MBA - MISMO

XML DTD
ENGINEERING GUIDELINES
(Draft)

Architecture Workgroup
MISMO
July 7, 2000

Document Version	Document Date	Change List
0.1	Jan-28, 2000	**ORIGINAL DOCUMENT (BIXBY).**
0.2	Jan-30, 2000	Correct grammar, typos (Bixby).
0.3	Feb-02, 2000	Correct/clarify grammar. Add more examples (Bixby).
0.6	Apr-18-28, 2000	Revamp with votes, amendments, and corrections from Architecture workgroup (Minton)
0.7	Jun-8-15, 2000	Incorporate comments in from industry through 1.0 DTD release (Minton)
0.8	July 7, 2000	Incorporate more comments from industry for 1.0 Final DTD release, delete some extraneous information (Minton)

XML DTD DEVELOPMENT GUIDELINE

Draft Version: 0.8
July 7, 2000
Author: MISMO Architecture Workgroup

TABLE OF CONTENTS

PREFACE

XML is not rocket science. The line below is an example of data in an XML format.

<City>Washington</City>

There are brackets around each "City" label. There is a forward slash in the second label. Why? Probably, so we can tell it apart from the first label. The important point is that the data unambiguously says, "Washington is a city".

XML provides a framework for clearly describing data. When we exchange data with our partners using the same framework, we call it a standard. When we don't, then we call it complicated, confusing and unreliable, but we provide more jobs for poor, starving software architects and consultants.

INTRODUCTION

The Extensible Markup Language (XML) is a set of rules for defining how documents and data are organized. Computer data has normally been separate from its descriptive labels, which appeared in the data specification. For example, a traditional X12 data record for Customer Relations Contact information can contain a person's name, phone number and email address, as shown in this sample record.

PER~CR~JOHN P DOE~TE~305-555-1212~FX~305-555-1213~EM~help@abc.com

The labels that describe the above data, plus the meaning of each data element are not in the data file itself. An XML representation of the above data would contain element names that clearly describe the data, in addition to the data itself.

```
<CustomerRelations>
  <Contact>
    <Name>
      <Last>DOE</Last>
      <First>JOHN</First>
      <Middle>P</Middle>
    </Name>
    <Phone>305-555-1212</Phone>
    <Fax>305-555-1213</Fax>
    <Email>help@abc.com</Email>
  </Contact>
</CustomerRelations>
```

The XML representation of the data, while verbose, provides a clear picture of the data represented. This makes it ideal for exchanging the data with other parties. Well...almost. We could also depict the same contact information in the following XML representation:

```
<CustRelationsContactInfo>
  <Name>JOHN P DOE</Name>
  <PhoneNum>305-555-1212</PhoneNum>
  <FaxNum>305-555-1213</FaxNum>
  <E-Mail>help@abc.com</E-Mail>
</CustRelationsContactInfo>
```

While the data is still the same, both the XML element names and the organization of the data have changed. The power of XML is that it allows two business partners to agree on a standard way of labeling element names and the way those elements are organized. This power increases dramatically with the addition of each new trading partner.

The Mortgage Industry Standards Maintenance Organization (MISMO), under the auspices of the Mortgage Bankers Association (MBA), is the body that facilitates the creation and maintenance of the mortgage industry XML data exchange standards. MISMO will define the XML element and attribute names that will represent the mortgage-related data elements in a standard manner, and also define how the mortgage information is organized. This document is a valid specification as set forth by the MISMO architectural workgroup for a set of guidelines to assist in the development of XML file specifications for the mortgage industry.

It is very important to emphasize to the readers of this document what we are standardizing. MISMO set out to standardize loan data sent between two organizations at a point in time. This standard is intended to span multiple transactions between trading partners. MISMO has not created a standard by which to archive loan data. Although companies are free to archive the files as they are sent back and forth within the industry, the data points and structures were not designed with archival in mind, rather, they were designed to be stateful data at an instance between organizations when they needed to move data from one to the other.

KEY DIFFERENCES BETWEEN X12 AND XML

An important discussion point that has come up numerous times in architectural discussions regarding MISMO, is the differences between the X12 standardization effort and that of MISMO and its use of XML as the specification format. While X12 and XML are basically different expressions of the same information, there are key differences and advantages to each format as summarized below.

X12 Advantages

- Smaller file size allows for faster transmission and more efficient storage.
- Common Envelope Structure already exists for transmitting X12 transactions between businesses.

XML Advantages

- Files are human-readable – can be interpreted with only a text editor.
- Each data element is preceded by an element name, which describes the data element in a simple descriptive set of words.
- There is much wider industry support for XML - Integration and conversion utilities are being provided with web browsers, databases and operating systems. This will make it easier and less expensive for small to medium size businesses to import and export data in an XML format. The Microsoft's SQL 7.0 and Oracle8i databases have the native capability to read and generate XML data. Microsoft Internet Explorer 4 and 5 have the ability to parse, validate and display XML data.
- Tools to convert XML data into printed reports or web pages are readily available. Cascading Style Sheets (CSS) are already available in a number of software packages for this purpose. Extensible Style Language (XSL) is a new standard for converting XML data into a variety of human-readable formats, and is far along in its development.
- XML is by its designed for *extensibility*. Thus, it is possible to define a format for the interchange of data between trading partners, allow those users of the standard to extend it for their own use, while preserving the property to normalize back to the standard if necessary. This allows those who are working on the standardization effort to *use data that are not part of the standard yet*, and allows users of the standard to *add their own proprietary data* to the exchange format and still be compliant to the standard.

MORTGAGE INDUSTRY XML ENGINEERING GUIDELINES

Engineering guidelines provide a framework for developing a set of data structures that are similar in format, and contain common syntax and structures. The XML Architecture Workgroup has been, and will continue to define the guidelines that will be used for the mortgage industry XML data structures. The issues listed below are some of the key areas to be discussed and approved by the XML Architecture Workgroup. This list will evolve as the process moves forward.

