XML-TK Binary Format

The XML-TK is a collection of simple light-weight tools which will let the user combine these tools in a flexible manner to accomplish powerful data processing transformations. The manner in which the tools are combined bears great similarity to the manner in which the various unix tools are combined using the pipe operator (which is represented by '|'). Since the toolkit is expected to deal with huge amounts of data, it is quite possible that the cost of carrying out a set of operations in a pipeline could be dominated by the cost of data flowing from one stage of the pipeline to the other. The idea behind XML-TK binary format is to come up with a smart encoding for XML data which will reduce the size of data flowing between the successive stages in the pipeline and hence the transmission time. This encoding also preserves the structure of the original XML document.

XML-TK Binary Content Structure

The following data types are used in the specification of the XML-TK Binary format.

Data Type	Description
byte	8 bytes of opaque data
u_int8	8 byte unsigned integer
mb_u_int	unsigned integer encoded in multi-byte integer format

Multi-byte Integers

This encoding uses a multi-byte representation for integer values. Multi-byte integer consists of a series of octets, where the most significant bit is the continuation flag and the remaining seven bits are a scalar value. The continuation flag indicates that an octet is not the end of the multi-byte sequence. A single integer value is encoded into a sequence of N octets. The first N-1 octets have the continuation flag set to a zero (0). The final octet in the series has a continuation flag set to one (1).

The remaining seven bits in each octet are arranged with the most significant bit first. The octets are transmitted starting with the most significant seven bits. All unused bits in the first octet are set to zero (0).

For example, the integer value 160 (which is same as 0xA0) will be encoded as follows :

- 160 represented in binary is 1010 0000
- Breaking this down into sequence of 7 bits, we get 0000001 and 0100000

- Thus the two octets transmitted are **0**000 0001 and **1**010 0000.

BNF for the Document Structure

The following is the BNF like description of the tokenized structure. We will use the character '|' to designate the alternatives and capitalized words indicate single-byte tokens, which are defined later. Optional elements are enclosed in '[' and ']'. Elements may be followed by * specify zero or more repetitions of the preceding element.

```
document = header body

header = version
body = p* XGT_END

p = [ table ] pair

table = XGT_TABLE (mapping)* XGT_END
mapping = termstr, token, syntactic_entity, type_token
pair = tag content

termstr = charset-dependent string with termination
token = mb_u_int
syntactic_entity = XST_ELEMENT | XST_ATTRIBUTE
type = COMPLEX_CONTENT | INTEGER | STRING | ………

tag = [ type_override ] token
content = body | string | type

type_override = XGT_TO_TOKEN type_token

version = u_int8
```

Document

Document consists of a header followed by a body. The header at present consists of a version number. The body consists of encoded xml document interleaved with rows from the table which stores the mapping of the tags to their encoding. The character data, when identified as a known data type like integer or date is encoded. If nothing can be inferred about its type, then it is transmitted as an inline string.

Header

All the binary XML documents contain a version number in their first byte. This specifies the version number of the XML-TK binary format that is being used. The version byte contains the major version minus one in the upper four bits and the minor version in the lower four bits.

Table

The table which stores the mapping of tags to their encodings interleaves with the encoded XML data in this binary format. The reason for doing this is that in the absence of a schema or a DTD, the mapping table is not fully determined before the entire document has been seen, which would mean that any stage in the pipeline can not start till the stage before it has seen the entire document.

A better strategy is to output the encoding of a tag as soon as it gets encoded. This ensures two things. One, a particular stage in a pipeline doesn't have to wait to process the entire document before it provides any data for the following stage. Two, it ensures that the encoding of a tag is output before the actual document containing that encoded element is transmitted. As a result, the tool downstream always knows how to interpret the incoming binary stream.

However, every time a table row is transmitted, it has to be demarcated by tokens which mark the begin and the end of the table. To avoid the overhead of 2 tokens for each transmitted table entry, our encoding also allows a group of table entries to be transmitted at a given time.

A table entry consists of four items :

- `termstr` : the actual tag name or the attribute name transmitted as a string
- token : the encoding of the tag
- syntactic_entity : whether the tag is an element name or an attribute name
- type : the default type associated with the tag. (Note that, this can be overridden)

The default type is the type inferred from the schema or the information provided by the user on the command line.

