

Databases Preservation Issues

Author: Remco Verdegem. Project manager, Digital Preservation Testbed, The Netherlands.

This paper is based on a presentation to the *Erpanet* workshop on *Long-term Preservation of Databases* in Bern, Switzerland on 9 April 2003. Slides from the original presentation can be found at <http://www.digitaleduurzaamheid.nl>

Introduction

This paper presents some of the work we have undertaken at the Digital Preservation Testbed on the preservation of databases. This Erpanet workshop is actually organized a few weeks too early as we only recently started exploring the preservation of databases. Although we did some small experiments, converting databases to XML, we expect to finish the experiments with databases at the end of May. Our final recommendations will be available on our website. I will begin by introducing the background and scope of the Testbed project, putting our work into context.

I will then present a broad-brush overview of some database preservation issues in order to provide a conceptual framework for the main focus of the presentation: bilateral database to XML conversions.

This paper will close with a description and screenshots of the conversion tool we have designed and developed ourselves to convert databases to XML.

Background

The Digital Preservation Testbed is part of a wider Dutch Initiative called the Digital Longevity Programme. Other projects in the Digital Longevity Programme included the Digital Longevity Taskforce, a project on Record Keeping systems, and a Quality Survey. The different projects work together to complement each other.

The Testbed itself was established in October 2000 by the Ministry of the Interior and Kingdom Relations and the Ministry of Education, Culture and Sciences (of which the Dutch National Archives is a linked institution).

The Testbed is a three-year research project with the overall goal of investigating options to secure sustained accessibility to authentic archival records over the long term.

The Testbed is a practical research project that carries out experiments in a controlled and secure environment. This allows us to ascertain the effects of undertaken preservation action on archival records. Our direction is dictated by the Research Questions laid down at the beginning of the project.

Research Questions

The Research Questions have three main areas of interest: General, Metadata, and Attribute-based. General research questions include:

- What are the advantages and disadvantages of implementing the different preservation approaches?
- How can the effectiveness of each approach be measured and or demonstrated?
- What are the factors that affect the effectiveness or appropriateness of each preservation approach? For example, cost? Record type? Authenticity requirements and retention periods?
- What are the basic requirements for preservation functions? For example, what are the requirements for accessing and retrieving records from the preservation function?

Metadata research questions address such issues as:

- What factors affect the metadata required for preservation? For example, record type and preservation approach, and how?
- What are the options for associating metadata with records?

We also consider attribute research questions. The Testbed classifies electronic records according to the five attributes identified by Rothenberg¹. These are: Content, Context, Structure, Appearance and Behavior. We consider such aspects as

- What are the options for preserving record attributes?
- What is the relationship between the preservation of specific attributes and the cost of preservation?

In all, these Research Questions cover a very broad spectrum.

Scope

Such a broad spectrum, in fact, that our scope is limited to four specific record types. We consider digital preservation from the file level upwards and are concerned only with records that are born digital. The four record types chosen for inclusion in our experiments are:

- Text documents - for example, MS Word or WordPerfect documents
- Emails – from, amongst others, Outlook, Eudora, Novell GroupWise, Hotmail, and KMail
- Spreadsheets – including MS Excel and Lotus
- Databases – for example, MS Access and Oracle

Within these four record types, we examine three preservation approaches:

- Migration – the transfer of digital materials from one hardware and/or software platform to another;
- Emulation - the recreation of one hardware and/or software environment on another; and
- XML. The XML approach can be implemented in various different ways. For example, conversion from another format into XML can be considered as a particular type of migration technique. XML is also a highly promising original data format for archiving and interoperability. It therefore deserves to be considered as a preservation approach in its own right.

What is a database?

There are several definitions available, for example a database is a collection of data created for a specific purpose and accessible for one or more computer systems. In this paper I refer to three distinct components or elements of a Database System:

- (i) The contents of the database.
- (ii) The database management system (DBMS) – e.g. Oracle 8i.
- (iii) The database application – for instance an Oracle Forms or a Visual Basic application to enter/display data and to run backend data processing scripts.

The database system comprises all three elements. In this paper the term database is used to include at least the contents of a database.

Different types of databases

They're any many types of database management systems, the main types being:

¹ Rothenberg, Jeff & Bikson, Tora: *Digital Preservation - Carrying Authentic, Understandable and Usable Records Through Time* (Digitale Duurzaamheid, The Hague, 1999).

- (a) Relational database - Oracle Databases, Microsoft Access
- (b) Hierarchical database – a database organized in the form of a tree structure
- (c) Native XML database – Tamino
- (d) Object database - the Testbed System uses Oracle iFS to map objects to an underlying relational database. So whilst this is not an example of a native object database it is an example of how object related commands could be issued to a database.
- (e) Network database

Object, hierarchical and network databases are not widely used. Native XML databases are new, so not widely used yet. Our research within the Testbed will focus on relational databases, which is the dominant database type particularly for large government systems.