- **DTD versus XML Schema Format For Defining Data**
- **Element and Attribute Usage**
- **Element and Attribute Naming Conventions**
- **Tracking MISMO XML DTD Versions**
- **The use of acronyms or abbreviations in XML tag names**
- **Date and Time data specification adoption**
- **Test and QA/QC Release Guidelines**
- **Other XML Architecture Workgroup Tasks**

DTD vs XML SCHEMA FORMAT

Should MISMO adopt the DTD format or the XML Schema format? Before tackling that question, we will briefly define what we mean by DTD or XML Schema. "DTD" is an acronym for Document Type Definition – a set of rules for defining what a data set should contain, and how it is organized. In the XML 1.0 specification, DTD is a specific format for defining an XML data set. Several alternative formats or schemas for defining data sets have emerged that are more precise than the XML DTD. One of the proposed alternatives to XML DTD is called "XML Schema". Others are "XML Data", "Biztalk" and "SOX" (Schema for Object-oriented XML).

Below are sample data definitions for a simple "NAME" record. It is made up of the elements "First", "Middle", "Last" and "Generation". The "Generation" element should only contain one of the following values - Jr, Sr, II, III, and IV. If you think about it, the previous two sentences are a type of "data definition" describing the elements that are generally used for a person's name.

We will not go into a detailed explanation of the different formats in this document; there are plenty of books and web sites available on the subject. The purpose of displaying them here is to simply show what the different data definition formats look like.

"XML DTD" SAMPLE

```
<!-- Name.dtd - XML 1.0 DTD format -->
<!ELEMENT NAME (First, Middle?, Last, Generation?)>
<!ELEMENT First (#PCDATA )>
<!ELEMENT Middle (#PCDATA )>
<!ELEMENT Last (#PCDATA )>
<!ELEMENT Generation EMPTY>
<!ATTLIST Generation generationType (Jr | Sr | II | III | IV) #REQUIRED>
```

"XML SCHEMA" SAMPLE

```
<!-- Name.xsd - XML Schema (XSDL) format -->
<?xml version ="1.0"?>
<schema name = "Name.xsd"
         xmlns = "http://www.w3.org/1999/05/06-xmlschema-1/structures.xsd">
        <elementType name = "NAME" model = "open">
                <sequence>
                        <elementTypeRef name = "First"/>
                        <elementTypeRef name = "Middle" minOccur = "0" maxOccur = "1"/>
                        <elementTypeRef name = "Last"/>
                        <elementTypeRef name = "Generation" minOccur = "0" maxOccur = "1"/>
                </sequence>
        </elementType>

        <elementType name = "First" model = "open">
                <datatypeRef name = "string"/>
        </elementType>

        <elementType name = "Middle" model = "open">
                <datatypeRef name = "string"/>
        </elementType>

        <elementType name = "Last" model = "open">
                <datatypeRef name = "string"/>
        </elementType>

        <elementType name = "Generation" model = "open">
                <empty/>
                <attrDecl name = "generationType" required = "true">
                        <datatypeRef name = "ENUMERATION">
                                <enumeration>
                                        <literal>Jr</literal>
                                        <literal>Sr</literal>
                                        <literal>II</literal>
                                        <literal>III</literal>
                                        <literal>IV</literal>
```

```
                        </enumeration>
                    </datatypeRef>
                </attrDecl>
        </elementType>

        <ATTRDECL NAME = "GENERATIONTYPE" REQUIRED = "TRUE">
                <datatypeRef name = "ENUMERATION">
                    <enumeration>
                        <literal>Jr</literal>
                        <literal>Sr</literal>
                        <literal>II</literal>
                        <literal>III</literal>
                        <literal>IV</literal>
                    </enumeration>
                </datatypeRef>
        </attrDecl>
</schema>
```

“XML DATA” SAMPLE

```
<!-- Name.xdr - XML Data format -->
<?xml version ="1.0"?>
<Schema name = "Name.xdr"
         xmlns = "urn:schemas-microsoft-com:xml-data"
         xmlns:dt = "urn:schemas-microsoft-com:datatypes">
        <ElementType name = "NAME" content = "eltOnly" order = "seq">
                <element type = "First"/>
                <element type = "Middle" minOccurs = "0" maxOccurs = "1"/>
                <element type = "Last"/>
                <element type = "Generation" minOccurs = "0" maxOccurs = "1"/>
        </ElementType>

        <ElementType name = "First" content = "textOnly"/>
        <ElementType name = "Middle" content = "textOnly"/>
        <ElementType name = "Last" content = "textOnly"/>
        <ElementType name = "Generation" content = "empty">
                <AttributeType name = "generationType" dt:type = "enumeration"
                            dt:values = "Jr Sr II III IV" required = "yes"/>
                <attribute type = "generationType"/>
        </ElementType>

</SCHEMA>
```

“BIZTALK” SAMPLE

```
<!-- NAME.BIZ - XML BIZTALK FRAMEWORK -->

<?XML VERSION ="1.0"?>

<SCHEMA NAME = "NAME.BIZ"

         XMLNS = "URN:SCHEMAS-MICROSOFT-COM:XML-DATA"

         XMLNS:DT = "URN:SCHEMAS-MICROSOFT-COM:DATATYPES">

        <ELEMENTTYPE NAME = "NAME" CONTENT = "ELTONLY" ORDER = "SEQ">

                <ELEMENT TYPE = "FIRST"/>

                <ELEMENT TYPE = "MIDDLE" MINOCCURS = "0" MAXOCCURS = "1"/>

                <ELEMENT TYPE = "LAST"/>

                <ELEMENT TYPE = "GENERATION" MINOCCURS = "0" MAXOCCURS = "1"/>
```

```
        </ELEMENTTYPE>


        <ELEMENTTYPE NAME = "FIRST" CONTENT = "TEXTONLY"/>

        <ELEMENTTYPE NAME = "MIDDLE" CONTENT = "TEXTONLY"/>

        <ELEMENTTYPE NAME = "LAST" CONTENT = "TEXTONLY"/>

        <ELEMENTTYPE NAME = "GENERATION" CONTENT = "EMPTY">

                <ATTRIBUTETYPE NAME = "GENERATIONTYPE" DT:TYPE = "ENUMERATION"

                                        DT:VALUES = "JR SR II III IV" REQUIRED = "YES"/>

                <ATTRIBUTE TYPE = "GENERATIONTYPE"/>

        </ELEMENTTYPE>


</SCHEMA>
```

