BinX - The Binary XML Description Language

Project Title: OGSA-DAI GridServe

Document Title: BinX - The Binary XML Description Language

Document Identifier: EPCC-GDS-WP5-BinX v0.1

Distribution Classification: Commercial In Confidence

Authorship: Martin Westhead

Approval List: *EPCC:* Rob Baxter

Distribution List:
Unrestricted

Document History:

Personnel	Date	Summary	Version
MDW	13/04/2002	First draft	0.1

Contents

1 Introduction

This paper outlines work-in-progress on a proposed new XML Schema standard: Binary XML description language (BinX), which provides the ability to describe the physical representation and the overall structure of arbitrary binary data files.

After considering the issues involved in the representation of Scientific Data sets in Grid environments we concluded that although XML can provide a very useful mechanism for representing metadata, it is often inappropriate for representing large scientific datasets themselves. However, there is a need for a standard way to describe binary datasets and to that end we have developed BinX. We have also developed JAJA (Java Access to Just-about-any-Array), a simple prototype browser for binary array files to demonstrate how the standard might be used. Finally, we mention existing standards that could support the construction of self-describing files that would include an XML metadata file description, as well as the dataset itself.

An important note: BinX could be used in a number of different ways. The overall aim is to facilitate data exchange by providing a machine intelligible description of data representations. In general for storing specific metadata about datasets (parameters used, data generatoed etc.) we recommend the construction of custom XML schema, which could be used in combination with BinX for a complete representation of a dataset.

1.1 Motivation

XML is clearly today's standard of choice for the representation and exchange of structured data, particularly where that data must be read and interpreted by different applications written by different groups. XML and XML Schema provide a convenient, potentially human readable, easily extensible representation standard. It is tempting to assume, therefore, that all data exchanged on the Grid would be exchanged as XML. However, for many users1 in the scientific community the prospect of producing output data as XML presents many disadvantages and offers very few opportunities.

The datasets for many scientific users are stored in very large (tens of gigabytes) regularly structured binary files, often one or more large arrays or tables. They have tools for reading and manipulating these files, often written in languages like Fortran with primitive file handling capabilities. Whilst it is possible, in principle, to provide XML representations of such data it is not clear why you would want to. An XML representation would have a number of drawbacks:

- The XML representation would be significantly (around 2-4 times) larger than the simple binary representation and therefore take longer to write, transport etc.
- Inappropriate representations: The proposed standard representation for a multidimensional array in XML to effectively build a tree of lists (everything in XML is a tree). This is a poor representation for scientific users because commonly required operations such as extracting a slice or a diagonal becomes difficult to do.

XML also present few advantages:
Extensibility – The extensibility of XML is not enormously useful in this case. XML is most useful when the data is represents is richly structured. The simple arrays and tables that we are looking at do not have those features and are unlikely to change their in their basic representations.

[1] The BinX team have had requirements discussions with a number of eScience communities including Astrogrid, MyGrid, RealityGrid and QCDGrid.

Readability – A 10Gb array is not very human readable. It can obviously be visualised with the right software but representing it as XML does nothing to improve this situation.
Available tools – The available XML tools have not been designed for efficiently parsing such large files and their scalability may be severely tested.

It seems unlikely, therefore, that users with such datasets will represent them in XML. However, there is enormous value, and corresponding interest, in representing the metadata associated with the data in XML. The metadata will typically describe such things as how the data was produced (parameters, algorithms used etc), when and by whom. It would be very useful if that metadata could also contain a standard, canonical description of the structure and representation of the data itself. The work in progress, reported on here, is a straw man proposal for an XML Schema to address that need. The approach taken is described in the Section 2.

1.1.1 Related work

One of the earliest pieces of work this area was a system developed by IBM called EXPRESS [1] (data Extraction, Processing and REStructuring System). It supported access to a wide variety of data and restructuring of it for new uses. The system was driven by two very high level nonprocedural languages: DEFINE for data description and CONVERT for data restructuring. Program generation and cooperating process techniques were used to achieve efficient operation.

Another important data representation standard is STEP [2], STandard for the Exchange of Product model data, the unofficial name for the evolving IS0 standard 10303-Product Data Representation and Exchange. This aims to facilitate data/information exchange between CAD/CAM/CAE systems. The standardization effort begun in 1984 and was joined by PDES (Product Data Exchange using STEP), an American standardization initiative being developed by the IGES/PDES Organization. The first set of International Standard documents was approved in 1994. Information modeling, supported by the language EXPRESS, addresses information about a product's entire life cycle.

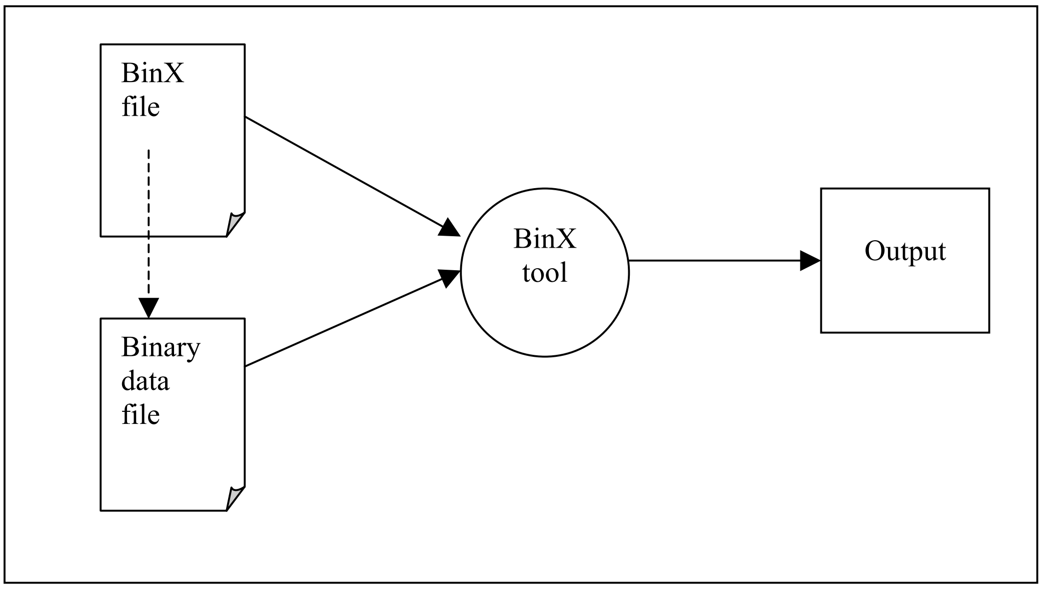
The External Data Representation Standard (XDR) is an IETF standard defined in RFC1832 [3]. It is a standard for the description and encoding of data in binary files. It differs from BinX in that:

- It defines aspects of the data encoding. BinX is intended to describe any (most) encodings rather than specify features of the encoding that should be used.
- BinX is XML based.

The Hierarchical Data Format (HDF) project [4] is run by NCSA. It involves the development and support of software and file formats for scientific data management. The HDF software includes I/O libraries and tools for analyzing, visualizing, and converting scientific data. HDF also, however, defines a binary data format in which the data is represented. HDF also provides software that allows the conversion of (most) HDF files to a standard XML representation.

1.2 BinX

BinX is the Binary XML description language. Its aim is to provide a canonical description for data stored in binary files. Once we have carried out some initial groundwork we propose



to take the work forward as a GGF standard.

Figure 1. Showing how the BinX file is used in to describe the format of the binary file.

Figure 1 is intended to illustrate how the BinX file could be used in practice. The BinX file describes the structure and format of a binary data file and can also contains a URL which points to that file. This allows the construction of tools that can read a very wide range of file formats. Such tools could be presented as Web services and could have functionality to convert between formats, to extract pieces of the data (e.g. slices or diagonals of an array), or to browse the file. JAJA, discussed below, is a BinX tool that provides simple browsing functionality of an array described in BinX.