Why is the issue of database preservation particularly difficult compared to other record types/formats?

- (a) Each database system (content, database management system, application) is unique, so if the contents of the database were converted to a preservation friendly format, the application to read in the preservation friendly data and use or view or process the data in the same way as the original database system would have to be custom developed in each case.
- (b) The native environment (the application) to change or use or view the data is usually not widely available and is generally database specific. Knowledge about how to use these customized systems is generally not widespread.
- (c) The technical challenge for reliably converting a range of or all aspects of databases into a preservation friendly format (and back again) is high. This is because the basic structure, i.e. a number of linked data tables and number of features, e.g. joins, indexes, user issues, etc associated with a relational database make it complex - it is not just a simple flat file. Also, the structure of data in a relational database is far removed from a preservation friendly format - it is designed to take up minimal storage space, to maintain consistency of the data and for operational efficiency. However, a lot of work has been put in by the IT industry to address these issues, so tools may be available to help us.
- (d) Operational database contents are subject to frequent changes. These are the transactional changes associated with user/application generated database updates, inserts and deletes. There are also the even more rapid changes of dynamic performance tables, which are continuously updated while the database is open, and in use. These latter tables/views relate primarily to database performance.
- (e) Furthermore the relationship between a record and a database is unclear and it is context dependent.

Complex relationship between databases and records

I am not referring to the confusion between archivists and IT people when the term record is used. As you know for IT experts a record generally is a row in a database, whereas for archivists the term ‘record’ is reserved for an official document that needs to be archived. In this context I am using record in the archival sense of the word. But even then: in case of a text document, a spreadsheet or an email message, I think it is clear what the actual digital record is, namely the text document, the spreadsheet or the email message. With a database that relationship is not that unambiguous.

There are different options:

- (a) Records are contained, as whole objects, in the database.
- (b) The contents of the database contain records. Each record is spread over tables.
- (c) The contents of the database is the record.
- (d) Certain database data (in the database as whole objects or spread across tables) accessed and presented in a precise manner in the application form a record. A number of records can be formed in the database system, as long as the correct data is accessed and presented in the correct manner in each case.
- (e) The whole database system is the (archival) record.
- (f) A database is not an archival record at all.

So what are we trying to preserve?

In other words: what should be stored that is sufficient to reproduce the record in an authentic manner, or what should be stored to constitute the record (multiples are possible):

- (a) The data record/s as discrete, easy to access entities.
- (b) References relating to the requirements to form/reproduce records, for example, the steps required to use the application to reproduce each record.
- (c) The ability to reconstruct the contents of the database.
- (d) The whole database system.

Dutch Archival Regulation

Dutch archival regulation, article 6e:

”for databases: the original storage format or ASCII (flat file, with separator tokens), accompanied by documentation, preferably as an XML-DTD, about the structure of the database (at least encompassing the complete logical data model with a description of the entities); queries should be stored in the query language SQL (SQL2)”.

The Dutch archival regulation contains the direction that databases should be stored in their original file format or as a flat file whilst the structure should preferably be described in a DTD. This raises the question: why use a DTD if the content of the database itself does not need to be stored in XML? The Regulation says nothing about the question whether a database should be saved as a transactional system rather than as a series of snapshots in time.

Scope of our work so far

The focus of the Testbed database research is on relational databases, mainly because relational databases are widely used within the Dutch government. Testbed will investigate the migration of small and large databases, but we are currently concentrating on the conversion of databases to XML, specifically their content and structure. We have reviewed three commercial tools available for converting MS Access format databases to XML. Access 2002 itself, WinAllora Express from HiT Software and XML Junction from Data Junction Corporation. Each of these is good quality software with a well-designed user interface. WinAllora and XML Junction were able to interface to a large number of different database packages. However, we found that it required a lot of work to configure them to give us the output we required, which is to map the database information to the XML in the desired structure. It was difficult to fully control the final form of the XML in particular to transform the entire contents of a database in a single set of operations, which is why we decided to design our own tool. We wanted to be able to choose how the information in the database was distributed over the XML files, which information to include and which information to leave out.

However, there is a lot of software around for accessing databases in various ways, so there are good building blocks available to make it quick to build your own system - as we found with our own Java-based tool, that uses the Java 2 Standard Edition (J2SE) java.sql package, connected to ODBC drivers via JDBC for each database. For creating the XML, Testbed used the J2SE Java API for XML Parsing (JAXP) and the Apache Crimson XML Parser classes.

