

BinX – A tool for retrieving, searching, and transforming structured binary files

Rob Baxter, Robert Carroll, Denise J. Ecklund, Bob Gibbins, Davy Virdee, Qingying Wen
edikt, National e-Science Centre University of Edinburgh, Edinburgh UK
www.edikt.org info@edikt.org

Abstract

Numerous large files of binary data store a wealth of e-Science information. Existing applications, in areas such as astronomy and particle physics, process the bulk of these files. To further our scientific knowledge, there is a strong need to share binary files among collaborating scientists and to use the data in new ways, such as data mining to discover new relationships and to identify data subsets of interest. To share large binary datasets among collaborating scientists, a common description language is required to describe the structure and significance of these data files. BinX is a tool to address these needs.

1 Introduction

Numerous large files of binary data store a wealth of e-Science information. Large datasets are output from e-Science analysis applications, or gathered and recorded by sensing devices. Particle physics and astronomy are example domains for such applications. Within and among these scientific communities, collaborating scientists have a strong need to share raw data and derived data. Through data sharing, these communities have the opportunity to make new observations, derive new models of their science and define new problem solving approaches. Sharing binary data is difficult because:

- no single data format (or representation) 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 IEEE endeavoured to define a single binary representation for numeric values. Although successful in defining numeric precision, byte-order for numeric values still differs between systems constructed with little-endian versus big-endian representations. 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. Basically data created on one machine or by one sensor is primarily targeted for processing on one particular type of machine. Clearly computing platforms differ from lab to lab. Hence, these issues form a significant obstacle to data sharing within and among our scientific communities.

1.1 Application Models

Being resourceful, scientists have developed singular solutions to facilitate sharing particular binary data files among a selected set of scientists. Each solution is specially crafted and typically only a rudimentary level of sharing is achieved. The effort required to support basic reading and reformatting of binary data left little human energy for building more sophisticated ways of sharing binary data.

Effective data sharing facilities 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
- Combining data from multiple sources
- Restructuring and reorganising data
- Building index and catalogue structures for efficient access to related data
- Efficient data transport among networked platforms

We motivate the need for each of these services with a brief example.

Selecting a Data Subset

An optical telescope records emissions from a specific spatial quadrant. Scientists sharing this recorded data may be interested in different or distinct objects residing in sub-regions of the quadrant. Independent queries can select data for specified sub-regions and deliver the data subset to a target platform for further analysis. Reading and transferring only the data of interest greatly reduces the cost to retrieve, transfer and store the data.

Combining Data from Multiple Sources

A set of mobile hydro-phones is deployed to gather seismic data. The recording devices (of

various ages from various manufacturers) are organized side-by-side to gather data from overlapping target areas. Each device produces a linear dataset for the target area it moves over and stores its data in a separate binary file. An enhanced application to predict geological features has been developed and scientists would like to investigate the region recorded in these datasets. Interleaved, coordinated reading and combining of the distinct data formats created by each device type allows creation of an integrated database.

Restructuring and Reorganising Data

Applications developed using FORTRAN tools and libraries read and write array data in column-major order. Consider a two-dimensional array having 3 columns and 100 rows. To initialise this array with values stored in a binary file, the first 100 values read from the file are used to fill the first column, the next 100 values fill the second column, and the next 100 values fill the third column. In contrast, applications developed using C language tools and libraries read and write array data in row-major order. To initialise the same array from a binary file, a C language program reads the first 3 data values to fill the first row, the next 3 data values to fill the second row, etc. If an analysis application written in C must read data created by a FORTRAN application, the array values must be transparently transposed as they are read from the data file.

Index and Catalogue Structures

Consider again the example of seismic data gathering from overlapping target areas. Forming an integrated dataset is a two step process: (1) convert data formats and representations from the varied devices, and (2) combine the separate datasets into a single dataset. Step 2 requires information about the spatial arrangement and target overlap of the hydrophones. To combine the converted data, a separate hierarchy of catalogue (metadata) information is essential. At the lowest level, metadata describes basic syntactic features of the files (such as byte order, array orientation, record structure, etc). The next level of catalogue structure includes domain-specific syntactic metadata (such as the relative spatial position of a sensor). Additional catalogue layers can hold domain-specific semantic metadata (such as provenance information on how and when the dataset was created and use restrictions on how the dataset might be applied). Such catalogue information is required to support proper dataset integration.

