array-3d – a three dimensional array parameterized by size and type
- array-4d – a four dimensional array parameterized by size and type
- separatedValueTable – a table of values with parameterized value and row separators

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Mario Antonioletti
(EPCC) -->
<xs:schema targetNamespace="http://www.dfdl.org/2003/structures"
xmlns:structures="http://www.dfdl.org/2003/structures"
xmlns:dfdl="http://www.dfdl.org/2003/dfdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:prim="http://www.dfdl.org/2003/primitives" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:import namespace="http://www.dfdl.org/2003/primitives"
schemaLocation="primitives.xsd"/>
   <xs:element name="textInteger" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="textFloat" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="nullTermString" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="complex-32" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="complex-64" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="array" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="array-2d" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="array-3d" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="array-4d" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
   <xs:element name="separatedValueTable" type="prim:typeOfPrimType"
substitutionGroup="dfdl:type"/>
</xs:schema>
```

3. Structural definition

The structural definition is an XML document written in SDL using the SDL schema extended with the schema extensions above. It defines the structure of the new types:

In SDL formal language:

```
textInteger := ( ( minusSign.? )::( digit.+ ) )
```

```
textFloat := ( ( minusSign.? )::( digit.+ )::( [fullStop; ( digit.+ )].? )::(
[( letter[ 'e' ] | letter[ 'E' ] ); textInteger].? ) )

nullTermString := [( char.* ); null]

complex-32 := [float; float]

complex-64 := [double; double]

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

array-2d( type, size-0, size-1 ) := ( ( type.size-0 ).size-1 )

array-3d( type, size-0, size-1, size-2 ) :=
   ( ( ( type.size-0 ).size-1 ).size-2 )

array-4d( type, size-0, size-1, size-2, size-3 ) :=
   ( ( ( ( type.size-0 ).size-1 ).size-2 ).size-3 )