XML pros and cons

Pros:

- XML is an open standard controlled by the World Wide Web consortium, widely accepted and applied, and well structured;
- XML is platform and program independent, and removes the dependency of the present day database system;
- It is a practical approach to the concept of separation of content, structure on one side and appearance on the other;
- XML is extensible and controllable and can be read by both humans and machines;
- XML is free of license – you don't have to pay royalties;
- Because XML is widely used, a lots of software tools is already available.

Cons:

- XML is verbose. Yes it is human readable, but sometimes the file might be just too big to read!;
- XML is complex material and much pioneering work is still to be done;
- What to do with the XML once you have got it? To access the information you could read it back into a future database software package, or you could use standard XML query tools, or you could use custom software. None of these are particularly simple. Though if you are representing complex information, then you are likely to need to do something complex to access it and analyze it.

Alternative file formats:

- Original file format
This seems to be a bad idea. Although there are some tools that can understand the native files of database systems, in general, you need to have the original database software, or at least a later version of a database from the same vendor to reliably read database files. The dependency on specific proprietary software is one of the key problems of digital longevity.
- ASCII
You can do most things with an ASCII file that you can do with an XML file, but the advantage of XML is that it offers a standardized approach to defining and documenting the structure of the file. Formatted ASCII files have been used as a storage and interchange format for complex data since ASCII was invented and it can work quite well. However, with your own ASCII format, you need additional metadata to define structure and the purpose and sizes of different fields etc. to achieve the things that XML does for you. Also ASCII in the strictest sense cannot deal with unusual characters - so you need to choose one of the extensions to ASCII. Unicode is becoming the accepted standard in this area, and XML uses Unicode.

XML versus migration to new database

A third alternative would be to migrate to a new version of the database management system. Typically migrations between databases need to have both systems (the old and the new version) running at the time of the migration. This would mean that the migration needs to happen every few years. There is a lot of work involved in such a migration and it requires specialist knowledge, so it would be expensive. Conversion to XML can be seen as an intermediate step in a migration between a present day and a future database. By converting to XML, you remove the dependency on the present day database system. Although you might still need to do the second half of the migration in order to efficiently access the information. But even if you don't do the second half, you no longer risk losing the information forever.

Progress so far

Problems we encountered developing and using the tool:

- Our conversion tool uses Java function calls (from the package `java.sql` - a standard part of the Java language), which is used to get information from the database via JDBC or an ODBC/JDBC bridge. This means that, in principle, the same program code can be used to access any database that supports ODBC. This meant that we did not have to make any significant changes in the code when we switched from using MS Access to using Oracle.
But...in practice, the details are different for different databases. Separating the data tables from the system tables can be complicated. The way in which the information about constraints is stored is different for Oracle and Access.
- Not all of the methods in `java.sql` work in all cases. This is probably a limitation of the ODBC drivers.
- In Oracle, the data tables for one application are typically associated with a particular table owner and that is the best way to decide which tables are the ones you want, so you need to know the relevant owner account.
- Resources: our approach requires building a Document Object Model (DOM) representation of the XML file in memory before writing it out to a file. This works fine for small to medium tables, but for very large tables this requires a lot of memory. One of the Oracle system tables we converted caused the Java Virtual Machine to run out of memory. The alternatives are a big powerful computer, or changing the approach to a more efficient one that doesn't try to store the whole thing in memory at one time. For similar reasons, large tables also cause a problem when reading the XML file into Internet Explorer for viewing. This could be solved with a different viewer, designed for large files.
- Images in the Access database: the Categories table (of the Northwind database that is part of MS Access) contains bitmap images in a column called "Picture". In Access, the data type for this column is given as "OLE-object". The XML converter gets the type via ODBC and calls it "Long Binary". The characters stored in the XML file appear to be a hexadecimal representation of the file (1 character, 0-9 or A-F, for each 4 bits of the original). However the converter program is not able to extract any information that explains what this format is, or what kind of program would be needed to represent the file. We know it is a bitmap from looking at the original. It is possible that we could decode the hexadecimal representation and store it in a file and then use standard bitmap viewers to process it, but haven't tried that yet. This is an example of a more general problem, similar to the problem of e-mail attachments. It is possible to store all kind of binary files in databases. Converting the database to XML

doesn't help you understand what to do with these files. One does need extra metadata and a separate preservation strategy for each binary file type.

How does our database to XML conversion tool work?

The tool concentrates on the data in the database. Our XML format is designed to closely follow the structure of the relational database itself.