Efficient Data Transport

When sharing very large datasets it is most cost-effective to share a single copy of the dataset, moving analysis applications to the data. This is not always possible. Policies and services may prevent moving applications from one organization or computing platform to another. If large binary datasets are copied, the two primary issues are: (1) the speed (and cost) of copying, and (2) 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.

Extensive sharing of binary datasets enhances our ability to “do science”. Tools are required to provide the services we have described. BinX is such a tool.

1.2 Organisation of this Paper

In Section 2, we present an overview of previous work addressing aspects of the problem we defined in section one. In Section 3, we introduce the BinX language for describing the structure of binary files. Example file descriptions are provided. Section 4 presents the BinX toolset, composed of a library and a set of generic utilities for manipulating binary datasets. Examples of the BinX Library APIs are also presented. In Section 5, three case studies using the BinX solution for sharing binary datasets are presented. This paper concludes with a summary and future plans for BinX.

2 Related Work

Sharing and exchanging data among varied platforms and programming environments has long been recognized as a significant problem. Several projects have attacked various aspects of this problem, using specification languages to describe application data.

The Binary Format Description (BFD) language [4] was developed as part of the US Department of Energy-funded Scientific Annotation Middleware (SAM) project at the Pacific Northwest National Laboratory. BFD is based on the eXtensible Scientific Interchange Language (XSIL) and is very similar in concept to BinX. A reference implementation has been developed.

The Hierarchical Data Format (HDF) project [8] is run by the National Center for Supercomputing Applications (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, visualising and converting

scientific data. HDF defines a binary data format. All other data formats must be converted to HDF format. A library of conversion functions is provided. In addition, HDF provides software to convert (most) HDF files to a standard XML representation.

The External Data Representation Standard (XDR) is an IETF standard defined in RFC1832 [11]. It is a standard for the description and encoding of data in binary files. It differs from BinX in that XDR is prescriptive of the data format whereas BinX is descriptive and the BinX language is XML based.

The Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) specification [10]. RDF defines an XML-based language to describe metadata for web resources. The goal is to enable the processing of web data from differing platforms. Hence RDF and BinX share similar goals and take similar approaches, but the RDF focus is web resources not binary datasets for e-Science.

3 BinX – Binary in XML

BinX defines an annotation language for representing meta-data for binary data streams, and specifies a data model to describe the data types and structures contained in binary data files.

As an annotation language, it has the power to describe primitive data types such as character, byte, integer and floating point, and to represent data structures of sequences, arrays and unions.

BinX specifies a data model where a BinX document contains a virtual dataset and annotations describing the data elements in the dataset. The data values corresponding to the virtual dataset reside in a binary data file or several such files and are included by reference from the BinX document.

BinX supports three representations of an annotated binary data stream: (1) The common BinX representation, consisting of an unmodified binary data file (called the BinX dataset) with a standalone BinX document describing the binary file; (2) A DataBinX document (see Section 3.4), which contains both the meta-data and the data values in one XML document; and (3) A MIME data stream containing an unmodified binary data file with a BinX document describing the binary file.

3.1 Data Types

BinX supports a broad range of primitive data types, particularly those that are important for e-Science applications. Table 1 summarises currently supported BinX data types.

The underlying physical representation of each primitive data type must be described. These are

platform or environment-dependent aspects affecting how data were represented when written to a binary file. Two critical attributes are: byte ordering and block size. Byte ordering can be big-endian or little-endian. The block size of a primitive type describes how an application or input/output library pads data values to maintain a uniform block size over a set of data units.

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

Table 1. Primitive BinX data types.

