Developments in BinX, the Binary XML description language

Robert Carroll, Davy Virdee and Qingying Wen

Edikt, National e-Science Centre, University of Edinburgh, Edinburgh UK
http://www.edikt.org/binx/

Abstract

e-Science requires mechanisms by which data of different types and sources can be communicated and understood between interested application scientists. Edikt have developed the BinX, a tool to aid this by describing binary data and its layout using an XML meta-data document. This paper will describe the BinX Language and Library, and discuss some current eScience applications that use BinX in the fields of astronomy, particle physics and meteorology.

1. Introduction

One of the most common data storage formats in almost all fields of science is the binary data file, which can reach many terabytes in size. For example, in particle physics the Large Hadron Collider (LHC) at CERN is expected to generate about 12 petabytes of experimental data a year [1]. There is a strong need to share these files among collaborating scientists and to use the data in new ways; through enhanced analysis, visualisation or data mining techniques. Sharing binary data is difficult because no single data format is universal and the format of a binary data file is typically recorded within one or two software programs created specifically to process that data.

The byte-order for numeric values in binary data files differs between files constructed on a little-endian system versus a big-endian system. Compounding the problem, programming languages and their respective compilers, interpreters and libraries (such as FORTRAN, C, C++ and Java), support a different (although overlapping) set of numeric representations. In summary, data created on one type of machine is primarily targeted for processing on that particular type of machine. These issues form a significant obstacle to both data sharing and accessing legacy data. To address these issues, Edikt have developed the BinX Language and the BinX Library to provide mechanisms to aid this data sharing by describing binary data and its layout using an XML meta-data document.

1.1 Application Models

To effectively share data, a solution must provide more than basic cross-platform read and write capabilities. Data sharing services must also support the following operations on a binary dataset:

- Selecting a data subset, selected by some user specified query.
- Combining data from multiple sources.
- Restructuring and reorganising data.
- Efficient data transport among networked platforms.

Brief examples of these operations are provided below to demonstrate the need for a data sharing service to support these operations.

Selecting a Data Subset

The United Kingdom Meteorology Office uses the Unified model [6] to store weather and climate information in a binary file suitable for dense storage and weather model integrations. A typical requirement is to create a subset of the data by selecting parts of that data using geospatial and temporal conditions (e.g. a data subset at a particular time or a known vertical and horizontal slice). This reduces the cost and increases the speed by processing only the data of interest.

Combining Data from Multiple Sources

Often there is a requirement to merge and not just concatenate binary data files. In a set of sensors, each device stores its own data in a separate binary file but with the same data structure. An integrated file can be created by the co-ordinated reading and combining of the distinct parts of the data structure. For example, the elements of an array in data file 1 and data file 2 are merged in the correct sequence for the array in the combined file.

Restructuring and Reorganising Data

A typical requirement is to convert data from old formats to new formats as systems are

upgraded, or to convert data between two standard formats. In astronomy there are two widely used data formats; the XML schema-based VOTable and the binary FITS format. The binary FITS format is useful for data transportation but the XML VOTable format is more useful for applications. Clearly, a tool that can provide straight forward conversion between the two formats would be highly beneficial to Astronomers.

Efficient Data Transport

The most cost effective solution is to share the data especially for very large datasets rather than to copy the data to another site, but this is not always possible. Policies and services may prevent sharing the data or moving applications to analyse the data from one organisation to another. If large binary datasets are copied, the two primary issues are the speed (and cost) of copying the data, and the retention of metadata associated with the binary dataset. A copied binary dataset is useless without the syntactic and semantic metadata required to ensure platform-independent data access. In the case of astronomy, an XML VOTable file might be converted to a binary FITS file for efficient data transfer but care has to be taken not to lose the metadata associated with the XML mark-up tags.

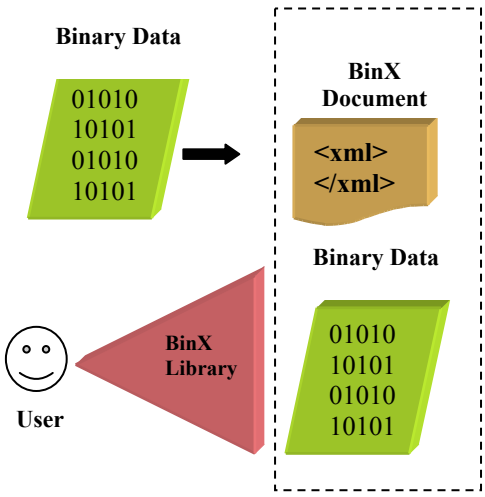

Figure 1: Using BinX. A BinX Document is used to represent *any* binary data, which then allows any user to read the binary data using the BinX Library

BinX

BinX, which consists of the BinX Language and the BinX Library, is a solution to efficient data transport as well as restructuring data, combining data and selecting a data subset. The manipulation of binary data is the fundamental purpose of BinX. This is achieved by representing the structure of a binary file with a XML based BinX Language document and by providing a library of functions which a program can invoke to directly access the data in the binary file (Figure 1).