separatedValueTable( valueSeparator, rowSeparator ) :=
   ( [( [char - ( valueSeparator | rowSeparator ); valueSeparator].* );
   rowSeparator].* )
```

In XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="C:\Documents and Settings\martin\My
Documents\Grid\dfdl\Drafts\XML\xml2sdl.xsl"?>
<dfdl:dfdl xmlns="http://www.dfdl.org/2003/structures"
xmlns:dfdl="http://www.dfdl.org/2003/dfdl"
xmlns:prim="http://www.dfdl.org/2003/primitives"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dfdl.org/2003/structures
structures.xsd">
   <dfdl:definitions>
      <dfdl:define>
         <textInteger/>
         <dfdl:toBe>
            <dfdl:concatenate>
               <dfdl:repeat number="zeroOrOne">
                  <prim:minusSign/>
               </dfdl:repeat>
               <dfdl:repeat number="oneOrMore">
                  <prim:digit/>
               </dfdl:repeat>
            </dfdl:concatenate>
         </dfdl:toBe>
      </dfdl:define>
      <!--      -->
      <dfdl:define>
         <textFloat/>
         <dfdl:toBe>
            <dfdl:concatenate>
               <dfdl:repeat number="zeroOrOne">
                  <prim:minusSign/>
               </dfdl:repeat>
               <dfdl:repeat number="oneOrMore">
                  <prim:digit/>
               </dfdl:repeat>
               <dfdl:repeat number="zeroOrOne">
                  <dfdl:sequence>
                     <prim:fullStop/>
                     <dfdl:repeat number="oneOrMore">
                        <prim:digit/>
                     </dfdl:repeat>
                  </dfdl:sequence>
               </dfdl:repeat>
               <dfdl:repeat number="zeroOrOne">
                  <dfdl:sequence>
                     <dfdl:either>
                        <prim:letter>e</prim:letter>
                        <prim:letter>E</prim:letter>
                     </dfdl:either>
                     <textInteger/>
                  </dfdl:sequence>
               </dfdl:repeat>
            </dfdl:concatenate>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <nullTermString/>
         <dfdl:toBe>
            <dfdl:sequence>
               <dfdl:repeat number="unbounded">
                  <prim:char/>
               </dfdl:repeat>
               <prim:null/>
            </dfdl:sequence>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <complex-32/>
         <dfdl:toBe>
            <dfdl:sequence>
               <prim:float varName="real"/>
               <prim:float varName="imaginary"/>
            </dfdl:sequence>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <complex-64/>
         <dfdl:toBe>
            <dfdl:sequence>
               <prim:double varName="real"/>
               <prim:double varName="imaginary"/>
            </dfdl:sequence>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <array>
            <dfdl:parameter name="type"/>
            <dfdl:parameter name="size"/>
         </array>
         <dfdl:toBe>
            <dfdl:repeat number="size">
               <dfdl:paramType name="type"/>
            </dfdl:repeat>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <array-2d>
            <dfdl:parameter name="type"/>
            <dfdl:parameter name="size-0"/>
            <dfdl:parameter name="size-1"/>
         </array-2d>
         <dfdl:toBe>
            <dfdl:repeat number="size-1">
               <dfdl:repeat number="size-0">
                  <dfdl:paramType name="type"/>
               </dfdl:repeat>
            </dfdl:repeat>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <array-3d>
            <dfdl:parameter name="type"/>
            <dfdl:parameter name="size-0"/>
            <dfdl:parameter name="size-1"/>
            <dfdl:parameter name="size-2"/>
         </array-3d>
         <dfdl:toBe>
            <dfdl:repeat number="size-2">
               <dfdl:repeat number="size-1">
                  <dfdl:repeat number="size-0">
                     <dfdl:paramType name="type"/>
                  </dfdl:repeat>
               </dfdl:repeat>
            </dfdl:repeat>
         </dfdl:toBe>
      </dfdl:define>
      <!--       -->
      <dfdl:define>
         <array-4d>
            <dfdl:parameter name="type"/>
            <dfdl:parameter name="size-0"/>
            <dfdl:parameter name="size-1"/>
            <dfdl:parameter name="size-2"/>
            <dfdl:parameter name="size-3"/>
         </array-4d>
         <dfdl:toBe>
            <dfdl:repeat number="size-3">
               <dfdl:repeat number="size-2">
                  <dfdl:repeat number="size-1">
                     <dfdl:repeat number="size-0">
                        <dfdl:paramType name="type"/>
                     </dfdl:repeat>
                  </dfdl:repeat>
               </dfdl:repeat>
            </dfdl:repeat>
         </dfdl:toBe>
      </dfdl:define>
      <dfdl:define>
         <separatedValueTable>
            <!-- NB these must be subsets of char for this to be valid -->
            <dfdl:parameter name="valueSeparator"/>
            <dfdl:parameter name="rowSeparator"/>
         </separatedValueTable>
         <dfdl:toBe>
            <dfdl:repeat number="unbounded">
               <dfdl:sequence>
                  <dfdl:repeat number="unbounded">
                     <dfdl:sequence>
                        <!-- value -->
                        <dfdl:exclude>
```

```
                           <dfdl:either>
                              <dfdl:paramType name="valueSeparator"/>
                              <dfdl:paramType name="rowSeparator"/>
                           </dfdl:either>
                           <dfdl:from>
                              <prim:char/>
                           </dfdl:from>
                        </dfdl:exclude>
                        <dfdl:paramType name="valueSeparator"/>
                     </dfdl:sequence>
                  </dfdl:repeat>
                  <dfdl:paramType name="rowSeparator"/>
               </dfdl:sequence>
            </dfdl:repeat>
         </dfdl:toBe>
      </dfdl:define>
   </dfdl:definitions>
</dfdl:dfdl>
```