BinX provides the ability to describe three levels of features in a binary file:

1. The underlying physical representation (e.g. bit/byte ordering)
2. The primitive types used (e.g. IEEE float, integer)
3. The structure of the data itself (e.g. array, list of fields, table)

The representation of data in binary files is much more standard than it used to be. The prevalence of the IEEE floating point standard [5] has simplified a number of the issues. From the point of the physical representation it is still necessary to specify:

- The byte ordering – big-endian/little endian
- The bit ordering – big-endian/little endian (although this is almost always big endian)
- Blocksize – many binary formats pad out data fields so that they are always a multiple of a given block size.

BinX provides for the representation of a broad range of different primitive types. Including all those that can be represented in XML Schema [6]. We anticipate that the standards process will eventually bring this to an appropriate representative set.

In terms of structural representations we have been guided by the work on XDR. Anything that can be represented in XDR can be represented in BinX so we have provision for variable and fixed length arrays, structs, strings, unions etc. We have also made provision for the description of data streams.

The definition of BinX includes a “typeDef” mechanism that allows the user to define/rename new types. The intention in our design is to try to provide the minimum number of basic types and then to provide an include file that defines a set of standard extensions.

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset     xmlns="http://http://schemas.nesc.ac.uk/binx/binx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://http://schemas.nesc.ac.uk/binx/binx
binx.xsd"
byteOrder="bigEndian" bitOrder="bigEndian" blockSize="32">
  <definitions>
    <typeDef typeName="complexType">
      <struct>
        <ieeeFloat-32 varName="real"/>
        <ieeeFloat-32 varName="imaginary"/>
      </struct>
    </typeDef>
  </definitions>
  <file src="http://www.epcc.ed.ac.uk/testFile.bin">
    <ieeeFloat-32 varName="inputParameter1"/>
    <integer-32 varName="inputParameter2"/>
    <arrayFixed>
      <defType typeName="complexType"/>
      <dim indexFrom="0" indexTo="99" name="x"/>
      <dim indexFrom="0" indexTo="4" name="y"/>
    </arrayFixed>
  </file>
</dataset>
```

Figure 2. *An example BinX file.*

Figure 2 shows a small simple BinX file. The root tag is <dataset> which is can contain a set of type definitions, contained within the <definitions> tag, followed by one or more file descriptions contained within the <file> tag. In this example we can see that a new type “ComplexType” has been declared in the definitions section that is defined to be a struct containing two floats, one called “real” and one called “imaginary”. There is only one file in this example, which is located at:

`http://www.epcc.ed.ac.uk/testFile.bin`

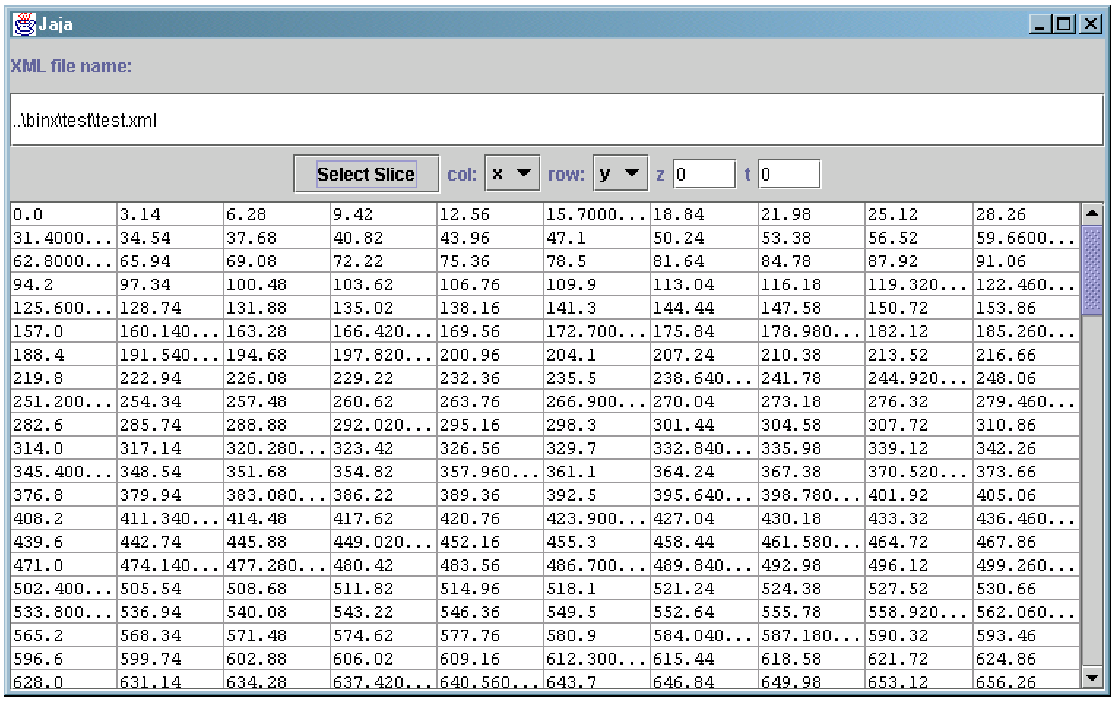
This file contains two numbers a float (inputParameter1) followed by an integer (inputParameter2) followed by a two dimensional array of our new complex number type.

Notice that the <dataset> tag allows us to define the byte order, the bit order and the blocksize. These can be changed for individual files, or indeed for individual fields.

1.2.1 Current status

BinX, at time of writing, is in early stages. We have taken the basic idea far enough to feel confident that it is a practical proposition and to outline an approach that could be taken. We anticipate presenting BinX as a proposed standard and that this could result in significant changes before a standard is agreed upon.

1.3 JAJA



XML file name:

..\binx\test\test.xml

col: x row: y z 0 t 0

0.0	3.14	6.28	9.42	12.56	15.7000...	18.84	21.98	25.12	28.26
31.4000...	34.54	37.68	40.82	43.96	47.1	50.24	53.38	56.52	59.6600...
62.8000...	65.94	69.08	72.22	75.36	78.5	81.64	84.78	87.92	91.06
94.2	97.34	100.48	103.62	106.76	109.9	113.04	116.18	119.320...	122.460...
125.600...	128.74	131.88	135.02	138.16	141.3	144.44	147.58	150.72	153.86
157.0	160.140...	163.28	166.420...	169.56	172.700...	175.84	178.980...	182.12	185.260...
188.4	191.540...	194.68	197.820...	200.96	204.1	207.24	210.38	213.52	216.66
219.8	222.94	226.08	229.22	232.36	235.5	238.640...	241.78	244.920...	248.06
251.200...	254.34	257.48	260.62	263.76	266.900...	270.04	273.18	276.32	279.460...
282.6	285.74	288.88	292.020...	295.16	298.3	301.44	304.58	307.72	310.86
314.0	317.14	320.280...	323.42	326.56	329.7	332.840...	335.98	339.12	342.26
345.400...	348.54	351.68	354.82	357.960...	361.1	364.24	367.38	370.520...	373.66
376.8	379.94	383.080...	386.22	389.36	392.5	395.640...	398.780...	401.92	405.06
408.2	411.340...	414.48	417.62	420.76	423.900...	427.04	430.18	433.32	436.460...
439.6	442.74	445.88	449.020...	452.16	455.3	458.44	461.580...	464.72	467.86
471.0	474.140...	477.280...	480.42	483.56	486.700...	489.840...	492.98	496.12	499.260...
502.400...	505.54	508.68	511.82	514.96	518.1	521.24	524.38	527.52	530.66
533.800...	536.94	540.08	543.22	546.36	549.5	552.64	555.78	558.920...	562.060...
565.2	568.34	571.48	574.62	577.76	580.9	584.040...	587.180...	590.32	593.46
596.6	599.74	602.88	606.02	609.16	612.300...	615.44	618.58	621.72	624.86
628.0	631.14	634.28	637.420...	640.560...	643.7	646.84	649.98	653.12	656.26

Figure 3

JAJA (Java Access to Just-about-any Array) is a prototype BinX tool, built to demonstrate the potential of the work. The JAJA interface (Figure 3) can be used to display arbitrary slices through a multidimensional array specified in BinX.

1.4 Self describing files

An important requirement that came originally from the Astrogrid team was that it should to be possible to include the description of a binary file in the file itself. If the description of the file is located in a different file there is a danger that the correspondence between the two files could be lost (as they are copied/moved around) and then the data would be rendered effectively useless.

There are a number of approaches that could be adopted to solve this problem. A number of ad-hoc methods are frequently employed for this purpose, such as making the first four bytes of the file be an integer representing the offset in bytes to the start of the data etc. However this is a general problem and there are existing standards that can be applied. The most relevant of which is DIME which is designed to provide a way of attaching binary data to XML files, such as SOAP messages. DIME [7] is a simple, lightweight message format that encapsulates multiple messages with header information (such as MIME type) that allow the individual messages to be later extracted using a DIME parser.