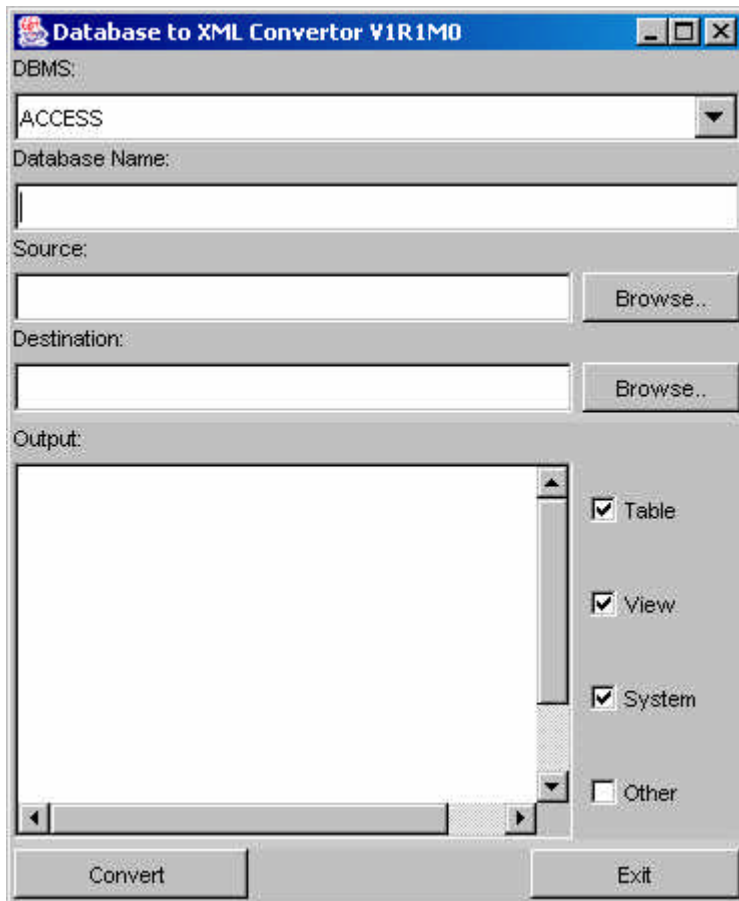


Figure 1: GUI of the Database to XML convertor tool

We have one XML file for each table in the database. This is structured as a series of rows, with each row containing a series of elements corresponding to the contents of the columns of the database table.

```

<?xml version="1.0" encoding="UTF-8" ?>
- <Suppliers >
  - <row >
    <SupplierID>1</SupplierID>
    <CompanyName >Exotic Liquids</CompanyName >
    <ContactName >Charlotte Cooper</ContactName >
    <ContactTitle >Purchasing Manager</ContactTitle >
    <Address>49 Gilbert St.</Address>
    <City>London</City>
    <Region>NULL</Region>
    <PostalCode>EC1 4SD</PostalCode>
    <Country>UK</Country >
    <Phone>(171) 555-2222</Phone>
    <Fax>NULL</Fax>
    <HomePage>NULL</HomePage>
  </row >

```

Figure 2: part of the XML file, describing the table Suppliers

We have a separate overview XML file which lists all of the tables and describes their important properties, for example the data type of each column and any constraints: whether a column is unique, or “not null”, a primary key, or a foreign key etc. So the constraint information in the overview file describes the structure of the database - it could be used to produce an entity-relationship diagram as required in the Dutch regulation, article 6e.

```

- <database name ="Test" databaseProductName ="ACCESS"
  databaseProductVersion="04.00.0000" userName ="admin">
  - <table name ="Categories" remarks="">
    <column name ="CategoryID" dataType="COUNTER" nullable="no"
      auto-Increasing="yes" maxLength="11"
      ColumnLabel="CategoryID" />
    <column name ="CategoryName" dataType="VARCHAR"
      nullable="yes" auto-Increasing="no" maxLength="15"
      ColumnLabel="CategoryName" />
    <column name ="Description" dataType="LONGCHAR"
      nullable="yes" auto-Increasing="no" maxLength="1073741823"
      ColumnLabel="Description" />
    <column name ="Picture" dataType="LONGBINARY" nullable="yes"
      auto-Increasing="no" maxLength="2147483646"
      ColumnLabel="Picture" />
  - <constraint name ="CategoriesProducts" type="PRIMARY KEY">
    <columnName >CategoryID</columnName >
  </constraint >
</table >

```

Figure 3: part of the XML overview file, describing the table Categories, including constraint information.

We also (optionally) store database views - which shows some of the ways that users could access the data in the database. However, this may not be the whole story as that could also be controlled by program code outside of the database.

What we don't do: is to re-organize the data from the database into "objects" or "records" or other collections of items that might more closely resemble how the original user of the database would access the data.

Conclusions

- Preservation of databases is still largely "uncharted territory".
- There are still a lot of questions and just a few answers. An important issue that needs to be dealt with is appraisal, making the distinction between the technical and the archival record.
- XML is able to preserve content and structure.

For further information, visit our website: <http://www.digitaleduurzaamheid.nl> or contact our team directly: Testbed@ictu.nl