The syntactic_entity gets encoded as follows :

Token Name	Token	Description
XST_ELEMENT	00	Indicates that the table entry encodes element tag
XST_ATTRIBUTE	01	Indicates that the table entry encodes an attribute name

Type Override

If the type of the content does not conform to the type specified for that it in the table entry, then that type can be overridden by using a XGT_TO_TOKEN token followed by the new type. If no known type can be inferred, then the data is transmitted as inline string. If however a new type is inferred, then this new type replaces the old type in the table entry.

Algorithm for Producing XML-TK Binary Format

All the stages in the pipeline start doing the encoding process in parallel. Each stage first encodes the tags that it sees on the command line. Then it starts encoding the input xml file (if any). Since all the stages of the pipeline do encoding in parallel, it is quite possible that

- same tag may get encoded as two different tokens in two successive stages of the pipeline.
- two different tags may have been mapped to the same token in two successive stages of the pipeline.

Let us first consider (a). Suppose the tag <abc> gets encoded as 01 in the first stage and as 03 in the second stage. The binary output produced by the stage 2 will have the tag <abc> mapped to 03.

If however, the second stage has some other tag, say <def> mapped to 01. Then, we will find the smallest number to which no tag has been mapped in the second stage and map <def> to that number.

Token Structure

Global Tokens

Token Name	Token	Description
XGT_END	0	Indicates the end of the body or the table
XGT_TABLE	1	Indicates the start of the table
XGT_TO_TOKEN	2	Indicates type override. It is followed by the token which encodes the new type.

Example

```
<bib>
     <book year="2000">
          <title> Data on the web </title>
          <author> Abiteboul </author>
          <author> Buneman </author>
          <author> Suciu </author>
     </book>
</bib>
```

The following is an annotated version of the binary stream for the above XML document.

Token Stream	Description
00	version number : XML-TK Binary Format version 1.0
01	XGT_TABLE
‘b’ ‘i’ ‘b’ 00	string
80	encoding for bib
00	XST_ELEMENT
00	type COMPLEX_CONTENT
00	XGT_END
80	**bib**
01	XGT_TABLE
‘b’ ‘o’ ‘o’ ‘k’ 00	string
81	encoding for book
00	XST_ELEMENT
00	type COMPLEX_CONTENT
‘y’ ‘e’ ‘a’ ‘r’ 00	string
82	encoding for year
01	XST_ATTRIBUTE
02	type INTEGER
00	XGT_END
81	**book**
82	**year**
0F D0	**2000** transmitted as an multi byte integer
01	XGT_TABLE
‘t’ ‘i’ ‘t’ ‘l’ ‘e’ 00	String
83	encoding for title
01	XST_ELEMENT
01	type STRING
00	XGT_END
83	**title**

‘D’ ‘a’ ‘t’ ‘a’ ‘ ‘ ‘o’ ‘n’ ‘ ‘ ‘t’ ‘h’ ‘e’ ‘ ‘ ‘W’ ‘e’ ‘b’ 00	**Data on the Web** transmitted as string
00	XGT_END (marks the end of **title** element)
01	XGT_TABLE
‘a’ ‘u’ ‘t’ ‘h’ ‘o’ ‘r’ 00	string
01	XST_ELEMENT
84	encoding for author
01	type STRING
00	XGT_END
84	**author**
‘A’ ‘b’ ‘i’ ‘t’ ‘e’ ‘b’ ‘o’ ‘u’ ‘l’ 00	**Abiteboul** transmitted as string
00	XGT_END (marks the end of **author** element)
84	**author**
‘B’ ‘u’ ‘n’ ‘e’ ‘m’ ‘a’ ‘n’ 00	**Buneman** transmitted as string
00	XGT_END (marks the end of **author** element)
84	**author**
‘S’ ‘u’ ‘c’ ‘i’ ‘u’ 00	**Suciu** transmitted as string
00	XGT_END (marks the end of **author** element)
00	XGT_END (marks the end of **book** element)
00	XGT_END (marks the end of **bib** element)

Notes

- All comments are removed in WBXML.
- The encoding scheme doesn’t incorporate entities and PI. However, it can be extended very easily to accomplish this.
- The encoding scheme doesn’t talk about the data types that it is going to support.
- The tag renaming that takes place across two adjacent stages in the pipelining may, in worst case lead to renaming of every tag.