THESE FOUR SAMPLE FORMATS ARE DIFFERENT METHODS OF DEFINING THE SAME "NAME" DATA. WHICH ONE(S) SHOULD THE MISMO ADOPT?

- **THE "XML DTD" FORMAT IS AN APPROVED STANDARD, BUT DOES NOT ALLOW FOR PRECISE DEFINITION OF THE DATA ELEMENT SIZES OR FORMATS.**

- **THE "XML DATA" AND "BIZTALK" ARE MICROSOFT PROPOSALS FOR A DATA DEFINITION STANDARD.**

- **THE "XML SCHEMA" FORMAT DOES PROVIDE THESE DESIRED FEATURES, BUT IT IS NOT YET PART OF THE APPROVED XML STANDARD AND IS NOT FULLY SUPPORTED IN THE CURRENT INTERNET BROWSERS AND DATA PARSERS.**

Currently, the **XML DTD** format is the better method for describing mortgage industry data. It is an approved industry standard that is widely supported in the XML tools and software available today. As the XML Schema and other formats evolve, we could migrate or adopt another format. The Architecture Work Group voted on the use of the DTD specification. Once the XML schema specification has been ratified as a W3C standard, the Architecture Work Group plans to revisit this issue. It will not impact the version 1.0 releases of the specification.

What is more important initially is that we agree on an overall XML Mortgage Industry data set **structure**, and the **element and attribute names** that will be used. The DTD and XML Schema are simply different methods of describing this structure. Once the data structure and element names are defined (in the Data Model and Data Dictionary), it is fairly easy to convert this information into a DTD format, XML Schema format or any of the other data definition formats.

ELEMENT AND ATTRIBUTE USAGE

XML provides two types of structures to define data -- elements and attributes. There are no set XML rules for how these two structures are to be used, but by defining guidelines for their use, MISMO can attain some degree of commonality between its various data structures. The following are some proposed guidelines for using elements versus attributes.

Elements typically contain the actual text or data that you might see on a human-readable loan file or report. Including the full text description in the element data eliminates the need to convert a code into text when preparing a human-readable report or web page.

Attributes, on the other hand, provide more information about the element that it is included with, such as the source of the data, the type of data, or coded values for the data. You can specify an enumerated list of allowable values for an attribute, which makes them handy for storing code sets for data such as account types, loan types, loan purpose, amortization type, property type and so on. Storing codes in XML attributes, and storing text data in XML elements, allows for easy integration into existing software systems, without the need for those systems to interpret the text data.

Element Types

There are five types of XML elements commonly used in the Mortgage Industry DTDs.

- **Container Element** – These elements simply contain other elements. In the NAME example used earlier, "NAME" is the Container Element for "First", "Middle", "Last" and "Generation".

```
<NAME>
  <First>John</First>
  <Middle>L</Middle>
  <Last>Consumer</Last>
  <Generation generationType="Jr"/>
</NAME>
```

  Container elements cannot contain any text themselves. For example:

```
<NAME>
      JOHN A. JOHNSON
      <SecondMiddleName>THOMAS</SecondMiddleName>
</NAME>
```

- **Date Element** – These elements hold date data, stored in a CCYY-MM-DD format as in the date, 2000-02-09. If there is no "day" value for a date it will be stored in a CCYY-MM format as in the date, 2000-02.

```
<InterviewDate>1999-12-10</InterviewDate>
```

- **Date-Time Element** – This element holds date and time data stored in a CCYY-MM-DDTHH:MM:SS format. Note that the date and time are separated by the letter "T" (e.g. 1999-09-01T13:45:03). The "seconds" value may be omitted (e.g. 1999-09-01T13:45). Time values use the "military format" (i.e. 13:00 = 1pm, 14:00 = 2pm, etc.).

```
<ReportDateTime>1999-12-04T10:13</ReportDateTime>
```

- **Empty Element** – This is a special type of XML element that contains only attributes. **Empty** elements do not need a separate "end" tag, only a slash and a closing bracket as shown below. In

this example, `equifax="Yes"` and `experian="Yes"` are the attributes of the empty element, `RepositoriesRequested`.

```
<RepositoriesRequested equifax="Yes" experian="Yes"/>
```

- **Text Element** – This element can hold a text value consisting of a word, phrase, sentence or paragraph depending on its purpose. It could also contain numeric values such as account balances. A text element may also contain one or more attributes.

```
<LastName>Jones</LastName>
```

NOTE: As we evolve to the XML Schema or other format, we will probably add other element types that allow the data type to be defined more precisely.

Attribute Types

There are four types of XML attributes used in Mortgage Industry DTDs:

- **Enum Attribute** – This type of attribute has an enumerated list of allowable values defined in the DTD or Schema. When an **Enum** attribute is used in an XML data file, only a value defined in the enumerated list may be used. In the Name DTD sample used earlier, the `generationType` attribute limits the valid options for the attribute value to "Jr", "Sr", "II", "III" and "IV". Any other value would cause an error to be generated when the file is processed.