2. BinX – Binary in XML

The BinX Language is an annotation language which uses the XML mark-up tags to describe the data types and structures in binary data files. It has the ability to describe primitive data types such as characters, bytes, integers (16, 32 and 64 bits), floating point numbers (32 and 64 bits) and strings. It can also describe arrays, sequences (structs) and unions. Another advantage of BinX is the ability to define a user type based on an array, sequence or union. The description of the binary data is stored in an XML file known as a BinX Document. The BinX Document contains metadata about each data element within the data file. Within the BinX Document is a reference to the binary data file.

There are two representations of an annotated binary data file that is supported by the BinX Language. The first representation is the unmodified binary data file with a standalone BinX Document describing the binary data file. The second representation is the metadata and the data values in one XML document known as a DataBinX Document (Figure 2). The usual representation is the separate BinX Document where common or repeated elements can be defined once rather than repeatedly. This leaves the original binary data unchanged in the binary file.

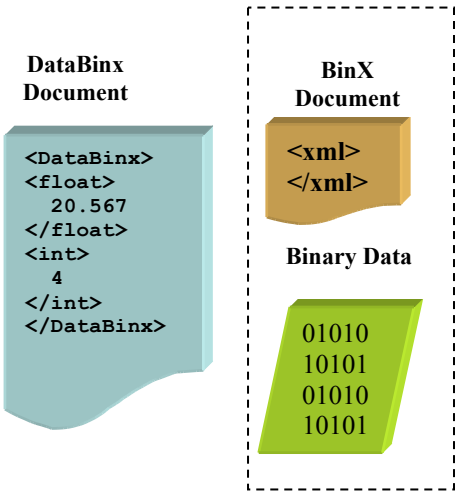

Figure 2: In a DataBinx Document, (left) the binary data is marked-up in XML using the BinX Document (right), which describes the binary data.

2.1 BinX Document

A BinX Document is an XML document which describes a given binary file using the BinX Language. The document consists of a sequence of XML elements which describe the individual binary fields in the binary file. The binary file is specified using the "src" attribute of the dataset element. There are also other XML tags which describe more complex data structures. Figure 3 is an example of the BinX Language, as used in a BinX Document.

```
<binx byteOrder="littleEndian">
  <dataset src="sss.fits">
    <string delim="\n" varName="name"/>
    <arrayFixed varName="Header">
      <byte-8/>
      <dim indexTo="17279"/>
    </arrayFixed>
    <struct varName="Field_data">
      <string size="8" varName="ID"/>
      <double-64 varName="RA" unit="Rad"/>
    </struct>
  </dataset>
</binx>
```

Figure 3: An example BinX Document

2.2 Data Types

BinX supports a wide range of primitive data types, particularly the most commonly used in e-Science applications. Table 1 summarises currently supported BinX data types.

Category	Size	Signed
Byte	8-bits	Signed & Unsigned
Character	8-bits	
Unicode	16 and 32-bits	
Integer	16,32 and 64-bits	Signed & Unsigned
Float	32, 64, 80, 96, and 128 bits	
Void	zero bits	

Table 1: Primitive BinX data types

The platform dependent aspects which affect how the primitive data type is written to a binary file must be described. Two of the critical attributes are byte ordering and block size. The order that bytes are stored for multi-byte data can be most significant byte first (big-endian) or least significant byte first (little-endian). The block size determines how a primitive data type might be padded (additional bytes added that are not used) to maintain a uniform block size over a set of data units.

2.3 String Data Type

There are three types of string that can be specified. In a fixed length string the number of characters is defined in the BinX Document. In a variable length string the number of characters is determined by a binary value within the data file immediately preceding the string. In a delimited string the number of characters is determined by a specified ASCII character that terminates the string.

2.4 Data Structures

The BinX Language also supports a set of complex data structures. There are three basic structural elements which combine data elements in different ways. These elements are multi-dimensional arrays, ordered sequences of data fields (structs) and unions (which allow dynamic specification of data types when reading a binary data file). Table 2 summarises currently supported BinX data structures. Three types of arrays are supported. In a fixed size array with one or more dimensions, the number of elements in each dimension is defined in the BinX Document. In a variable size array with one or more dimensions, the number of elements in each dimension is defined in the BinX Document except for the outer dimension. This dimension is determined by a binary value in the data file immediately preceding the sequence of element instances in the array. A streaming array is a one-dimensional array with

an unknown number of entries. The number of entries is discovered by reading array values from the binary file until reaching the end of the file.