1.5 Future directions

The next steps for this work will be to bring it to review within the standards processes of the Global Grid Forum. In the immediate future we aim to work on:

- Tools and libraries for reading and writing data files described using BinX.
- Testing activities with our user communities to investigate whether BinX can capture all that is required.
- A Java GUI for writing BinX descriptions so that users are not forced to write the XML by hand.

1.6 Acknowledgements

The work reported here was carried out at EPCC in the University of Edinburgh under the auspices of the UK National e-Science Centre.

1.7 References

[1] EXPRESS: A Data EXtraction, Processing, amd REStructuring System EXPRESS – Nan C. Shu, Barron C. , R. W. , Sakti P. Ghosh and Vincent Y. Lum. TODS vol 2, No 2, 1997, pp134-174.

[2] http://filebox.vt.edu/users/vkern/step.html

[3] http://www.faqs.org/rfcs/rfc1832.html

[4] HDF

[5] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985.

[6] http://www.w3.org/TR/xmlschema-2/

[7] H. Nielsen, H. Sanders, E. Christensen, " Direct Internet Message Encapsulation (DIME)", INTERNET DRAFT "http://www.ietf.org/internet-drafts/draft-nielsen-dime-01.txt", Microsoft, May 2001. This is work in progress.

2 BinX requirements

BinX has the following list of requirements:

1. BinX should provide a notation capable of describing the format of any[2] binary file so that the extraction and interpretation of the contents of that file could be achieved by general purpose libraries and applications who have read the description. In particular BinX should provide a mechanism for describing database relations so that they can be exchanged between databases.
2. The notation used by BinX should take maximal advantage of standard definitions of types and formats. Including for example the standard representations in XML Schema.
3. The notation used by BinX should be succinct, e.g. it should be able to import and reuse previously defined patterns and it should be able to re-use, by reference, subpattern definitions that have been declared elsewhere in the same description.
4. BinX should have a defined W3C XML Schema.
5. It should be possible to use BinX in such a way that humans can read it and easily comprehend the structure it describes.
6. BinX should be capable of convenient and independent extension.

3 Overview of schema design

In this section provide a description of the various files that comprise the BinX XML Schema and standard typedefs. The description is current for BinX version 0.2

[2] In fact BinX is inherently limited in its description of numerical values. We have tried to include a wide selection of common representations but it is clearly open to debate as to whether these are the right ones. In general, it is impossible to cover all bases, vendor specific floating point representations for example are not (and probably should not) be included.

3.1 Files

The BinX consists of the following files:

- binx.xsd – principle file describing structure of a binx document
- types.xsd – file describing notation for the primitive types: numbers, characters, bytes etc.
- datatypes – file describing notation for the compound types: arrays, strings structs etc.

Here we provide a high level overview of the contents of these files and some of the design principles applied and issues encountered. For a detailed description of the elements in the files see Appendix A.

An important thing to remember is that the schema described here provides a description of the syntax of the XML BinX file. It does not (and cannot) describe the semantics intended by the BinX tags. (That semantics is outlined informally here and should be specified precisely in Appendix A.) That is to say that a variable array is an array in which the array values are preceded by an integer giving the array size. It is tempting to thing that the Schema would describe a variable array, therefore, as an integer followed by an array. However the Schema describes the semantics of the BinX XML file. The array size is not given here and so it should not be included in the Schema.

3.1.1 File: binx.xsd

The file 'binx.xsd' is the root document for the BinX Schema. It describes the structural representation of a BinX file. There are three global elements: the dataset, defType and definitions.

3.1.1.1 The `<dataset>` tag

The primary root element of a BinX file is the `<dataset>` tag. A BinX document will normally consist of a dataset which contains an optional set of definitions (typeDef statements – see below) followed by a sequence of one or more `<file>` tag containing a sequence of type descriptions or an `<arrayMultiFile>` tag denoting an array spanning multiple files.

Each file can contain a sequence of types. These can be primitive types, datatypes, or user defined types.

The arrayMultiFile feature allows of the situation where a single array is stored across multiple files with each file containing a segment of the array divided with respect to the slowest changing index. (In BinX this is the *last* index to be specified).

3.1.1.2 The `<defType>` tag

This tag is used refer to user defined tags (see typedef mechanism below). It is included as a global tag for ease of reference elsewhere in the schema. It is not intended to be used as a root tag.

3.1.1.3 The <definitions> tag

The definitions tag is can be optionally used as the first element in a dataset. It is here that the user can define and rename types that can be referred to in the rest of the document (see typedef mechanism below for details). This tag is included as a global element to allow the definition of type extension files. These are BinX files that just contain definitions of new types and structures. An example of such a file is the standard XDR types include file that is described below.

The use of external include files raises some design questions. The intention with these include files a reference to the file could be included directly in the definitions section. The standard way to do this in XML is to use XInclude (http://www.w3.org/TR/xinclude/), however this is at time of writing a new standard that is not supported by the current generation of parsers. It is therefore a requirement that BinX applications provide (possibly limited) support for XInclude. We aim to provide a naïve implementation of some of XInclude for use with the Apache tools (until Apache implement it properly themselves).