When `generationType` is defined in the DTD it appears as shown below. Note the list of allowed values for the attribute are enclosed within parenthesis.

```
<!ELEMENT Generation EMPTY>
<!ATTLIST Generation generationType (Jr | Sr | II | III | IV) #REQUIRED>
```

- **ID Attribute** – Each XML **ID** attribute value used in an XML data file must be unique. In the XML Credit Report DTD, the `creditRecordID` **ID** attribute has a unique value that is not duplicated. For example, the borrower's Equifax credit file might have a `creditRecordID` attribute set to "EFX-1". The co-borrower's Equifax credit file might be assigned a value of "EFX-2" and so on. These ID attributes can be used along with IDREF attributes, to link liability, public record, credit score and inquiry data to the specific repository bureau "credit file" that provided the data.

```
<CREDITFILEVARIATION creditRecID="EFX-1" repositorySource="Equifax">
        <PARTY PartyType="Borrower">
                <NAME>
                        <First>JERRY</First>
                        <Middle>L</Middle>
                        <Last>LANGER</Last>
                </NAME>
                <SocialSecNo>442628888</SocialSecNo>
                <BirthDate>1955-06-18</BirthDate>
        </PARTY>
</CREDITFILEVARIATION>
```

- **IDREF Attribute** – The **IDREF** attribute is used in data records that "refer to" or "reference" a record that has a particular **ID** attribute. For example, in the XML Credit Report DTD, each liability, public record, credit score, and inquiry record has multiple credit file **IDREF** attributes (`creditRecordIDREF01`, `creditRecordIDREF02`, etc.). If a credit report liability record had a `creditRecordIDREF01` attribute set to "EFX-1", that would indicate which credit file provided that particular liability record. The combination of **ID** attributes and **IDREF** attributes

provide a powerful method for locating and extracting specific groups of records within an XML data file.

- **Text Attribute** – This type of attribute contains a text string, which could be a code, word or phrase. Unlike the **Enum** attribute there is no list of valid values provided in the DTD or Schema file. Examples of **Text** attributes in the XML Credit Report DTD are the `codeFromVendor` and `codeFromRepository` attributes. These attributes are used to store coded values for their associated element's text values, but an enumerated list of valid values is not maintained in the DTD.

In the following sections, three approaches are given for defining a structure for this data. The DTD will be shown first, followed by a "snapshot" of actual employment data displayed in a web browser with no special programming. Of course, there are many more possible methods for structuring this data, but these three methods display a full range of possibilities.

FIRST APPROACH – XML ELEMENTS ONLY

This approach uses an "EMPLOYER" container element to hold the other employer elements.

EMPLOYER-1.DTD

```
<!-- Employer-1.DTD                                                    -->
<!ELEMENT EMPLOYER ( EmployerType?,
                     EmploymentStartDate?,
                     EmploymentEndDate?,
                     EmployerTitle?,
                     EmployerTypeofBusiness?,
                     EmployerPosition?,
                     EmployerPositionDescription?,
                     EmployerMonthsinLineofWork?,
                     SelfemploymentIndicator?,
                     EmployerName?,
                     EmployerStreetAddress?,
                     EmployerUnitorSuite?,
                     EmployerCity?,
                     EmployerState?,
                     EmployerPostalCode?,
                     EmployerCountryName?,
                     EmployerTelephone? )>
<!ELEMENT EmployerType                 (#PCDATA)>
<!ELEMENT EmploymentStartDate          (#PCDATA)>
<!ELEMENT EmploymentEndDate            (#PCDATA)>
<!ELEMENT EmployerTitle                (#PCDATA)>
<!ELEMENT EmployerTypeofBusiness       (#PCDATA)>
<!ELEMENT EmployerMonthsinLineofWork   (#PCDATA)>
<!ELEMENT SelfemploymentIndicator      (#PCDATA)>
<!ELEMENT EmployerName                 (#PCDATA)>
<!ELEMENT EmployerStreetAddress        (#PCDATA)>
<!ELEMENT EmployerUnitorSuite          (#PCDATA)>
<!ELEMENT EmployerCity                 (#PCDATA)>
<!ELEMENT EmployerState                (#PCDATA)>
<!ELEMENT EmployerPostalCode           (#PCDATA)>
<!ELEMENT EmployerCountryName          (#PCDATA)>
<!ELEMENT EmployerPosition             (#PCDATA)>
<!ELEMENT EmployerPositionDescription  (#PCDATA)>
<!ELEMENT EmployerTelephone            (#PCDATA)>
```

Here is the output from Internet Explorer 5 when browsing a data file containing a single employer data set.

EMPLOYER-1.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE EMPLOYER (View Source for full doctype...)>
- <EMPLOYER>
    <EmployerType>CurrentPrimary</EmployerType>
    <EmploymentStartDate>1996-12-15</EmploymentStartDate>
    <EmployerTypeofBusiness>Information Services</EmployerTypeofBusiness>
    <EmployerPosition>Network Security Analyst</EmployerPosition>
    <EmployerMonthsinLineofWork>120</EmployerMonthsinLineofWork>
    <SelfemploymentIndicator>No</SelfemploymentIndicator>
    <EmployerName>INFO1</EmployerName>
    <EmployerStreetAddress>6010 Dawson Blvd.</EmployerStreetAddress>
    <EmployerCity>Norcross</EmployerCity>
    <EmployerState>GA</EmployerState>
    <EmployerPostalCode>30093</EmployerPostalCode>
    <EmployerTelephone>800-699-6789</EmployerTelephone>
</EMPLOYER>
```

SECOND APPROACH – XML ELEMENT CONTAINER WITH ALL ATTRIBUTES

The following DTD structure uses the Empty Element, “Employer” to hold the remaining data, which will be stored as XML Attributes.

EMPLOYER-2.DTD