3.2 Data Structures

The BinX language supports an extensible set of data structures. The basic data structures include multi-dimensional arrays, ordered sequences of data fields, and unions (which allow dynamic specification of data types when reading a binary data file). Table 2 contains a summary of the basic BinX data structures. Three types of arrays are supported. A fixed size array consists of one or more dimensions. The extent of each dimension is defined specifically in the BinX document. A variable size array consists of one or more dimensions. The extent of all but one dimension is defined in the BinX document. The size of the variable length extent is resolved by reading an extent value from the binary data file. A streaming array is a one dimensional array containing 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 types.

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 data types. Each occurrence of the union includes

exactly one of the data types from the selection list. The selected type can vary with each occurrence.

3.3 Data References

The common BinX representation consists of a standalone BinX document describing and referencing data in separate binary files. The BinX data model defines two ways of referencing a binary data values from the BinX document:

- 1) Local file system pathname – The pathname is evaluated to locate the BinX dataset. All default annotations in the BinX document pertain to the BinX dataset. (This type of reference is shown in Figure 1 by the *src* attribute of the dataset element.)
- 2) XPath/XLink expression – The expression is evaluated to locate the BinX dataset elements. The data elements may be in the same document or another one.

Another data reference model in BinX is the “defineType” mechanism, which enables users to define their own data structures as macros that can be reused (referenced) repeatedly. This feature is important as the user can assign semantically meaningful names to types and individual data elements. In section four, we discuss how these names can be used to select meaningful (named) subsets of a binary file. Figure 1 shows a BinX document exemplifying the BinX language features we have discussed in this section.

```
<binx byteOrder="bigEndian"
  blockSize="32">
  <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 1. An example BinX description.

Figure 1 shows a simple BinX description. The root tag is <binx>. User-defined type definitions are contained within the <definitions> tag, followed by one or more file descriptions contained within the <dataset> tag. In the

definitions section, a new type “complexType” has been declared. It defines a struct containing two floats, one called “real” and one called “imaginary”. One binary file is specified in the dataset section. It is located in /testFile.bin. The file contains a float (StdDeviation) followed by an integer (IterCount) followed by a fixed sized two dimensional array. Each array entry is of the new complex number type.

3.4 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 maintain the compact BinX document. This representation is well suited for applications that manipulate the entire binary file. To manipulate a subset of the binary file, it can be advantageous to build a **DataBinX** representation of the binary file. A DataBinX file contains data values from the binary file and elements from the BinX document. Each binary file value is converted to character representation and annotated with all of the BinX document elements required to fully describe the original binary representation of the data value. The DataBinX representation is useful when performing queries (based on annotations) over a binary file. Figure 2 shows the DataBinX file for a fixed sized array containing two columns and two rows of the user-defined type “complexType”.