4. API

4.1 textInteger

Purpose: text representation of integers

call		**semantic**
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	return value, raise error on overflow
long	**getAsLong**();	return value, raise error on overflow
char	**getAsChar**();	return value, raise error on overflow
float	**getAsFloat**();	return value, raise error on overflow
double	**getAsDouble**();	return value, raise error on overflow
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	return value
void	**set** (byte value);	set value as a string representation of a decimal
void	**set** (short value);	set value as a string representation of a decimal
void	**set** (int value);	set value as a string representation of a decimal
void	**set** (long value);	set value as a string representation of a decimal
void	**set** (char value);	set value as a string representation of a decimal
void	**set** (float value);	set value as a string representation of a decimal
void	**set** (double value);	set value as a string representation of a decimal
void	**set** (boolean value);	raise an exception
void	**set** (String value);	set value, raise exception if invalid
byte[]	**getAsByteArray**();	return underlying byte representation
short[]	**getAsShortArray** ();	return an array with a single value, raise error on overflow
int[]	**getAsIntArrayt**();	return an array with a single value, raise error on overflow
long[]	**getAsLongArray** ();	return an array with a single value, raise error on overflow
char[]	**getAsCharArray** ();	return an array with a single value, raise error on overflow
float[]	**getAsFloatArray** ();	return an array with a single value, raise error on overflow
double[]	**getAsDoubleArray** ();	return an array with a single value, raise error on overflow
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	return an array with a single value
void	**set** (byte[] value);	Interpret as underlying byte representation
void	**set** (short[] value);	If the array has a single element set value else raise an exception
void	**set** (int[] value);	If the array has a single element set value else raise an exception
void	**set** (long[] value);	If the array has a single element set value else raise an exception
void	**set** (char[] value);	If the array has a single element set value else raise an exception
void	**set** (float[] value);	If the array has a single element set value else raise an exception
void	**set** (double[] value);	If the array has a single element set value else raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	If the array has a single valid element set value else raise an exception

4.2 textFloat

Purpose: text representation of floating point numbers

call		semantic
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	return value, raise error on overflow
double	**getAsDouble**();	return value, raise error on overflow
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	return value
void	**set** (byte value);	set value as a string representation of a decimal
void	**set** (short value);	set value as a string representation of a decimal
void	**set** (int value);	set value as a string representation of a decimal
void	**set** (long value);	set value as a string representation of a decimal
void	**set** (char value);	set value as a string representation of a decimal
void	**set** (float value);	set value as a string representation of a decimal
void	**set** (double value);	set value as a string representation of a decimal
void	**set** (boolean value);	raise an exception
void	**set** (String value);	set value, raise exception if invalid
byte[]	**getAsByteArray**();	return underlying byte representation
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	return an array with a single value, raise error on overflow
double[]	**getAsDoubleArray** ();	return an array with a single value, raise error on overflow
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	return an array with a single value
void	**set** (byte[] value);	Interpret as underlying byte representation
void	**set** (short[] value);	If the array has a single element set value else raise an exception
void	**set** (int[] value);	If the array has a single element set value else raise an exception
void	**set** (long[] value);	If the array has a single element set value else raise an exception
void	**set** (char[] value);	If the array has a single element set value else raise an exception
void	**set** (float[] value);	If the array has a single element set value else raise an exception
void	**set** (double[] value);	If the array has a single element set value else raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	If the array has a single valid element set value else raise an exception

4.3 nullTermString
Purpose: null terminated string format

call		semantic
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	return value
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	set value as a strin containing a single character
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	set value as a string
byte[]	**getAsByteArray**();	return underlying bytes
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	return characters that form the string in an array
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	Return an array with a single element, the string value
void	**set** (byte[] value);	Set underlying bit, raise an exception if invalid
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	Convert characters to a string value
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	If one element, set value to this, otherwise raise error

4.4 complex-32

Purpose: complex number composed of two 32 bit floats

call		**semantic**
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	Return a string representation of the floating points in a standard form
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	Set if string is valid (standard form), otherwise raise exception
byte[]	**getAsByteArray**();	Return underlying bytes
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	Return an array of size two containing the floating point numbers
double[]	**getAsDoubleArray** ();	Return an array of size two containing the floating point numbers
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	Return an array of size one with the string representation in it
void	**set** (byte[] value);	Set underlying bits, raise an exception if invalid
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	If the array is of size two set the values
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	If array is of size one and first string is a valid representation, set to this, otherwise raise exception

