BUSINESS XML

INTRODUCTION

Business level XML is a subset of XML designed to make business-to-business XML interchange more likely to result in robust infrastructure.

- In general it leans towards making the XML human readable whenever possible.
- A minimum specification approach allowing the semantics of the data to reflect the actual structure.
- Encouraging correct behavior even when simple mistakes such as case mis-match occur between multiple teams.
- Allows additional attributes to occur in a data structure. Think of this as having the ANY tag on the end of all specified elements in a DTD.
- Stays as far away from tightly specified formats such as XML-RPC as possible.
- Stretch.: Extends the current XML schema thinking to allow a structure that will be able to generate human readable documentation trees from the same document where the data structures are defined.
- Make the data structures readable enough that non-programming business users can understand and modify them.
- Make the data structure documentation readable enough that non-programming business users can understand and modify them.
- Protect the privacy of system design specifications when passing data structures from one system to another. A good example of this is the tag <qty>93</qty>. Both systems knows that it represents the quantity to be purchased but neither one should know where the other has modeled it as a integer, long integer, Binary coded decimal, etc. The same rule holds true for lists, arrays, hashes, etc.
-

Business Level XML is a specific subset of XML was inspired by the work done by Bert Bos as published at http://www.w3.org/XML/RDB.html. Bert's work paralleled my own in such a way that I have been referencing it ever since. I have used some semantics somewhat different than Bert's work so it can be more generally applied to the context of business-to-business XML communications. I have also reconciled it such that it is compatible with the samples published by commerce one.

The goal for this work is to publish a supportable standard approach that programmers can reference for when defining their own XML data structures and that managers can use to measure the XML structure design of their teams to determine if they are approaching the problem correctly.

This work was started as a result of issues caused when most programmers hired during 1999 came pre-disposed to xml-rpc and other strongly typed xml data structures. This xml-rpc type thinking is natural for programmers who are Java centric in their thinking. Java programmers tend towards strongly typing everything in all aspects of their thinking. I found this a serious problem because our system design was cross language, cross system, cross OS, etc. We where working in many languages and where trying to get multiple loosely coordinated teams working together and the strongly typed approach allowed a form a laziness in the interface coding that caused very fragile systems. We lost at least 20% of our total productivity over a 18 week period as a result of things breaking when a Java team changed an internal structure slightly and did not inform the Python team or visa versa. It was not until we where able to remove most of the tightly coupled RPC styled XML from the system that we actually started making adequate progress again.

The XML structures documented in this document fits with commerce one published at samples http://www.commerceone.com/xml/cbl/docs/samples.html as of Feb-2000. In particular the purchase order sample has been tested as compatible with a parser built along these assumptions. Most Biztalk samples also seem to work with this subset.

RULES FOR COMPONENT, ATTRIBUTE & ELEMENT NAMES.

- All attribute names are lower case. Actually they can be anything in the XML data string but the parser will convert them all to lower case prior to attempting to retrieve them by key value. During testing several complex XML interfaces it was found that case mis-match was one of the largest problems causing fragility in the code so this rule was imposed.
  - Correct
    - buy
    - place_order
    - placeorder
  - Incorrect
    - Buy
    - PlaceOrder
    - Place-Order
    - place order

- All words in attribute names are separated by "_". This is because the normal Smalltalk conventions of making the first letter of each word uppercase does not work to make the individual words human readable when case insensitivity is used. In our parser we automatically converted "-" to "_" during the parsing process. In this way we strongly encouraged correct behavior.
  - Correct
    - Billing_agent
  - Incorrect
    - Billing-Agent
    - BillingAgent
    - billingAgent

Billing-Agent
Billingagent

- The characers [0-9-a-zA-Z_] are allowed in element and attribute names. No spaces are allowed.
  - Correct

    Billing_agent
    Bill_67_order
  - Incorrect

    billing \ agent
    bill?agent
    bill agent

ADDING DOCUMENTATION SEMANTICS TO XML DOCUMENTS

A way was needed to provide a description of an XML structure that could generate both the XML sample data structured, schema, DTD and viable human digestible documentation of the data structure.

The basic approach was to build a sample of the data structure we need and then build a template in which to insert the extra information in a larger XML structure. The larger structure is then maintained and automated tools are used to re-generate the rest of the documents.

It should be noted that a rather different strategy was used than the normal XML schema documents since we use the inherent structure of the original sample all the way through and then utilize added forms of specialized tags to effectively document / decorate the original data structure. I think this is somewhat easier to train non programmers to read and understand than the typical RDF or XML schema approach.