```
<databinx>
  <dataset>
    <arrayFixed>
      <dim name="x">
        <dim name="y">
          <struct>
            <float-32>0.334e5</float-32>
            <float-32>0.112</float-32>
          </struct>
        </dim>
      <dim name="y"
        <struct>
          <float-32>0.445e6</float-32>
          <float-32>0.223</float-32>
        </struct>
      </dim>
    </dim>
  <dim name="x">
    <dim name="y"
      <struct>
        <float-32>0.556e7</float-32>
        <float-32>0.334</float-32>
      </struct>
    </dim>
  <dim name="y"
    <struct>
      <float-32>0.667e8</float-32>
      <float-32>0.445</float-32>
    </struct>
  </dim>
</arrayFixed>
</dataset>
```

</databinx>

Figure 2. An example DataBinX file.

The DataBinX representation is a verbose, but appropriate representation for executing queries expressed in XPath or XQuery to select a subset of the binary file data. It is an intermediate representation that facilitates fine-grained searching and restructuring of a binary data file. It is not intended to be a long-term, persistent representation scheme.

4 The BinX Library

The BinX Library is a middleware package supporting the BinX model by providing access to large binary data files through a BinX document. As a software product, the BinX Library provides three levels of service:

- 1) Implements a broad range of generic utilities that solve common data transformation problems,
- 2) Defines and exports a high-level API for easy application development by scientists who need to access binary datasets, and
- 3) Defines and exports a flexible mid-range API supporting the development of potentially complex data translation and transformation tools by software developers, such as the edikt team.

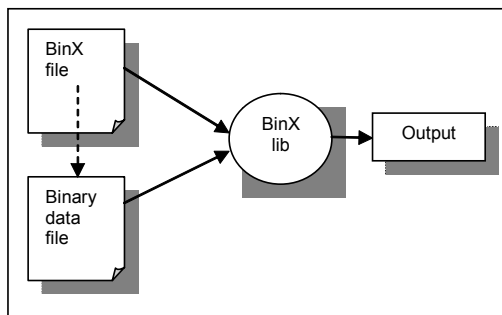


Figure 3. The BinX Library using a BinX document to process a binary file.

Figure 3 illustrates how the BinX file is currently used in practice. The BinX file describes the structure and format of a binary data file and may contain a link to that file. The BinX Library provides the foundation for tools that can read a very wide range of file formats. Such tools could be presented as Web services and provide functionality to convert between formats, to extract data subsets (e.g. slices or diagonals of an array), or to browse the file. DataBinX, for example, is a tool that generates an XML version of a binary data file based on the BinX description.

In many applications we anticipate that BinX descriptions will be included as part of application specific metadata, so that tools based on the BinX Library can be used to efficiently process those binary files.

4.1 The Core Functionality

The BinX Library provides a core C++ API to read data from and to write data to binary files based on the BinX description. The library includes the following core functions.

- Access to data elements in binary files based on the BinX description:
 - The data is presented in native language types
 - Access is efficient – only those things requested are read from disk, data can be read in chunks and copying is minimized.
- Automatic conversion of data to native form:
 - Swapping of byte ordering
 - Removing or adding padding
- Annotate a binary dataset:
 - Generate a BinX document based on memory-resident data structures
 - Write the data values into binary files
 - Write the associated BinX document

4.2 Generic Utilities

In addition to the library, BinX provides the following utilities which can be used as part of an end-user application. These tools are based on the core functionality accessible through the BinX Library APIs. Available utilities include:

- DataBinX generator – explicitly generates the XML view of the binary data.
- DataBinX parser – generates a binary data file and a BinX document from a DataBinX file.
- SchemaBinX creator – aids in creating a BinX document file.
- Data extractor – extracts a sub- or super-dataset from one or more BinX-annotated datasets, using a selection specification and an output specification.
- Binary file indexer – builds an index of byte offsets to identified data items in a binary file.

4.3 Application Programming

The API can be used to develop domain-specific applications that require data extraction and data conversion of arbitrary binary data files. The API defines C++ classes associated with 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>. The

BinX Library implements methods to parse the BinX document, read, write, and convert binary data, and output a new BinX document or a DataBinX file. Given the rich functionality provided by the BinX Library, BinX applications can be developed with minimal effort.

4.4 Current Status

BinX is currently being developed by edikt (<http://www.edikt.org/>), based at the National e-Science Centre at the University of Edinburgh. The first release of a C++ library which implements much of the above functionality is now available from <http://www.edikt.org/binx>.

5 Case Studies

The BinX Library has been used mainly as a data transformation tool in the field 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.

5.1 Astronomical Data Conversion

BinX has been applied to solve data convergence problems in the Astrogrid [3] project. Astrogrid applications use a few complex, table-based data formats which include a self-describing binary table format called FITS (Flexible Image Transport System) [7] and an XML annotated table representation called VOTable (Virtual Observatory Table) [12].

A FITS file contains a primary header and a number of optional extensions. The primary header may contain a dataset in binary format determined by a designator in the header. Each extension has its own header and an inline binary dataset. Headers are composed of 80-character blocks where parameters are saved. The parameters can be variable-value pairs or simple commentary. A binary dataset can be an image or tabular data. For tabular datasets, metadata describing table size, etc. are given as predefined variable parameters in the extension header.

A VOTable is an XML-based data annotation language to describe syntactic and semantic aspects of astronomical datasets for storage and interchange. The VOTable model and the FITS model overlap in terms of expressive power, but each also presents some unique model aspects.

Applications assume a specific input format. If an application assumes FITS file input and the dataset is available only in VOTable format, then there is a requirement to convert a FITS file into a VOTable document. Other applications assume VOTable input and may require the reverse conversion.

When converting between these two formats, we must retain as much meta information as possible. Several rules are required to define mappings between the two formats. In either conversion process, DataBinX is used as an interlingua to store intermediary data for transformation through XSLT[15]. By using XSLT scripting it is easy to customise a partial or special transformation as required. BinX utilities are used to generate or parse a 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 containing annotated binary datasets are potentially too large to transport over the Internet. It is therefore necessary to reduce the document's size for cost-efficient transportation. Using the BinX Library and XSLT, the VOTable document is converted to a DataBinX representation and then to the common BinX representation. The common BinX representation is 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.

5.2 Particle Physics Matrix Transforms

The QCDGrid project [9] 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 each residing at 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, including:

- Matrix transposition – such as, converting row-major order to column-major order.
- Matrix expansion - such as, recovering a compressed three-by-three unitary matrix from a two-by-three unitary matrix, based on

an expressed mathematical property of the expanded matrix.

- Matrix coalescence – such as, appending a sequence of arrays, each representing a distinct time slice.

Figure 4 shows a BinX document for a typical single site, particle physics dataset. The definitions section defines two abstract data types: (1) the structure for a double precision complex number, and (2) 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="complexDouble">
    <struct>
      <double-64 varName="Real"/>
      <double-64 varName="Imaginary"/>
    </struct> </defineType>
  <defineType typeName="matrix2x3">
    <arrayFixed>
      <useType
        typeName="complexDouble"/>
      <dim name="row" indexTo="1">
      <dim name="column" indexTo="2" />
      </dim>
    </arrayFixed> </defineType>
</definitions>
<dataset src="PPfileTime">
  <arrayFixed
    varName="gaugeConfigTimeslice">
    <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> </dim>
  </arrayFixed>
</dataset>
</binx>
```