Category	Type	Feature
Array	Fixed size	Multi-dimensional
	Variable size	Multi-dim with one variable dimension
	Streaming	One-dim with dynamic # of entries
Struct		Ordered fields
Union		One of a list of types

Table 2 Primitive BinX data structures

A BinX structure defines an ordered sequence of other data types. Each occurrence of the structure must include all fields in the specified order. A BinX union defines a list of selectable alternative data types. Each instance of a union type in the binary file can be of any one of the data types from the selection list. A discriminant for the union is a field, present in each instance of the union in the binary file, which identifies the data type that has been selected. In a BinX union the discriminant must be a single field of integer type at the start of each union element.

2.5 Defining Data Types

The BinX Language offers a facility to define user-defined data types which can then be referenced in the rest of the BinX Document. This is achieved using the "defineType" mechanism, which enables users to define their own data structures as macros that can be reused later in the document. This feature is important as the user can assign semantically meaningful names to types and individual data elements. A user defined type is referenced in the document using the "useType" element. Figure 4 shows a BinX Document with a definition section.

The user-defined types are contained within the XML tag <definitions>, followed by the rest of the BinX Document. The user-defined data types must be declared first before they can be used. In the definitions section, a new data type with type name "complexType" has been declared. It defines a struct containing two floats, one with type name "real" and the other with type name "imaginary". The binary file is specified in the dataset section which contains a float and an integer, followed by a fixed array with two dimensions. Each array entry is of the new complex number type.

```
<binx byteOrder="littleEndian">
  <definitions>
    <defineType typeName="complexType">
      <struct>
        <float-32 varName="real"/>
        <float-32 varName="imaginary"/>
      </struct>
    </defineType>
  </definitions>
  <dataset src="testFile.bin">
    <float-32 varName="StdDeviation"/>
    <integer-32 varName="IterCount"/>
    <arrayFixed>
      <useType typeName="complexType"/>
      <dim indexTo="99" name="x">
        <dim indexTo="4" name="y"/>
      </dim>
    </arrayFixed>
  </dataset>
</binx>
```

Figure 4: An example BinX Document

2.6 DataBinX

A standalone BinX Document provides a highly efficient annotated representation for a binary data file. It requires minimal additional storage space and network bandwidth to transport the compact BinX Document. This representation is well suited for applications that manipulate the entire binary file. When performing queries over a subset of the binary file, it can be useful to build an XML representation of the binary file. This file contains the data values from the binary file and the XML elements from the BinX Document. Each binary file value is converted to a character representation and annotated with the XML tags from the BinX Document.

This new file is a DataBinX Document and is useful when performing queries (based on XML) over a binary file. Figure 5 shows the DataBinX Document for a fixed sized array containing two columns and two rows of the user-defined type "complexType" as used in the last example.

```
<databinx>
  <dataset>
    <arrayFixed>
      <dim name="x">
        <dim name="y">
          <struct>
            <float-32>0.223e5</float-32>
            <float-32>0.221</float-32>
          </struct>
        </dim>
        <dim name="y">
          <struct>
            <float-32>0.446e6</float-32>
            <float-32>0.332</float-32>
          </struct>
        </dim>
      </dim>
      <dim name="x">
        <dim name="y">
          <struct>
            <float-32>0.223e7</float-32>
            <float-32>0.443</float-32>
          </struct>
        </dim>
        <dim name="y">
          <struct>
            <float-32>0.446e8</float-32>
            <float-32>0.554</float-32>
          </struct>
        </dim>
      </dim>
    </arrayFixed>
  </dataset>
</databinx>
```

Figure 5: An example DataBinX Document

3. The BinX Library

The BinX Library provides an Application Program Interface (API) which can be called from an application program to access the data in a binary data file. The library uses the BinX Document to enable the application program to access the data in the binary file without having to know its low-level layout. The BinX Library API allows the application developer to retrieve binary file data in terms of the BinX language. This includes facilities for navigating sequentially through complex data structures such as arrays in the binary data file. A broad range of generic utilities based on the API have been developed that solve common data transformation problems.

The BinX Library provides the foundation for tools that can read a very wide range of file formats. Such tools could provide the functionality to convert data between different formats, to extract data subsets (e.g. slices or diagonals of an array), or to query the XML file DataBinX. The BinX Library also includes automatic conversion between little-endian and big-endian platforms so making cross platform data transportation simple and the API also has the functionality to programmatically create the BinX Document.