A note about the inclusion statements: They should be constructed so that the root tag `<definitions>` in the included document is *not* included.

3.1.2 File: types.xsd

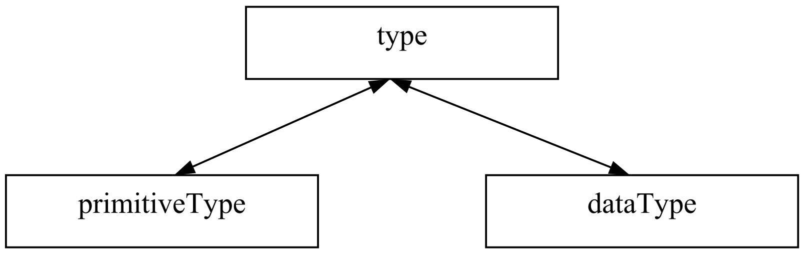
This file contains the base types for representing numbers, bytes, characters etc. We have adopted the convention of including the type's length in bits at the end of the name. So for example the types include: character-8, short-16, integer-32. Slightly unusual types are void-0 and enum-32.

The type void-0 allows the opportunity of representing a type of zero length. This can be useful when defining unions, since it can be useful there to have a case in which no data is stored.

The enum-32 is an enumerated type. The representation in the binary file is a standard 32 bit integer. This can take an arbitrary number of constant values each of which has a string associated with it. The strings can be defined within the enum-32 tag.

3.1.2.1 Substitution Groups

In order to organise the types and easily allow the XML Schema to have the appropriate notion of a type, the types are arranged into a hierarchy of substitution groups. This is illustrated in the following diagram:



So primitiveType and dataType can substitute for type. In the Schema definition there has to be an element associated with each of these substitution groups in order to define the groups. However we (clearly) never expect to see the primitiveType element appear in a BinX document, even though it would be legal to do so[3].

In addition to elements associated with each group there are also XML Schema types defined which collect all the common properties (attributed etc.) of the group. These are called typeType, primitiveTypeType and dataTypeType respectively. The shared attributes allow local redefinition of bit order, byte order and blockSize (see below for an explaination of the sematics). They also include a varName attribute which enables the user to name the field being defined, and an ignore attribute. The ignor attribute allows the user to indicate to applications that this field is not expected to be displayed. This might include padding data between parameters and output for example.

[3] There might be a better way of defining this stuff of course.

3.1.2.2 Common type attributes

All types (primitive or data) possess the following attributes:

1. bitOrder – order of the bits in the representation can be big or little endian (default is bigEndian)
2. byteOrder – order of the bytes in the representation can be big or little endian (default is bigEndian)
3. blockSize – size of the blocks in the representation, given in bits (default is 1). Types that have a size which is not a multiple of the block size will be padded to bring them up to the nearst multiple of the block size.
4. ignore – a Boolean type which provides a hint to browsers that the user is not interested in the data in this field (used for padding).
5. varName – user name for the field.
6. key – for identifying key fields in data, for example, in database relations. Can take one of the following enumerated values: key, primaryKey, secondaryKey, tertiaryKey.
7. info – an unspecified string field intended for application extensions.
8. comment – for user annotations to the data field.

The first four fields are part of an attribute group "representationDef" that is applied at several levels within BinX. All fields are optional.

3.1.3 File:datatypes.xsd

The datatypes.xsd file contains all the composite data types: arrays, structs and unions. In the rest of this section we look at each of these in turn.

3.1.3.1 Arrays

There are four tags associated with arrays:

1. array – defines a substitution group, not to be used directly
2. arrayFixed – A multidimensional array of fixed size.
3. arrayVariable – A multidimensional array in which the size of the slowest moving index is given as an integer in the binary file itself.
4. arrayStreamed – A multidimensional array in which the length of the slowest changing variable is not fixed.

The element "array" itself is used to define a substitution group within the Schema. It is not intended to be used directly.

The element "arrayFixed" allows the definition of a multidimensional array where the size of every index is fixed. The array can contain an arbitrary number of dimensions. Each dimension is represented by the use of a <dim> tag. This tag allows the start and end values of indices to be given (a step size of one is assumed). It also allows each index to be named. The dimensions must be given in order from fastest moving to slowest moving.

The element "arrayVariable" allows the definition of a multidimensional array similar to that in arrayFixed. The difference is that the last dimension specified (the slowest moving) is specified with a dimVariable tag. This tag does not allow the upper limit of the index to be given. Instead this value is assumed to be given as an unsigned integer in the first 32-bits of the data associated with the array.

The "arrayStreamed" array is similar again except that in this case the last dimension is specified with a dimStreamed tag. The dimStreamed tag also prevents the inclusion of an upper limit on the index value. This is to allow the processing of streamed arrays where the size of the array may be unknown.

3.1.3.2 Structs

Structs are simply sequences of types. They can be named by including them in typeDef tags.

3.1.3.3 Unions

The unions are discriminant unions based on those given in XDR. The union is defined by a declaration of the discriminant followed by a series of cases. The discriminant can be either an integer, and unsigned integer or an enumeration. Based on the value of the descriminant a case is selected which defines the ultimate type of the union. So in the binary file the union will be represented by a 32-bit discriminant followed by a type whose size depends on the discriminant value.

3.1.3.4 Other types

The aim in designing the BinX schema was to keep the definition as tight as possible. We have tried to avoid unnecessary redundancy in the definitions, particularly in the definitions of datatypes. In this section we review how booleans, options, strings and tables can be constructed using BinX.

A boolean can be constructed as an enumeration:

```
<enum-32 blockSize="32" byteOrder="bigEndian">
  <val value="0" name="false"/>
  <val value="1" name="true"/>
</enum-32>
```

or alternatively a bit-1 can be used.

An option is a union which uses a boolean and a void:

```
<union blockSize="32" byteOrder="bigEndian">
  <discriminant>
    <defType typeName="xdrBool"/>
  </discriminant>
  <case discriminantValue="TRUE">
    *** THE OPTIIONAL TYPE GOES HERE***
  </case>
  <case discriminantValue="FALSE">
    <void-0/>
  </case>
</union>
```

Strings are just single dimension arrays (fixed or variable) of character-8.

Tables are single dimension arrays (fixed, variable or streaming) of structs. Where the struct represents a table row.

3.2 Typedef mechanism

BinX makes use of XML Schema references to provide a simple mechanism for allowing users to construct new types and rename existing types. The mechanism is limited in that there is no way to allow parameterisation of the new of renamed types. For example it is possible to create a new type “my16x16FloatArray” which is a fixed size 16 by 16 two-dimensional array of floats. It is not possible however to rename the arrayFixed to provide a user defined token that can be used to define arrays of arbitrary size because there is no way to provide size parameters for the new type.

New types are defined within the definition section at the top of a BinX document using the `<typeDef>` tag. This tag takes a single argument “typeName” – the name of the new type. The type definition itself should then be enclosed within the typeDef start and end tags. Note that typeDef expects exactly one type to be enclosed. If the user wishes to enumerate a sequence of types the struct type should be used.

New types defined in this way are referred to using the defType element. The defType tag has an attribute “typeName”. The value of this attribute must match to the “typeName” of a typeDef declaration earlier in the same document. XML Schema references are used to enforce this matching requirement.

3.3 XDR file types

BinX will allow the representation of any type that can be represented using XDR. For convenience a series of typeDefs have been constructed to provide easy reference to these types. The following is a list of the types defined in this way:

- xdrInt
- xdrUnsignedInt
- xdrFloat
- xdrDouble
- xdrQuadruple
- xdrOpaqueVariable
- xdrString
- xdrVoid

Unfortunately due to the limitations of the typeDef mechanism, specifically the lack of parameterisation, some of the xdr types could not be defined in this way. These are:

- Enumeration
- Struct
- Union
- Option
- Fixed Array

For enumeration, structures, unions and fixed arrays simply use the equivalent BinX type ensuring that blocksize and byteOrder attributes are set to 32 and bigEndian respectively. To provide the semantics of the option type use the example given ealier.

3.4 Modifiers

At each level in Binx, the entire dataset, a whole file or the individual type, the user can set the bit order, byte order and block size of the types being used. Bit order can be bigendian or little endian and refers to the whether the first bit or the last bit is the most significant, respectively. Byte order can be big endian or little endian and refers to whether the bytes in a primitive type are most significant byte first or last respectively. Block size refers to the smallest block of the data that can be individually addressed. It is given in bits. Types whose size is not an whole multiple of block size are assumed to be padded to the nearest whole multiple.

3.5 Limitations and workarounds

BinX has various limitations. Whereas the aim is to be able to provide a notation for describing any binary file, it is impossible to cover all bases. The biggest limitation that we are aware of is the use of fixed primitive types. Thus it would be impossible, for example, in BinX, to represent a number given in a vendor-specific floating-point type. It would, of course, be possible to notate it as a fixed array of bytes, but the semantics of the floating-point representation would be opaque.

The other limitation of BinX is that the typeDef mechanism cannot be parameterised. If additional parameterised type definitions are necessary, the user will have to extend the schema itself. The advantage of using typeDefed types is that because the types are built strictly on top of the existing semantics, an applications can read typeDef include files and make immediate use of the new types. Extensions to the XML Schema will, inevitably, require code changes to apply.

3.6 Self describing files

One of the requirements for BinX which emerged from a discussion with the Astrogrid team is that there should be a mechanism for files to be self-describing. The problem is that if metadata such as the BinX description is kept in a separate file then there is a significant overhead in ensuring that metadata and data files are kept in correspondence, and as files are moved around between sites it is easy to see how that correspondence could be lost.

There are a number of ways in which the metadata and data files could be combined:
1. using of MIME or DIME headers to delineate the start and end of different files
2. fixed header length
3. ad hoc variable length header (first 4 bytes represents header length)
4. represent the binary data in XML using an ascii encoding standard (such as binhex)
5. simply append the binary data to the xml and encode the header length in BinX (the file tag "offset" attribute should allow you to do this).

What we would like is a simple mechanism that makes it easy for, for example a Fortran code, to ignore the header and access the data directly. Furthermore we would like the data to be raw so that it is easy to skip through (for example a Terabyte array) to access the pieces required. This means that we need the header size to be easily accessed (not encoded in XML) and we cannot use an encoding scheme on the data – ruling out 3 and 4. Options 2 and 3 seem very fragile. Any fixed size header is bound to be inconveniently large for some applications and too small for others, probably both. Encoding the header size at the start of the file is easy to do but requires well-understood convention to follow.

So, our favoured option is 1 since this is what the standards are defined for. We would expect as part of the future work to define a library that Fortran programmers could access that would carry out the appropriate extraction. However, we have recently encountered a problem with this approach. The problem involves the maximum file size. DIME chunks data with a max chunk size of 2^32-1. This unfortunately breaks our requirement that the data be raw because every 2^32-1 bytes we would need to begin and end a chunk. MIME allows you to specify data size using the Content-Length header. However I have been unable to determine whether this has a maximum value (and if so, what it is).

Our proposal to use MIME and deal with the size issue as follows: We assume that if the data file size exceeds the maximum Content-Length there will be just one data file present. The MIME headers will allow us to find the start of the file and the BinX representation should give us all the information required about its length. So in this case we propose to either make the Content-Length field negative to make it explicit that the size is not defined there or to miss it out entirely.

4 Examples of use

In this section we present first some examples of how BinX might be applied and then some examples showing the use of the BinX notation.

4.1 Example use cases

In this section we try to outline the sorts of ways in which a BinX file description might be used. Let us suppose that we have a file of binary data *f* and a BinX description of that file *df*. (These may be truly separate files, or separate MIME separated parts of the same file – the difference is not important for this discussion.) The description mechanism will then allow us to do the following:

1. Allow human inspection, editing and visualisation of *f* , navigating using the names and structures in *df* and displaying values in accordance with *df.*
2. Provide a canonical mapping from the data in *f* to a standard XML representation (such as using XSIL).
3. Allow extraction of subsets or derivatives of the data by interpreting queriesor expressions in some standard language. For example these could be Xqueries over the canonical XML representation implied by the data (see 2 above) or they could be datacutter array slices, the could even be SQL.
4. Given a second data format *df'* and a mapping *df→df'* the data in *f* could be converted to the new format.
5. Given a description *df* and some target programming language a library for accessing the values in any file complying with *df* could be automatically generated.

4.2 Example BinX files

This section contains some example BinX description files to serve as illustrations of the flexibility of the representation.

4.2.1 Astronomical table

This example is inspired by an example taken from the Astrogrid project. It illustrates how BinX can be used to represent a table using an array of structs. In this case the structs in question are fairly complicated and some of the elements involve the description of variable sized arrays. So taking the example in sections

4.2.1.1 Definitions

- String we begin by defining a string as a variable length array of character-8 primitives.
- Then we define the "tableRowType" which is a struct with the following members:
    - A fixed size character array (string) called "Star-name" of size 11 characters.
    - Two floats called "RA" and "Dec".
    - A 3 dimensional array, "Counts", in which the last dimension is of variable size.
- Finally we define the "TableType" which uses the "string" type we defined earlier to encode two strings "Table Name" and "Table Description, followed by a streamed single dimensional array of table row structs.

4.2.1.2 File

A single file is described its name is given as "astroFileLocation.bin" and it is described as containing an observer name (held in a variable length string) followed by a table of the type defined above.

4.2.1.3 BinX

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Martin
Westhead (EPCC) -->
<dataset xmlns="http://http://schemas.nesc.ac.uk/binx/binx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://http://schemas.nesc.ac.uk/binx/binx
C:\DOCUME~1\martin\MYDOCU~1\Grid\Binx\src\WP5\Binx-XML\binx.xsd">
  <definitions>
    <typeDef typeName="string">
      <arrayVariable>
```

```
        <character-8/>
        <dimVariable/>
      </arrayVariable>
    </typeDef>
    <typeDef typeName="tableRowType">
      <struct>
        <arrayFixed varName="Star-Name">
          <character-8/>
          <dim indexTo="10"/>
        </arrayFixed>
        <ieeeFloat-32 varName="RA"/>
        <ieeeFloat-32 varName="Dec"/>
        <arrayVariable varName="Counts">
          <integer-32/>
          <dim indexTo="2"/>
          <dim indexTo="3"/>
          <dimVariable/>
        </arrayVariable>
      </struct>
    </typeDef>
    <typeDef typeName="TableType">
      <struct>
        <defType typeName="string" varName="Table Name"/>
        <defType typeName="string" varName="Table Description"/>
        <arrayStreamed>
          <defType typeName="tableRowType"/>
          <dimStreamed/>
        </arrayStreamed>
      </struct>
    </typeDef>
  </definitions>
  <file src="astroFileLocation.bin">
    <defType typeName="string" varName="Observer"/>
    <defType typeName="TableType"/>
  </file>
</dataset>
```

4.2.2 XDR data file

This example shows how the XDR type definitions can be used, again we example the description by section.

4.2.2.1 Definitions

The first line of the definition includes the standard xdr definition set. Then the following types are defined:

- A struct called parameters containing:
    - an xdrFloat called "temperature" (with a comment identifying the units of measure).
    - an xdrInt called "increments",
    - and an xdrString called "dateOfExecution".
- A padding field defined as a fixed array of 48 bytes (which are marked to be ignored).
- Finally a type "data" is defined as a fixed two dimensional array of doubles.