Figure 4. BinX document for a PP dataset.

Datasets conforming to this schema and corresponding to distinct time slices are stored in separate files. An application wishes to use these datasets as input. To meet the input requirements of the target application, the disparate input files must be appended in time slice order, and each four-dimensional array must be transposed. The application code fragment in figure 5 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.

The BinX Library performs complex operations in a single method invocation. For example, the method *getDataset* opens and processes the BinX document, and opens and prepares the associated binary file for further processing. The BinX method *getArray* extracts

the four-dimensional array from the binary data file, based on the dataset structure defined in the BinX document. The BinX method *get* extracts a single array entry (which in this example is a two-by-three array of complex numbers). The BinX method *toStreamBinary* writes the binary representation of the extracted array entry to an output stream. The output stream can be connected to any entity supporting the stream protocol, such as a file system, a pipe, or a streaming message service.

```
for (int t=0; t<NumOfTslices; t++){
  BxBinxFileReader * pReader =
    new BxBinxFileReader(files[t]);
  BxDataset * pData1 =
    pReader->getDataset();
  BxArrayFixed * pArray =
    pData1->getArray(0);
  for (int z=0; z<16; z++) {
    for (int y=0; y<16; y++) {
      for (int x=0; x<16; x++) {
        for (int mu=0; mu<4; mu++){
          BxDataObject * pData =
            pArray->get(mu,x,y,z);
          if (pData != NULL){
            pData->toStreamBinary(fo);
            delete pData;
          } // if
        } // for (mu
      } // for (x
    } // for (y
  } // for (z
  delete pReader;
} // (for t
```

Figure 5. Transpose and coalesce matrices.

5.3 BinX and GridFTP

BinX has much to contribute to the grid. The GridFTP specification [1], developed in the Global Grid Forum (GGF), defines extensions to FTP services within a grid environment. High performance transportation of binary files is a critical service in the grid. An experiment is underway to incorporate the BinX Library and utilities into an implementation of GridFTP. GridFTP implements the data transport services and BinX manages data schema, provides local data access, and performs platform-dependent data transformations.

BinX also works above the network layer protocols of GridFTP to pack XML-annotated data using the common BinX representation. This is the same technique used to pack and unpack large VOTable documents for transport over the Internet. When unpacking data at the destination, SAX events can be used to notify GridFTP of each data element.