3.1 Generic Utilities

In addition to the BinX Library there are some generic utilities. These tools are based on the core functionality accessible through the API of the BinX Library. The available utilities include:

- A DataBinX program which generates a DataBinX Document from a BinX Document and a binary file.
- A DataBinX parser which generates a binary data file and a BinX Document from a DataBinX Document.
- A program GenSchemaBinX which generates a BinX Document from the data structures in memory.

3.2 Application Programming

The API can be used to develop applications that require data extraction and data conversion of arbitrary binary data files. The API is written in C++ and applications are compiled and link with the BinX Library using a C++ compiler. The API defines C++ classes associated with the BinX objects, such as the BinX document, and the primitive and composite data types. For example, there is a class BxInteger32 that corresponds to the BinX Language element <integer-32> and a class BxFloat32 that corresponds to the BinX Language element <float-32> and so on. The API has methods to parse the BinX Document; read, write, and convert binary data and output a BinX or a DataBinX Document. With these and other functions in the API of the BinX Library, BinX applications can be developed with minimal effort.

4. Case Studies

The BinX Library has been mainly used as a data transformation tool in the fields of Astronomy, Particle Physics, Bioinformatics, and within Grid Data Services (GDS). It is used as a means of data extraction, conversion, and transportation. In Astronomy, it is used to convert data between VOTable and FITS formats; and in particle physics, it is used to

merge datasets from distributed remote sites for central data processing. In addition, it is used to pack datasets for transport over the Internet and unpack datasets as SAX events for data mining applications. We describe these use cases in the following sections.

4.1 Data Conversion in Astronomy

BinX has been used to solve a data conversion problem in the Astrogrid [2] project. Two of the main data formats used by Astrogrid applications are a self-describing binary table format called FITS (Flexible Image Transport System) [3] and an XML annotated table representation called VOTable (Virtual Observatory Table) [4].

A FITS file contains a primary header and a number of optional extensions. The primary header may contain a primary data array. Each extension has its own header and binary data array. Headers are composed of blocks of text where the parameters are saved. Each line in the block contains a maximum of 80-characters. The parameters can be variable-value pairs or simple a comment. The binary extension can be an image or a data array. For a data array, metadata describing the size of the array are specified as predefined variable parameters in the extension header.

The VOTable format is an XML based data annotation language to describe the syntactic and the semantic aspects of astronomical datasets for storage and interchange. The XML based format facilitates transformations using XSLT engines.

Applications assume a specific format for the input file. If an application is designed for the FITS format and the data file is only available in the VOTable format then there is a requirement to convert the VOTable file into a FITS file. Other applications may assume the input file is in the VOTable format and so FITS files will need to be converted to the VOTable format.

When converting between these two formats, we must retain as much of the meta-data information as possible. Several rules are required to define the mappings between the two formats. In both conversion processes, DataBinx is used as a temporary file to store intermediary data for transformation using XSLT [5]. By using an XSLT script it is easy to customise a partial or special transformation as required. Some BinX utilities are used to generate or parse the DataBinX Document, and two additional programs (using XSLT) are responsible for parsing and generating the converted files.

The Astrogrid project is also interested in transporting large VOTable documents over the Internet. VOTable documents which contain annotated binary datasets are potentially too large to transport over the Internet. Therefore it would be useful to reduce the size of the document for cost-efficient transportation. One solution is to use the BinX library and an XSLT script to convert the VOTable document into a DataBinX Document which can then be parsed into a binary data file and a BinX Document. These files can be further compressed using zip and then transported over the Internet. At the destination, the reverse steps are performed to transform the compact package back to the original VOTable document.

4.2 Data slice for Meteorology

The data that is used in the Met Office Unified model [6] is stored in a binary file suitable for dense storage and model integrations. The structure of the binary file which has a header and a data field is known as pp format. The pp format data file is not very suitable for exploring the data through data analysis and visualisation tools. The BinX library could be used to describe the pp format data files in such a way as to allow direct analysis and exploration of the pp files without converting them to other formats.

The typical requirements are to create a subset of the pp data by selecting parts of that data using geospatial and temporal conditions (e.g. a data subset at a particular time or a vertical/horizontal slice). The geospatial and temporal descriptors are stored as metadata elements in the header of the pp format with the data field following the header containing the data for that specified location and time.

A BinX Document can be created to describe a pp format data file. Since the query only requires the data values in the header, a DataBinX Document only needs to be generated for the header of each block. From an XPath query over the DataBinX Document a subset of the data can be created. This data slice can then be analysed separately.