We may well want to introduce some API additions here

4.5 complex-64

Purpose: complex number composed of two 64 bit floats

call		semantic
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	Return a string representation of the floating points in a standard form
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	Set if string is valid (standard form), otherwise raise exception
byte[]	**getAsByteArray**();	Return underlying bytes
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	Return an array of size two containing the floating point numbers, raise exception on overflow
double[]	**getAsDoubleArray** ();	Return an array of size two containing the floating point numbers
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	Return an array of size one with the string representation in it
void	**set** (byte[] value);	Set underlying bits, raise an exception if invalid
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	If the array is of size two set the values
void	**set** (double[] value);	If the array is of size two set the values
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	If array is of size one and first string is a valid representation, set to this, otherwise raise exception

4.6 array

Purpose: a one dimensional array parameterized by size and type

call		semantic
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	raise an exception
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	raise an exception
byte[]	**getAsByteArray**();	raise an exception
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	raise an exception
void	**set** (byte[] value);	raise an exception
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	raise an exception

Need to define values for this…needs some thought

4.7 array-2d

Purpose: a two dimensional array parameterized by size and type

call		semantic
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	raise an exception
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	raise an exception
byte[]	**getAsByteArray**();	raise an exception
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	raise an exception
void	**set** (byte[] value);	raise an exception
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	raise an exception

Need to define values for this…needs some thought

4.8 array-3d

Purpose: a three dimensional array parameterized by size and type

call		**semantic**
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	raise an exception
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	raise an exception
byte[]	**getAsByteArray**();	raise an exception
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	raise an exception
void	**set** (byte[] value);	raise an exception
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	raise an exception

Need to define values for this…needs some thought

4.9 array-4d

Purpose: a four dimensional array parameterized by size and type

call		**semantic**
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	raise an exception
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	raise an exception
byte[]	**getAsByteArray**();	raise an exception
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	raise an exception
void	**set** (byte[] value);	raise an exception
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	raise an exception

Need to define values for this…needs some thought

4.10 separatedValueTable

Purpose: a table of values with parameterized value and row separators

call		**semantic**
byte	**getAsByte**();	raise an exception
short	**getAsShort**();	raise an exception
int	**getAsInt**();	raise an exception
long	**getAsLong**();	raise an exception
char	**getAsChar**();	raise an exception
float	**getAsFloat**();	raise an exception
double	**getAsDouble**();	raise an exception
boolean	**getAsBoolean**();	raise an exception
String	**getAsString**();	raise an exception
void	**set** (byte value);	raise an exception
void	**set** (short value);	raise an exception
void	**set** (int value);	raise an exception
void	**set** (long value);	raise an exception
void	**set** (char value);	raise an exception
void	**set** (float value);	raise an exception
void	**set** (double value);	raise an exception
void	**set** (boolean value);	raise an exception
void	**set** (String value);	raise an exception
byte[]	**getAsByteArray**();	raise an exception
short[]	**getAsShortArray** ();	raise an exception
int[]	**getAsIntArrayt**();	raise an exception
long[]	**getAsLongArray** ();	raise an exception
char[]	**getAsCharArray** ();	raise an exception
float[]	**getAsFloatArray** ();	raise an exception
double[]	**getAsDoubleArray** ();	raise an exception
boolean[]	**getAsBooleanArray** ();	raise an exception
String[]	**getAsStringArray** ();	raise an exception
void	**set** (byte[] value);	raise an exception
void	**set** (short[] value);	raise an exception
void	**set** (int[] value);	raise an exception
void	**set** (long[] value);	raise an exception
void	**set** (char[] value);	raise an exception
void	**set** (float[] value);	raise an exception
void	**set** (double[] value);	raise an exception
void	**set** (boolean[] value);	raise an exception
void	**set** (String[] value);	raise an exception

Need to define values for this…needs some thought

Author Information

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

Alan R. Chappell, chappella@battelle.org, Pacific Northwest National Laboratory, Battelle Seattle Research Center, 4500 Sand Point Way NE, Suite 100, Seattle, WA 98105-3949

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

Copyright (C) Global Grid Forum (date). All 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 on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

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

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM 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."

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://www.epcc.ed.ac.uk/dfdl