THE PROCESS

- Build the. in. sample file as shown below. It will include basic and optional elements of the structure. Make sure to include two elements for those that are represent arrays.
- Run the Detail file template converter. This will take the sample file and create a template, which contains all of the additional tags needed for the .detail representation.
- Rename the generated .detail file to .detail and then edit the new tags to add in the additional information. From this point on this is the only file that you will edit. All the other files will be automatically generated.
- Run the converter utility which accepts a .detail file as input and generates the following:
  - the .html file for human documentation,
- Re-run the converter / generator utility on a regular basis to ensure that all documents are consistent.

TODO: Do additional research and determine if we can use the normal XML schema format and still embed the additional information we need to generate the human readable documentation.

AN EXAMPLE .IN.SAMPLE FILE.

```
<book>
   <title>How to write a winning business plan</title>
   <isbn>0-671-07358</isbn>
   <author>Joseph R. Mancuso</author>
   <synopsis>A clear, step by step system for writing
      a business plan   that will attract the
      financing you need.
   </synopsis>
   <publisher>Simon &amp; Schuster</publisher>
   <category>business</category>
   <category>small business</category>
   <category>finance</category>
   <category>planning</category>
   <cost>
      <amount>15.00</amount>
      <rate>175</rate>
      <currency>USD</currency>
   </cost>
</book>

The elements are listed in their natural containment.  This is kept
during the conversion to the detail file

Note: Multiple categories represent a 1..N array.   If <_required> tag is
omitted in the detail file it will be a 0..N array.
```

AN EXAMPLE .DETAIL.IN FILE

The _type, _required, _title, _desc, _sample are used to generate fractional information for use by the generated schema, dtd, etc.

Only a few fields where expanded in order to keep the sample short enough for inclusion in this document however in normal use every field will have the _type, _title, _desc, _require attributes added.

```
<book>
     <_type>structure</_type>
     <_title>Minimal data structure to describe a book</_title>
     <_desc> This structure describes the information commonly
       displayed on the cover of all books.  It is also the
       information most often searched for in most book
       sites.
     </_desc>

   <title>
     <_type>string</_type>
     <_required/>
     <_title>short title of the book</_title>
     <_desc></_desc>
     <_sample>How to write a winning business plan</_sample>
   </title>

   <isbn>0-671-07358</isbn>
   <author>Joseph R. Mancuso</author>
   <synopsis>A clear, step by step system for writing
      a business plan   that will attract the
      financing you need.
   </synopsis>
   <publisher>Simon &amp; Schuster</publisher>
   <category>business</category>
   <category>small business</category>
   <category>finance</category>
   <category>planning</category>
   <cost>
      <amount>15.00</amount>
      <rate>175</rate>
      <currency>USD</currency>
   </cost>
</book>
```

EXAMPLE DTD FILE.

EXAMPLE HTML TABLE FORMAT OF DOCUMENTATION

Field, type, required, title + description, sample

TODO: BUID a Sample HTML file containing the generated documentation as it is expected to appear coming out of the conversion utility.

XML FILE NAMING CONVENTIONS

Each discrete component is created in a sub directory matching the comp

File Type	Usage & Description.
*.sample.xml	A sample XML file demonstrating an actual set of valid data values for a set of XML files which can be passed in to and back from a component for a given service call. This file is used to generate default .detail files and is in turn generated from
*.detail.xml	A file, which contains sufficient information to automatically generate .detail, .sample, .schema, .dtd files from. This is generally manually created by editing a template, which was automatically generated from an original sample.
*.dtd	The XML DTD generated from the .detail file for each structure used in the interactive environment.
***.schema.xml**	The XML schema file generated from the detail file for each data structure currently being used.
***.sample.in.xml**	A sample data structure that is generally a input data structure for a given service. The file prefix is generally the data-structure name but that is not mandatory since the actual name is specified in the *.component.detail.xml file which is documenting the component. The leading * is normally the name of the data structure which is normally the name of the service followed by "*.in.xml".
***.sample.out.in.xml**	A sample data structure, which generally represents a representative output for a given service. The leading * is normally the name of the data structure which is normally the name of the service followed by ".out".
.gen	A file which has been generated by an automated tool. The general intent is that a human will make a choice to accept the generated output and possibly replace an existing hand edited file. These file generally have

	significant empty sections

NEEDED TOOLS

- A Tool to read the .sample file and create .detail templates to save programmer typing and to ensure consistent templates.
- A tool to read the .detail files and the .component file and generate the .sample, .dtd and .schema files.

CXML DOCUMENTATION

cxml.pdf