6 Conclusions and Future Work

The ability to read binary data that was written by a sensor or by an application running on

another system is critical for our applications. Using the BinX description, the BinX Library can read binary data and present it to an application in data types that are native to the host machine and the programming environment in which the application is executing. The application can immediately perform calculations using these values. All platform-specific conversions have been performed by BinX.

BinX is descriptive rather than prescriptive. This means that it can be used to access the large amount of legacy scientific data stored in binary data files. It is hoped that the BinX Library will be used to provide Grid data services for the binary data files on various platforms and file systems.

In the future, BinX will play a more major role in the grid. The Data Format Definition Language (DFDL) working group in GGF [6], grew out of BinX. DFDL is an XML-based descriptive language to delineate the structure and semantic content of all file types [14]. It embraces rich semantic descriptions based on standalone ontologies. BinX is viewed as a subset of DFDL, restricted to binary files and supporting limited semantics. We hope to incrementally extend the BinX Library, creating early prototypes of the DFDL specification [13].

The Data Access and Integration (DAIS) Working Group in GGF has specified protocols and interfaces for a Grid Data Service (GDS). A GDS assists with access and integration of data from separate data sources via the grid. The DAIS specification [2] is broad and flexible, allowing the construction of GDSs that provide access to relational databases, XML databases, and file systems composed of directories, text files and binary data files.

BinX can be packaged as a GDS to access binary data files. A BinX-based GDS could accept and execute XPath and XQuery expressions that select all or parts of one or more binary data files, and combine them into a single result set. It is also possible to build a simple SQL wrapper to translate a restricted set of SQL queries into XQuery expressions for execution. We hope to package BinX as its own grid data service, using edikt's Enterprise Level Data Access Services (Eldas) [16].

7 References

- [1] Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Tuecke, S., "GridFTP: Protocol Extensions to FTP for the Grid", <http://www.wfp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf>
- [2] Antonioletti, M., Atkinson, M., Chue Hong N., Krause, A., Malaika, S., McCance, G., Laws, S., Magowan, J., Paton, N., and Riccardi, G., "Grid Database Service Specification (GGF8 Draft)", June 6, 2003, <http://www.globalgridforum.org/documents/Drafts/default.htm>
- [3] Astrogrid project webpage <http://www.astrogrid.org/>
- [4] "Binary Format Description (BFD) Language", <http://collaboratory.emsl.pnl.gov/sam/bfd/>
- [5] BinX project webpage <http://www.edikt.org/binx>
- [6] DFDL Working Group webpage <http://forge.gridforum.org/projects/dfdl-wg/>
- [7] FITS webpage <http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html>
- [8] "Goddard DAAC's Hierarchical Data Format (HDF)", http://daac.gsfc.nasa.gov/REFERENCE_DOCS/HDF/gdaac_hdf.html
- [9] QCDGrid project webpage <http://www.gridpp.ac.uk/qcdgrid/>
- [10] Resource Description Framework (RDF) Model and Syntax Specification, Editors: Ora Lassila and Ralph R. Swick, W3C Recommendation, 22 February 1999, <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>
- [11] Sgouros, T., "DODS User Guide, Version 1.11", May 22, 2003, <http://www.unidata.ucar.edu/packages/dods/user/guide-html/guide.html>
- [12] VOTable webpage <http://www.usvo.org/VOTable/>
- [13] Westhead, M. Chappell, A., "Data Format Description Language - Structural Description (GGF8 Draft)", June 4, 2003 <http://www.globalgridforum.org/documents/Drafts/default.htm>
- [14] M. Westhead, T. Wen, R. Carroll, "Describing Data on the Grid", to appear in 4th International Workshop on Grid Computing (Grid2003), in conjunction with Supercomputing 2003, November 17, 2003, phoenix Arizona.
- [15] XSL – The Extensible Stylesheet Language Family <http://www.w3.org/Style/XSL/>
- [16] Baxter, R., Ecklund, D., Fleming, A., Gray, A., Hills, B., Rutherford, S. and Virdee, D. "Designing for Broadly Available Grid Data Access Services", August 2003, All Hand Meeting (these proceedings).