```
<!-- Employer-2.DTD                                                    -->
<!ELEMENT Employer EMPTY>
<!ATTLIST Employer  EmployerCity                CDATA  #IMPLIED
                    EmployerCountryName         CDATA  #IMPLIED
                    EmployerMonthsinLineofWork  CDATA  #IMPLIED
                    EmployerName                CDATA  #IMPLIED
                    EmployerPosition            CDATA  #IMPLIED
                    EmployerPositionDescription CDATA  #IMPLIED
                    EmployerPostalCode          CDATA  #IMPLIED
                    EmployerState               CDATA  #IMPLIED
                    EmployerStreetAddress       CDATA  #IMPLIED
                    EmployerTelephone           CDATA  #IMPLIED
                    EmployerTitle               CDATA  #IMPLIED
                    EmployerTypeofBusiness      CDATA  #IMPLIED
                    EmployerUnitorSuite         CDATA  #IMPLIED
                    EmploymentEndDate           CDATA  #IMPLIED
                    EmploymentStartDate         CDATA  #IMPLIED
                    SelfemploymentIndicator     CDATA  #IMPLIED
                    EmployerType ( CurrentPrimary   |
                                   CurrentSecondary |
                                   PriorPrimary     |
                                   PriorSecondary)     #IMPLIED >
```

Here is the output from Internet Explorer 5 when browsing a data file containing a single employer data set.

EMPLOYER-2.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE Employer (View Source for full doctype...)>
```

```
<Employer EmployerCity="Norcross" EmployerMonthsinLineofWork="120" EmployerName="INFO1"
    EmployerPosition="Network Security Analyst" EmployerPostalCode="30093" EmployerState="GA"
    EmployerStreetAddress="6010 Dawson Blvd." EmployerTelephone="800-699-6789"
    EmployerTypeofBusiness="Information Services" EmploymentStartDate="1996-12-15"
    SelfemploymentIndicator="No" EmployerType="CurrentPrimary" />
```

THIRD APPROACH – XML CONTAINER, ELEMENTS AND LIMITED ATTRIBUTES

A third method would be to use "EMPLOYER" as a container for common data elements, and two attributes.

EMPLOYER-3.DTD

```
<!-- Employer-3.DTD                                                    -->
<!ELEMENT EMPLOYER ( Name?,
                     StreetAddress?,
                     UnitorSuite?,
                     City?,
                     State?,
                     PostalCode?,
                     CountryName?,
                     Telephone?,
                     StartDate?,
                     EndDate?,
                     MonthsinLineofWork?,
                     Title?,
                     TypeofBusiness?,
                     Position?,
                     PositionDescription?,
                     SelfemploymentIndicator? )>
<!ATTLIST EMPLOYER Type ( CurrentPrimary   |
                          CurrentSecondary |
                          PriorPrimary     |
                          PriorSecondary)     #IMPLIED >
<!ATTLIST EMPLOYER SelfEmployed  ( Yes | No)  #IMPLIED >
<!-- Name, StreetAddress, UnitorSuite, City, State, PostalCode, CountryName,
     Telephone, StartDate, EndDate  are common data elements defined previously -->
<!ELEMENT MonthsinLineofWork      (#PCDATA)>
<!ELEMENT Title                   (#PCDATA)>
<!ELEMENT TypeofBusiness          (#PCDATA)>
<!ELEMENT Position                (#PCDATA)>
<!ELEMENT PositionDescription     (#PCDATA)>
```

Here is the output from Internet Explorer 5 when browsing a data file containing a single employer data set.