4.2.2.2 File

The file is defined to contain an initial integer (called header offset) followed by a parameters struct followed by padding and in turn followed by data.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Martin
Westhead(EPCC) -->
<dataset xmlns="http://schemas.nesc.ac.uk/binx/binx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.nesc.ac.uk/binx/binx
C:\DOCUME~1\martin\MYDOCU~1\Grid\Binx\src\WP5\Binx-XML\binx.xsd"
xmlns:xi="http://www.w3.org/2001/XInclude">
  <definitions>
    <xi:include href="xdrTypes.xml#xpointer(//definitions/*)"/>
    <typeDef typeName="parameters">
      <struct>
        <defType typeName="xdrFloat" varName="Temperature"
comment="degrees Celsius"/>
        <defType typeName="xdrInt" varName="increments"/>
        <defType typeName="xdrString" varName="dateOfExecution"/>
      </struct>
    </typeDef>
    <typeDef typeName="padding">
      <arrayFixed ignore="true" blockSize="32" byteOrder="bigEndian">
        <byte-8/>
        <dim indexTo="48"/>
      </arrayFixed>
    </typeDef>
    <typeDef typeName="data">
      <arrayFixed blockSize="32" byteOrder="bigEndian">
        <defType typeName="xdrDouble"/>
        <dim indexTo="64" name="x"/>
        <dim indexTo="128" name="y"/>
      </arrayFixed>
    </typeDef>
  </definitions>
  <file src="xdrExampleData.bin">
    <defType typeName="xdrInt" varName="headerOffset"/>
    <defType typeName="parameters"/>
    <defType typeName="padding"/>
    <defType typeName="data"/>
  </file>
</dataset>
```

4.2.3 Multiple file array example

This example was inspired by the data from the RealityGrid.

4.2.3.1 Definitions

The following types are defined:

- A struct called “siteData” containing:
    - An integer “index”
    - An array of 15 floats called “f”
    - An array of 15 floats called “g”
    - An array of 2 floats called “vector-d”
    - Two integers “phi” and “rho”

4.2.3.2 ArrayMultiFile

Rather than a single file this time, this data is a single array which has been split across multiple files. Indexes over the first three dimensions run through all the files, the indexes of the last dimension are split across the files: file1, file2, file3 and file4.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Martin
Westhead (EPCC) -->
<dataset xmlns="http://schemas.nesc.ac.uk/binx/binx"
xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.nesc.ac.uk/binx/binx
C:\DOCUME~1\martin\MYDOCU~1\Grid\Binx\src\WP5\Binx-XML\binx.xsd">
  <definitions>
    <typeDef typeName="siteData">
      <struct>
        <integer-32 varName="index"/>
        <arrayFixed varName="f">
          <ieeeFloat-32/>
          <dim indexTo="14"/>
        </arrayFixed>
        <arrayFixed varName="g">
          <ieeeFloat-32/>
          <dim indexTo="14"/>
        </arrayFixed>
        <arrayFixed varName="vector-d">
          <ieeeFloat-32/>
          <dim indexTo="2"/>
        </arrayFixed>
        <integer-32 varName="phi"/>
        <integer-32 varName="rho"/>
      </struct>
    </typeDef>
  </definitions>
  <arrayMultiFile>
    <defType typeName="siteData"/>
    <dim indexTo="256" name="x"/>
    <dim indexTo="256" name="y"/>
    <dim indexTo="256" name="z"/>
    <dimMultiFile name="t">
      <file src="file1" offset="0" indexFrom="0" indexTo="64"/>
      <file src="file2" offset="0" indexFrom="0" indexTo="64"/>
      <file src="file3" offset="0" indexFrom="0" indexTo="64"/>
      <file src="file4" offset="0" indexFrom="0" indexTo="64"/>
    </dimMultiFile>
  </arrayMultiFile>
</dataset>
```

4.2.4 A BMP picture file

This example is based on the standard for monochrome BMP graphics file.

4.2.4.1 Definition

The following types are defined:

- fileHeader is a struct containing the following fields:
    - a fixed sized character array “string-‘BM’”
    - an integer representing file size
    - two arrays of two bytes “Reserved – 0”
    - an integer called “imageOffset”
- imageHeader is a struct containing:
    - a series of named integers and shorts
    - a two byte array defining the colour map
    - an integer defining the “numberOfFirstColours”

- image data is defined as a streamed array of bits. The array is defined as being streamed because we do not know its size. (This is a bit of a hack – the size information is encoded in the binary file. See future work).

4.2.4.2 File

The file itself, “picture.bmp”, consists of a fileheader, followed by and image header, followed by image data.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Martin
Westhead (EPCC) -->
<dataset xmlns="http://schemas.nesc.ac.uk/binx/binx"
xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.nesc.ac.uk/binx/binx
C:\DOCUME~1\martin\MYDOCU~1\Grid\Binx\src\WP5\Binx-XML\binx.xsd">
  <definitions>
    <typeDef typeName="fileHeader">
      <struct>
        <arrayFixed varName="string-'BM'">
          <character-8/>
          <dim indexTo="2"/>
        </arrayFixed>
        <integer-32 varName="fileSize"/>
        <arrayFixed varName="Reserved - 0">
          <byte-8/>
          <dim indexTo="2"/>
        </arrayFixed>
        <arrayFixed varName="Reserved - 0">
          <byte-8/>
          <dim indexTo="2"/>
        </arrayFixed>
        <integer-32 varName="imageOffset"/>
      </struct>
    </typeDef>
    <typeDef typeName="imageHeader">
      <struct>
        <integer-32 varName="imageHeaderSize"/>
        <integer-32 varName="imageWidth"/>
        <integer-32 varName="imageHeight"/>
        <short-16 varName="numPlanes"/>
        <short-16 varName="bitsPerPixel"/>
        <integer-32 varName="compressionType"/>
        <integer-32 varName="horizontalResolution"/>
        <integer-32 varName="verticalResolution"/>
        <integer-32 varName="numberOfColours"/>
        <arrayFixed varName="colourMap">
          <byte-8/>
          <dim indexTo="2"/>
        </arrayFixed>
        <integer-32 varName="numberOfFirstColours"/>
      </struct>
    </typeDef>
    <typeDef typeName="imageData">
      <arrayStreamed>
        <bit-1/>
        <dimStreamed/>
      </arrayStreamed>
    </typeDef>
  </definitions>
  <file src="picture.bmp">
    <defType typeName="fileHeader"/>
    <defType typeName="imageHeader"/>
    <defType typeName="imageData"/>
  </file>
</dataset>
```

5 Future

In this section we consider a number of ways in which we are planning to or considering taking BinX in the future. These are not in any particular order.

5.1 Binary Access Library

A clear requirement in taking this work forward is a library which can read and write binary files describe by BinX files including data files which have had BinX descriptions embedded in them (e.g. using MIME). For interoperability with Fortran, C, C++ and Java, these libraries should probably be written in C++ with a C API defined to them.

5.2 XML Description Library

This would be a library to support the writing of BinX descriptions with the important property that it would automatically provide the right encoding (bit, byte ordering, block size, integer/floating point representation) for the platform it was writing on, by default.

5.3 GUI for writing/editing BinX files

Built on top of the previous library this would be a GUI to allow users to easily describe their data files.

5.4 JAJA extensions

JAJA (Java Access to Just-about-any Array) is a BinX browser. It was constructed for BinX 0.1 and at time of writing is hopelessly out of date. This idea should be revived for testing and demonstration and really represents a GUI on top of the Binary Access Library.

5.5 Standards process

BinX needs to be taken forward as an international standard through the GGF.

5.6 Test cases

We need to develop test cases based on real datasets from our test users and others to ensure that the description is sufficiently powerful and that the tools can do what we claim.

5.7 Implied XML representation

From a BinX description it would be possible to infer a standard XML representation of a file. Such a representation could be based on an existing standard for Scientific Data representation such as XSIL. The advantage of formalising this representations is that it would allow:

1. XPath XQuery and Xpointer to be used to return data from a binary file encoded with BinX
2. With a tool like Cacoon XSL transformations of BinX files would be possible.

5.8 Transformations

There are three kinds of transformation that we would like to be able to support:

1. **Binary Compatibility Transformations –** these are transformations in which the structure of the data represented remains unchanged, but the underlying binary representation (e.g. byte order or Blocksize) changes. Such transformations can be implied from two BinX files (start and end) and a conversion tool would be easy to construct from the Binary Access Library.
2. **Structural transformations –** these are transformations in which data fields are reordered, deleted, renamed etc. From the Implied XML representation of a file we could use a tool such as Cacoon to carry out such transformations. A BinX generator would be written using the BinX Access Libraray to generate SAX events corresponding to the implied XML representation while parsing a binary file. These events can be passed down the Cacoon pipeline and be transformed using XSLT. A serialiser on the far end can turn the transformed events into a new BinX file.
3. **Numerical transformations –** some level of simple coverstion of data (e.g. between units – see below) could be handled by XSLT and Cacoon. However large scale numerical operations and calculations such as array transpositions etc. are unlikely to be very efficient. Such transformations could be handled by applying the binary access library to a tool such as Data Cutter.

5.9 Units field

It would be useful to have a standard optional field which allowed the expression of units. Better still the units should be described as an NMTOKEN type which would enumerate a standard set (SI units plus common alternatives, say). The advantage of having a standard representation is that the semantics can then be defined and therefore safely inferred from the unit used and so a generic tool for conversion between units becomes possible.

5.10 Date/Time representation

A standard BinX representation for Data and Time would probably be useful. Whilst this is, in general, a can of worms, it should be possible to define a standard set of typeDefs to represent these plus semantics to accompany them. This would at least save users having to do it from scratch every time.

5.11 Classes vs. instances

It has been left a rather open question whether BinX should be used to define classes or instances of a data files. That is does a BinX file describe the set of all possible bmp files or does it describe this particular bmp file “myPicture.bmp”. With simple files, there is no real difference. Simple files means either:

- Files with static sized fields or
- Files which use the XDR conventions for variable sized fields and discriminated unions (essentially that the data is proceeded by and integer representing size or descriminant).

However, consider the bmp case. One of the fields in the bmp header is file size. Unfortunately there is no way to indicate to BinX the this is file size. So it is possible in BinX to

- provide a complete description of a particular bmp file.
- or to provide a partial description of all (perhaps a class of) all possible bmp files. However this partial description will have to leave the size open.

The fact that the data files in BinX are specifically named suggests that we are actually describing instances and not classes of files. On the other hand it would be useful if we could settle on a file description that was the same for all files of that type. How useful it would be depends on the use case scenarios. The next sections considers some extensions that would provide greater flexibility in this area.

5.12 Variables

A useful extension (and one which would be necessary for the subsequent suggestions on parameterisation) is the addition of variables to BinX. A syntax like to one used in XSL would work for example a variable would be set with:

```
  <integerVariable name="arrayLength">32</variable>

and used with:
  <arrayFixed>
    <byte-8/>
    <dim indexTo="$arrayLength"/>
  </arrayFixed>
```

The XSL semantics is that a variable is write-once, read-many that would simplify their use. At this point integers are probably the only type we would need.

5.13 Parameterised length relationships

So we would like to be able to set variables from fields in a binary file, carry out simple calculations on them and use the results as part of the description. There are two ways we would like to set the field, one by discovering the true size of some of the data, the other by taking the value of a data field. For example a file might contain:

```
  <variable-sizeOf name="headerSize">
    <arrayVariable varName="author name">
      <character-8/>
      <dimVariable/>
    </arrayVariable>
    <ieeeFloat varName="parameter1"/>
    <variable-valueOf name="fileSize">
      <integer-32 varName="fileSize"/>
    </variable-valueOf>
  </variable-sizeOf>
  <arrayFixed varName="data">
    <byte-8/>
    <dim indexTo="$fileSize-$headerSize"/>
  </arrayFixed>
```

So in this contrived example, the header is of variable size because the "author name" field is a variable array. The header provides as one of its fields the actual size of the file. We have to calculate the header size – captured in the variable "headerSize" – and extract the file size – captured in the variable "fileSize" and then subtract the two to get the data size.

This sort of flexibility makes describing classes of files much more feasible – an important question is: do we need to do that?

5.14 Parameterised typedefs

A significant limitation in BinX is the inability to supply parameters to the typeDefs. We would like to be able to say for example that a "table" is a one dimensional fixed sized array of structs, but leave the definition of how big it is, and which structs until later. XSL has a parameter passing syntax that we could borrow for this. There is a danger that our type system could become undecidable or, at least NP-complete.

Appendix A – XML Schema documentation

Schema **binx.xsd**

schema location:	C:\Documents and Settings\martin\My Documents\Grid\Binx\src\WP5\Binx-XML\binx.xsd
targetNamespace:	http://schemas.nesc.ac.uk/binx/binx

Elements
dataset
definitions
defType

schema location:	C:\Documents and Settings\martin\My Documents\Grid\Binx\src\WP5\Binx-XML\types.xsd
targetNamespace:	http://schemas.nesc.ac.uk/binx/binx

Elements	Complex types	Simple types	Attr. groups
bit-1	primitiveTypeType	orderType	representationDef
byte-8	typeType		
character-8			
enum-32			
ieeeDouble-64			
ieeeFloat-32			
ieeeQuadruple-128			
integer-32			
longInteger-64			
primitiveType			
short-16			
type			
unicodeCharacter-32			
unsignedInteger-32			
unsignedLongInteger-64			
unsignedShort-16			
void-0			

schema location:	C:\Documents and Settings\martin\My Documents\Grid\Binx\src\WP5\Binx-XML\datatypes.xsd
targetNamespace:	http://schemas.nesc.ac.uk/binx/binx

Elements	Complex types
array	arrayFixedType
arrayFixed	arrayStreamedType
arrayStreamed	arrayType
arrayVariable	arrayVariableType
dataType	dataTypeType
struct	dimStreamedType
union	dimType
	dimVariableType
	structType
	unionType

schema location:	C:\Documents and Settings\martin\My Documents\Grid\Binx\src\WP5\Binx-XML\xinclude.xsd
targetNamespace:	http://www.w3.org/2001/XInclude

Elements
fallback
include

element **dataset**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	definitions file arrayMultiFile
identity constraints	Name / Refer / Selector / Field(s) key — typeDefKey — — bx:definitions/bx:typeDef — @typeName keyref — typeDefRef — typeDefKey — .//defType — @typeName
annotation	documentation A description of an complete dataset possibly spread across mutiple files.

element **dataset/file**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	type
annotation	documentation A description that allows (ultimately) the description and interpretation of a complete file.

element **dataset/arrayMultiFile**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	type dim dimMultiFile
annotation	documentation A description that allows the identification and interpretation of an array that has been decomposed across multiple files. The decomposition must be with repect to regions of the slowest moving index.

element **dataset/arrayMultiFile/dim**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	dimType

element **dataset/arrayMultiFile/dimMultiFile**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	file
annotation	documentation The slowest moving dimentsion is defined here and this is where the decoposition of the array into multiple files is defined.

element **dataset/arrayMultiFile/dimMultiFile/file**

namespace	http://schemas.nesc.ac.uk/binx/binx
annotation	documentation Each file contains a chunck of the array as defined by the indexes specified. No check is performed (by the XML) to ensure that the indexes are contiguous.

element **definitions**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	typeDef xi:include
used by	element dataset
annotation	documentation This is where the user defined types are located. In a conventional BinX file this would be the first part of the dataSet tag. However BinX can use XInclude to include additional definitions from

		external files. To allow these files to be validated against this schema, the definitions tag is made a global element so that in an include file, this is the root.

element **definitions/typeDef**

namespace	http://schemas.nesc.ac.uk/binx/binx	
children	**type**	
annotation	documentation	This is where the user-defined types are defined.

element **defType**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	extension of **dataTypeType**	
annotation	documentation	A user defined type

element **bit-1**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	restriction of **primitiveTypeType**	
annotation	documentation	A single bit - has size 1 bit

element **byte-8**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	restriction of **primitiveTypeType**	
annotation	documentation	A single byte - has size 8 bits range from -128 to 127

element **character-8**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	restriction of **primitiveTypeType**	
annotation	documentation	An eight bit character (char) this is equivalent to an unsigned byte

element **enum-32**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	extension of **primitiveTypeType**	
children	**val**	
used by	element	**unionType/discriminant**
annotation	documentation	An enumerated type. Will be represented in the file as a 32-bit integer string values assigned according to type definition.

element **enum-32/val**

namespace	http://schemas.nesc.ac.uk/binx/binx

element **ieeeDouble-64**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**

annotation	documentation A 64-bit IEEE floating point number

element **ieeeFloat-32**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation A 32-bit IEEE floating point number

element **ieeeQuadruple-128**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation A 128-bit IEEE floating point number

element **integer-32**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
used by	element **unionType/discriminant**
annotation	documentation A 32-bit integer

element **longInteger-64**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation A 64-bit integer

element **primitiveType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	extension of **primitiveTypeType**
annotation	documentation Used to define the substitution group of all primitive types

element **short-16**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation A 16-bit integer

element **type**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**typeType**
used by	elements **dataset/arrayMultiFile** **unionType/case** **dataset/file** **definitions/typeDef** complexTypes **arrayStreamedType** **arrayType** **arrayVariableType** **structType**
annotation	documentation Defines the substitution group of types

element **unicodeCharacter-32**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation A 32 bit unicode character

element **unsignedInteger-32**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
used by	element **unionType/discriminant**