4.3 Particle Physics Matrix Transformation

The QCDGrid project [7] develops applications that use data distributed across several sites. A typical single site dataset consists of a four-dimensional array, where each array entry is a two-by-three array of complex numbers. This four-dimensional array stores data for a single point in time. The total input dataset for an application is constructed from a

sequence of four-dimensional arrays, each storing data for a different time slice and for different sites. Sites are somewhat independent and may store array values based on different orderings of array dimensions.

Various types of data transformations are required to construct a total input dataset for a QCDGrid application, for example matrix transposition (converting row-major order to column-major order) or matrix coalescence (appending a sequence of arrays).

Figure 6 shows a BinX Document for a typical single site, particle physics dataset. The definitions section defines two abstract data types; the structure for a double precision complex number, and a two-by-three array of complex numbers. The dataset section describes a dataset containing a four-dimensional array, where each array entry is a two-by-three array of complex numbers.

```
<binx byteOrder="bigEndian">
  <definitions>
    <defineType typeName="complexType">
      <struct>
        <double-64 varName="real"/>
        <double-64 varName="imaginary"/>
      </struct>
    </defineType>
    <defineType typeName="matrix2x3">
      <arrayFixed>
        <useType typeName="complexType"/>
        <dim name="row" indexTo="1">
          <dim name="column" indexTo="2"/>
        </dim>
      </arrayFixed>
    </defineType>
  </definitions>
  <dataset src="fileTime">
    <arrayFixed varName="Timeslice">
      <useType typeName="matrix2x3"/>
      <dim name="mu" indexTo="3">
        <dim name="x" indexTo="15">
          <dim name="y" indexTo="15">
            <dim name="z" indexTo="15"/>
          </dim>
        </dim>
      </dim>
    </arrayFixed>
  </dataset>
</binx>
```

Figure 6: BinX Document for a pp dataset

To use the data an application must append the input files in time slice order and each four-dimensional array must be transposed. The application uses the BinX library to read the input datasets in time slice order, transpose the four-dimensional matrix from the ordering [mu, x, y, z] to the ordering [z, y, x, mu], and write the transposed, coalesced matrix to a single binary file.

5. Tooling Support and Future Work

5.1 Graphical Tools

Working with application scientists and our other users has led us to begin development on two generic tools. The first tool, *BinX Query Tool*, is a graphical user interface (GUI) that will allow users to create a BinX Document based on the binary data file. The users can then use the GUI to perform XPath type queries on the binary file. This tool removes the user's need to understand XML, and empowers the user by allowing them to subset and mine their data. The second tool, *BinX Converter*, builds on our experiences working with the astronomy community. BinX Converter will allow users to convert from one user-specified data file format to another user-specified data file format. Users will specify the file format by creating a BinX Document. This tool will also incorporate a GUI to allow generation of BinX Documents. BinX Query Tool and BinX Converter are intended for release by the end of 2004.

5.2 Web and Grid Service Integration

We are also working on the integration of BinX with Edikt's data access and integration middleware, Eldas [8]. This integration will allow users to query binary data files over the Internet using both Web Services and Grid Services. Edikt's work in this area will in the future allow users to join, manipulate and merge binary data with relational databases.

5.3 Extensions to the BinX Library

The next data type that is planned to be added is text (ASCII) which will be based on an array of the string data type. Further work will also be done on looking at improving the performance at handling really large datasets.

6. Conclusion

One of the powerful features of using the BinX Library is that binary data can be read from various sources and presented to an application in data types that are native to the host machine and the programming environment. All platform specific conversions have been performed automatically by the BinX Library.

BinX can also be used to access the large amount of legacy scientific data stored in binary data files and if required update the formats and help with queries over the data.

Downloading BinX

BinX is free to download from the Edikt Web site [9]. It is available in a pre-compiled library format for Solaris and Linux. The source code is available for download for academic users. Documentation and the BinX Library API are also available from the Edikt Website.

References

[1] LHC Computing Grid Project http://lcg.web.cern.ch/LCG/

[2] Astrogrid project webpage http://www.astrogrid.org/

[3] FITS webpage http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html

[4] VOTable webpage http://www.us-vo.org/VOTable/

[5] XSL – The Extensible Stylesheet Language Family http://www.w3.org/Style/XSL/

[6] British Atmospheric Data Centre, Met. Office Unified Model http://badc.nerc.ac.uk/data/um/

[7] QCDGrid project webpage http://www.gridpp.ac.uk/qcdgrid/

[8] Eldas project webpage http://www.edikt.org/eldas

[9] BinX project webpage http://www.edikt.org/binx