EMPLOYER-3.XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE EMPLOYER (View Source for full doctype...)>
- <EMPLOYER Type="CurrentPrimary" SelfEmployed="No">
    <Name>INFO1</Name>
    <StreetAddress>6010 Dawson Blvd.</StreetAddress>
    <City>Norcross</City>
    <State>GA</State>
    <PostalCode>30093</PostalCode>
    <Telephone>800-699-6789</Telephone>
    <StartDate>1996-12-15</StartDate>
    <MonthsinLineofWork>120</MonthsinLineofWork>
    <TypeofBusiness>Information Services</TypeofBusiness>
    <Position>Network Security Analyst</Position>
</EMPLOYER>
```

This DTD has several significant changes:

- The "Employment_" prefix is stripped away from common core elements like like "City", "State", "Postal Code", etc., since the data elements are contained within an "EMPLOYER" container already.

This allows the common elements to be defined once, but also used in a borrower address, creditor address, loan originator address or ANY address.

- "EmployerType" attribute becomes a simple "**type**" attribute that describes the type of employment (Prior, current, primary or secondary). The "SelfemploymentIndicator" element becomes a "SelfEmployed" attribute with a "Yes" or "No" option.
- The elements are ordered in much the same way that the Uniform Residential Loan Application shows them.

The MISMO Architecture Work Group held a vote on the above mentioned options, and voted in favor of the mixed XML element and attribute usage option. The Architecture Work Group further voted to mix the XML elements and use XML attributes in the MISMO standard DTD's as per Mike Bixby's proposal (at the beginning of this section).

ELEMENT AND ATTRIBUTE NAMING CONVENTIONS

Using naming conventions for XML elements and attributes, provides a uniform look and feel to the Mortgage Industry DTDs. Described below are four ways of attaining a useful and uniform naming convention.

Use Understandable Element Names

One of the original "intents" of the proponents of XML was that raw XML files should be easily decipherable to someone viewing the file with a text editor or browser. The meaning of Elements and Attributes tag names should be understandable to a reader who is generally familiar with the mortgage industry.

For example, an X12 data element describing the current balance of a liability would appear as "**`AMT~T4~250`**", whereas the same data would appear in an XML file as "**`<CurrentBalance>250</CurrentBalance>`**".

Element names shall not contain spaces or special characters.

Derive Element Names from a Data Model

Because of the work done by the Mortgage Industry Data Modeling work group, there was a readily available source for deriving underwriting element names for the XML DTDs. This group has also done extensive work to consolidate similar data elements into a single common element. If possible, a data model should be the first source used for XML element names. Another document explaining recommended methods for creating a process area data set, along with lessons learned from underwriting, credit, secondary, service order, and MI process groups is also discussed.

Although very useful, it is not mandated by the MISMO Architectural workgroup to use a data model when creating a MISMO Process Area DTD or logical data dictionary.

Use Standard Capitalization Formats

Although there are no XML specifications regarding capitalization of element names, there are some common practices already in use in the computer industry. The convention listed below is one recommended method for standardizing capitalization and making it easy to distinguish the function of a "name" by how it looks.

- **Container** Elements in ALL CAPS - Example: `<CREDITREPORT>`
- All **Other Element** Names in "UpperCamelCase" format – Example: `<ReportNumber>`
- All **Attribute** Names also in "UpperCamelCase" format – Example: `<X12LoanType>`

The "UpperCamelCase" and "lowerCamelCase" naming conventions for elements and attributes are used in the Biztalk Framework.

Element Ordering

The MISMO Core Data workgroup specified that the elements within the DTD needed to follow ordering guidelines. The element should be organized such that the detailed information about that element is specified in the top portion of the element declaration, followed by any grouping or contained elements. If those container elements have detailed information, that information follows each container element in sequence. The contained elements and attributes should be in alphabetical order after the initial ordering guidelines have been followed.

Use of Decimal Place Information

The MISMO Core Data workgroup specified that decimal information would be utilized in the logical data dictionary. Any data point that is a number and consists of multiple decimal points will have the number of decimal points/places specified as part of the data dictionary.

For the special type money, which is two decimal places, the Core Data workgroup specified that "money" would be added as an additional logical (dictionary) data type.

Number Convention

The MISMO Core Data Workgroup specified that when using numbers in XML names in MISMO, if the number is less-than or equal to twenty, spell the number out in the tag. If the number is greater-than 20, then you can use the Arabic number equivalent for the number. For example:

NumberLoanPaymentTenDaysLate
and *NumberLoanPayment30DaysLate*

USE STANDARD ACRONYMS, DO NOT USE ABBREVIATIONS

In some cases, using descriptive names can result in some long element tag names. To alleviate this somewhat, the XML Architecture work group considered adopting a standard set of common abbreviations for element and attribute names. If abbreviations were to be adopted, one source for common abbreviations to consider using would be the Data Interchange Standards Association (DISA) document, "*X12-XML: Representation of X12 Semantics in XML Syntax*".

The MISMO Architecture Work Group met and voted *against* the use of abbreviations, but voted *for* the use of acronyms.

The Architecture Work Group voted on a proposed list of standard acronyms. Acronyms receiving eight or more votes were ratified and included in the MISMO standard. Requests for additions to this list may be submitted to MISMO through DISA. The Architecture Work Group will vote on each submission.

Votes For Keeping	Acronym	Description	Source
11	ARM	Adjustable Rate Mortgage	CHL
11	FHA	Federal Housing Administration	CHL
11	HUD	Housing & Urban Development	CHL
10	APR	Annual Percentage Rate	CHL
10	REO	Real Estate Owned	CHL
10	SSN	Social Security Number	CHL
9	FICO	Fair Issac & CO	CHL
9	GNMA	Government National Mortgage Association	CHL
9	HMDA	Home Mortgage Disclosure Act	CHL
9	LTV	Loan to value	CHL
9	MERS	Mortgage Electronic Registration Systems	ASC
9	PITI	Principal, Interest, Taxes and Insurance	ASC, MGIC

9	PUD	Planned Unit Development	CHL
8	AKA	Also Known As	CHL
8	CUSIP	Common Unique Security Identifier	CHL
8	FDIC	Federal Deposit Insurance Corporation	ASC
8	FEMA	Federal Emergency Mgmt Agency	CHL
8	HELOC	Home Equity Line of Credit	ASC
8	IRA	Individual Retirement Account	ASC
8	IRS	Internal Revenue Service	ASC
8	LIBOR	London InterBank Offered Rate	CHL
8	RESPA	Real Estate Settlement Procedures Act	CHL
8	URLA	Uniform Residential Loan Application	ASC
8	VA	Veterans Administration	CHL

General approach to creating names

A name is formed through the use of keywords. The naming format is (from left to right) Primary Qualifier(s), Prime Word, Secondary Qualifier(s), and Class Word, as shown in the matrix below. The most descriptive term or prime word appears first, followed by successively less selective terms or secondary qualifiers. The entity name can be used as the primary qualifier to remove ambiguity for its elements. Primary qualifiers are often used to indicate ownership and/or the point in the life cycle where the business term falls. A name should be able to stand on its own, without depending on context for interpretation.