annotation	documentation An unsigned 32-bit integer

element **unsignedLongInteger-64**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation An unsigned 64-bit integer

element **unsignedShort-16**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation An unsigned 16-bit integer

element **void-0**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **primitiveTypeType**
annotation	documentation An empty type 0 bits. Useful in unions for declaring the possible absence of a number.

complexType **primitiveTypeType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	extension of **typeType**
used by	elements **bit-1** **byte-8** **character-8** **enum-32** **ieeeDouble-64** **ieeeFloat-32** **ieeeQuadruple-128** **integer-32** **longInteger-64** **primitiveType** **short-16** **unicodeCharacter-32** **unsignedInteger-32** **unsignedLongInteger-64** **unsignedShort-16** **void-0**
annotation	documentation The superclass of all primitive types should always subclass

complexType **typeType**

namespace	http://schemas.nesc.ac.uk/binx/binx
used by	element **type** complexTypes **dataTypeType** **primitiveTypeType**
annotation	documentation The superclass of all types. This is where the basic attributes shared by all types are defined.

simpleType **orderType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **xsd:string**
used by	attributes **representationDef/@bitOrder** **representationDef/@byteOrder**
facets	enumeration bigEndian enumeration littleEndian
annotation	documentation Big endian or little endian

attributeGroup **representationDef**

namespace	http://schemas.nesc.ac.uk/binx/binx
used by	elements **dataset/arrayMultiFile** **dataset** **dataset/file** complexType **typeType**
annotation	documentation The collection of attributes common to all types.

element **array**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**arrayType**
children	**type** **dim**
annotation	documentation Used to define the array substitution group and fix common properties of arrays.

element **arrayFixed**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**arrayFixedType**
children	**type** **dim**
annotation	documentation An array of fixed size. This can contain multiple dimensions (specified using the dim element).

element **arrayStreamed**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**arrayStreamedType**
children	**type** **dim** **dimStreamed**
annotation	documentation A streamed array. The slowest moving index (specified using the dimStreamed element) cannot be constrained in size as it does not provide a indexTo field.

element **arrayVariable**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**arrayVariableType**
children	**type** **dim** **dimVariable**
annotation	documentation An array of variable size. The total size of this array is determined by a 32-bit integer value which lies in the data file before the array elements start.

element **dataType**

namespace	http://schemas.nesc.ac.uk/binx/binx

type	**dataTypeType**
annotation	documentation This tag is used to define the dataType substitution group it is not intended to be used directly.

element **struct**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**structType**
children	**type**
annotation	documentation This is a struct type which simply represents a sequence of other types. Structs can be assigned names usng the typeDef mechanism.

element **union**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**unionType**
children	**discriminant case**
annotation	documentation This element is used to represent a declarative union. The first 32-bits of the data will be a discriminant. This value will determine the type present (according to the values in the case-type mappings).

complexType **arrayFixedType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	extension of **arrayType**
children	**type dim**
used by	element **arrayFixed**
annotation	documentation Type definition for arrays in which the size of the array is included in the type definition.

complexType **arrayStreamedType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **arrayType**
children	**type dim dimStreamed**
used by	element **arrayStreamed**
annotation	documentation Type definition for streamed array. Streamed arrays have a final dimension (the slowest moving one) in which the indexFrom attribute is removed so that the size of the array is left unspecified.

element **arrayStreamedType/dim**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**dimType**

element **arrayStreamedType/dimStreamed**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**dimStreamedType**

complexType **arrayType**

namespace	http://schemas.nesc.ac.uk/binx/binx

type	extension of **dataTypeType**
children	**type** **dim**
used by	element **array** complexTypes **arrayFixedType** **arrayStreamedType** **arrayVariableType**
annotation	documentation Superclass of all arrays

element **arrayType/dim**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**dimType**

complexType **arrayVariableType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **arrayType**
children	**type** **dim** **dimVariable**
used by	element **arrayVariable**
annotation	documentation Type definition for arrays in which the size determined by a single 32 bit integer located in the data file before the array starts. These arrays can hav only one dimension specified by the dimVariableType tag.

element **arrayVariableType/dim**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**dimType**

element **arrayVariableType/dimVariable**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	**dimVariableType**

complexType **dataTypeType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	extension of **typeType**
used by	elements **dataType** **defType** complexTypes **arrayType** **structType** **unionType**
annotation	documentation The type common to all datatype definitions. The term datatype is used here to refer to composite types composed of multiple primitive types such as arrays, structs and unions.

complexType **dimStreamedType**

namespace	http://schemas.nesc.ac.uk/binx/binx
type	restriction of **dimType**
used by	element **arrayStreamedType/dimStreamed**
annotation	documentation This is intended to provide the slowest moving dimension in a streamed type. The difference here is that the indexTo field is optional so accomodate an unspecified size of array.

complexType **dimType**

namespace	http://schemas.nesc.ac.uk/binx/binx
used by	elements **dataset/arrayMultiFile/dim** **arrayType/dim** **arrayVariableType/dim** **arrayStreamedType/dim** complexTypes **dimStreamedType** **dimVariableType**

annotation	documentation	A description (name, index limits) of an array dimension. indexFrom is the starting index of the array dimension indexTo is the final index both are specified in units of the array type (integer, float, double etc.).

complexType **dimVariableType**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	restriction of **dimType**	
used by	element	**arrayVariableType/dimVariable**
annotation	documentation	The size of a variable array is speecified as an integer written in the first 32 bytes of the structure. Variable sized arrays can have only one dimension and it cannot have and indexTo value.

complexType **structType**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	extension of **dataTypeType**	
children	**type**	
used by	element	**struct**
annotation	documentation	Type definition for the struct type, which represents a sequence of types

complexType **unionType**

namespace	http://schemas.nesc.ac.uk/binx/binx	
type	extension of **dataTypeType**	
children	**discriminant case**	
used by	element	**union**
annotation	documentation	Type definition for the union type, which represents a series of alternative type options chosen by a 32-bit integer discriminant.

element **unionType/discriminant**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	**integer-32 unsignedInteger-32 enum-32**

element **unionType/case**

namespace	http://schemas.nesc.ac.uk/binx/binx
children	**type**

element **xi:fallback**

namespace	http://www.w3.org/2001/XInclude	
children	**xi:include**	
used by	element	**xi:include**

element **xi:include**

namespace	http://www.w3.org/2001/XInclude
children	**xi:fallback**

used by	elements	**definitions** **xi:fallback**