Primary Qualifiers		Prime Word	Secondary Qualifiers	Class Word
Ownership	**Life Cycle**			
Applicant	Acquisition	Product	Percent Sold	Address
Borrower	Acquired	Seasoning	Percent Acquired	Amount
CoBorrower	Acquired Property	Guaranty Fee	Percent Owned	City
Contract	Current	Region	Scheduled	Code
Counterparty	Delinquency	Principal	Actual	Comment
Credit Bureau	Distressed Asset	Payment	Estimated	Count
Credit Report	Liquidated	Yield	Applied	Day
Deal	Liquidation	Index	Ratio	Description
Fannie Mae	MBS 4 Plus	Margin	Total	Factor
Investor	Modified	Interest	Adjustment	Identifier
Lender	Monthly		Method	Indicator
Loan	Original	etc., etc.	Summary	Limit
Mortgage Insurer	Originated		Type	Month
Pool	Origination		Category	Name
Project	Previous			Number
Property	Quarterly			Percent
Region	Unmodified			Period
Security	Weekly			Rate
Security Firm	Year-to-Date			State
Seller				Term
Servicer				Time
Third Party				Year
Trade				
Trust				

There is a balance that needs to be struck between the need to create an accurate name and the need for brevity. XML is in general a verbose language, and the longer the names, the higher the ratio of tags to actual data in a 'document'. If network bandwidth is a constraining factor on the performance of a demanding application, use of very short names may be necessary. In general, names should be as long as is needed to clearly and unambiguously identify the business concept.

Primary Qualifier(s)

- Names may begin with primary qualifiers if one or a combination of primary qualifiers is necessary to remove ambiguity. A good rule of thumb is to use a primary qualifier if the prime word applies to more than one "thing" such as the loan, property, borrower, or investment (e.g., Borrower Zip Code, Seller Zip Code, Mortgage Insurer Zip Code). It is also appropriate to use a Primary Qualifier if the value of an attribute can change at various points of the life cycle such as origination, acquisition, and liquidation.
- A name may have zero, one or multiple Primary Qualifiers.
- The naming standards matrix provides examples of primary qualifiers to indicate ownership and life cycle and an order in which to display them if both are applicable (ownership first, life cycle second).

Prime Word

- The most descriptive term included in the business name is the Prime Word. The Prime Word is the foundation of the business term. All other words included in the business name describe the Prime Word in some way.
- All business names must have a Prime Word.
- If the business name has Primary Qualifiers, the Prime Word comes immediately following the Primary Qualifier(s).

Secondary Qualifier(s)

- Used to further clarify the Prime Word. Secondary Qualifiers are successively less selective qualifiers required to describe the Prime Word.
- A term may have zero, one or multiple secondary qualifiers.
- The attached Naming Standards Matrix provides a list of categories of potential secondary qualifiers and a recommended order in which to display them if more than one is applicable.

Class Word

- All attributes must end in a class word that corresponds to the data type.

Class Words and Their XML Data Types

Class Word	Definition	Domain	XML Data Type
Address	A geographic location	Character	string
Amount	Any quantity of money (dollar amount)	Float	number, decimal
Code	Identifies classifications of nouns	Character	string
Comment	General narrative or text	Character	string
Count	A number reached by keeping count	Integer	number, integer
Date	A calendar date or range of dates	Date	dateTime
Day	The day portion of the calendar date	Integer	number, integer
Description	Narrative text that defines or describes a specific thing	Character	string
Factor	A quantity that when multiplied together with another quantity yields a given product	Float, Integer	number, decimal, real
Identifier	Alphanumeric string used to uniquely identify an item	Integer, Character	number, integer, string
Indicator	Denotes that a condition is true or false	Character	boolean
Limit	The greatest or smallest amount or number allowed	Integer, Float	number, integer, decimal, real
Month	The month portion of the calendar date	Character, integer	string, number, integer
Name	Identifies specific items	Character	string
Number	A numeric reference or identification	Character, Integer, Float	number, integer, decimal, real
Percent	Ratio between data	Float, Integer	number, decimal, real

Class Word	Definition	Domain	XML Data Type
	values		
Period	An interval of time	Integer	number, integer, timePeriod
Rate	A quantitative measure expressing a cost or service per unit	Float	number, decimal, real
Term	An interval of time	Integer	number, integer
Time	The time an even occurs	Time	time
Year	The year portion of the calendar date (4 digits)	Integer	number, integer

Date and Time Specification Adoption

The MISMO Architecture work group held a vote, and decided to adopt the existing International Standards Organization (ISO) format for dates and times as referenced below.

A Summary of the International Standard Date and Time Notation

(Adapted from Markus Kuhn)
International Standard ISO 8601 specifies numeric representations of date and time. This standard notation helps to avoid confusion in international communication caused by the many different national notations and increases the portability of computer user interfaces. In addition, these formats have several important advantages for computer usage compared to other traditional date and time notations. The time notation described here is already the de-facto standard in almost all countries and the date notation is becoming increasingly popular.

Contents: Date, Time of Day, Time Zone.

Date

The international standard date notation is

YYYY-MM-DD

where YYYY is the year in the usual Gregorian calendar, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31.
For example, the fourth day of February in the year 1995 is written in the standard notation as

1995-02-04

Other commonly used notations are e.g. 2/4/95, 4/2/95, 95/2/4, 4.2.1995, 04-FEB-1995, 4-February-1995, and many more. Especially the first two examples are dangerous, because as both are used quite often in the U.S. and in Great Britain and both can not be distinguished, it is unclear whether 2/4/95 means 1995-04-02 or 1995-02-04. The date notation 2/4/5 has at least six reasonable interpretations (assuming that only the twentieth and twenty-first century are reasonable candidates in our life time).

Apart from the recommended primary standard notation **YYYY-MM-DD**, ISO 8601 also specifies a number of alternative formats for use in applications with special requirements. All of these alternatives can easily and automatically be distinguished from each other:
The hyphens can be omitted if compactness of the representation is more important than human readability, for example as in

19950204

For situations where information about the century is really not required, a 2-digit year representation is available:

95-02-04 or **950204**

If only the month or even only the year is of interest:

1995-02 or **1995**

In commercial and industrial applications (delivery times, production plans, etc.), especially in Europe, it is often required to refer to a week of a year. Week 01 of a year is per definition the first week that has the Thursday in this year, which is equivalent to the week that contains the fourth day of January. In other words, the first week of a new year is the week that has the majority of its days in the new year. Week 01 might also contain days from the previous year and the week before week 01 of a year is the last week (52 or 53) of the previous year even if it contains days from the new year. A week starts with Monday (day 1) and ends with Sunday (day 7). For example, the first week of the year 1997 lasts from 1996-12-30 to 1997-01-05 and can be written in standard notation as

1997-W01 or **1997W01**

The week notation can also be extended by a number indicating the day of the week. For example, the day 1996-12-31, which is the Tuesday (day 2) of the first week of 1997, can also be written as

1997-W01-2 or **1997W012**

for applications like industrial planning where many things like shift rotations are organized per week and knowing the week number and the day of the week is more handy than knowing the day of the month.
An abbreviated version of the year and week number like

95W05

is sometimes useful as a compact code printed on a product that indicates when it has been manufactured.
The ISO standard avoids explicitly stating the possible range of week numbers, but this can easily be deduced from the definition.

Both day and year are useful units of structuring time, because the position of the sun on the sky, which influences our lives, is described by them. However the 12 months of a year are of some obscure mystic origin and have no real purpose today except that people are used to having them (they do not even describe the current position of the moon). In some applications, a date notation is preferred that uses only the year and the day of the year between 001 and 365 (366 in leap years). The standard notation for this variant representing the day 1995-02-04 (that is day 035 of the year 1995) is

1995-035 or **1995035**

Leap years are years with an additional day YYYY-02-29, where the year number is a multiple of four with the following exception: If a year is a multiple of 100, then it is only a leap year if it is also a multiple of 400. For example, 1900 was not a leap year, but 2000 is one.

Time of Day

The international standard notation for the time of day is

hh:mm:ss

where hh is the number of complete hours that have passed since midnight (00-24), mm is the number of complete minutes that have passed since the start of the hour (00-59), and ss is the number of complete seconds since the start of the minute (00-59). If the hour value is 24, then the minute and second values must be zero. [Although ISO 8601 does not mention this, the value 60 for ss might sometimes be needed during an inserted leap second in an atomic time scale like Coordinated Universal Time (UTC). A single leap second 23:59:60 is inserted into the UTC time scale every few years as announced by the International Earth Rotation Service in Paris to keep UTC from wandering away more than 0.9 s from the less constant astronomical time scale UT1 that is defined by the actual rotation of the earth.]
An example time is

23:59:59

which represents the time one second before midnight.
As with the date notation, the separating colons can also be omitted as in

235959

and the precision can be reduced by omitting the seconds or both the seconds and minutes as in

23:59, **2359**, or **23**

It is also possible to add fractions of a second after a decimal dot or comma, for instance the time 5.8 ms before midnight can be written as

23:59:59.9942 or **235959.9942**

As every day both starts and ends with midnight, the two notations **00:00** and **24:00** are available to distinguish the two midnights that can be associated with one date. This means that the following two notations refer to exactly the same point in time:

1995-02-04 24:00 = **1995-02-05 00:00**

In case an unambiguous representation of time is required, 00:00 is usually the preferred notation for midnight and not 24:00. Digital clocks display 00:00 and not 24:00.
ISO 8601 does not specify, whether its notations specify a point in time or a time period. This means for example that ISO 8601 does not define whether 09:00 refers to the exact end of the ninth hour of the day or the period from 09:00 to 09:01 or anything else. The users of the standard must somehow agree on the exact interpretation of the time notation if this should be of any concern.
If a date and a time are displayed on the same line, then always write the date in front of the time. If a date and a time value are stored together in a single data field, then ISO 8601 suggests that they should be separated by a latin capital letter T, as in **19951231T235959**.

Time Zone

Without any further additions, a date and time as written above is assumed to be in some local time zone. In order to indicate that a time is measured in Universal Time (UTC), you can append a capital letter **Z** to a time as in

23:59:59Z or **2359Z**

[The Z stands for the "zero meridian", which goes through Greenwich in London, and it is also commonly used in radio communication where it is pronounced "Zulu" (the word for Z in the international radio alphabet). Universal Time (sometimes also called "Zulu Time") was called Greenwich Mean Time (GMT) before 1972, however this term should no longer be used. Since the introduction of an international atomic time scale, almost all existing civil time zones are now related to UTC, which is slightly different from the old and now unused GMT.]
The strings

+hh:mm, **+hhmm**, or **+hh**

can be added to the time to indicate that the used local time zone is hh hours and mm minutes ahead of UTC. For time zones west of the zero meridian, which are behind UTC, the notation

-hh:mm, **-hhmm**, or **-hh**

is used instead. For example, Central European Time (CET) is +0100 and U.S./Canadian Eastern Standard Time (EST) is -0500. The following strings all indicate the same point of time:

12:00Z = **13:00+01:00** = **0700-0500**

There exists no international standard that specifies abbreviations for civil time zones like CET, EST, etc. and sometimes the same abbreviation is even used for two very different time zones. In addition, politicians enjoy modifying the rules for civil time zones, especially for daylight saving times, every few years, so the only really reliable way of describing a local time zone is to specify numerically the difference of local time to UTC. Better use directly UTC as your only time zone where this is possible and then you do not have to worry about time zones and daylight saving time changes at all.

Extendible XML Architectures

The purpose of this section is to describe the relationships between the DTDs created by the MISMO process, the DTD enhancements used by MISMO conforming services and vendors, and the DTDs that actually govern instances of XML messages interchanged by mortgage industry players.

First of all, it is necessary to understand that the standardization of the language of business messages is essential for communication to occur. XML provides a way for many kinds of messages, using many different "languages" (also called "vocabularies") to be understood ("parsed") by a single standard piece of software called an "XML Parser". Each business language, such as the mortgage business language being developed by the non-profit XML Mortgage Partners, is expressed as a formal written linguistic model called a "document type definition" or, more commonly, "DTD". An XML Parser can scan a DTD together with a single business message, and determine whether the message conforms to the syntactic constraints imposed by the DTD. In other words, the parser reports whether or not the message is interpretable according to the business language.

The ability to use a single standard piece of software (a validating XML parser) to verify that a message conforms to a business language is enormously significant. Everyone who sends messages can determine whether their messages will be understandable when they arrive, and everyone who receives messages can determine whether the messages make sense. When a messages proves to be unprocessable (i.e., when information interchange is unsuccessful), the message's conformance or lack of conformance to the syntactic constraints of the business language to which it presumably conforms can be determined unambiguously; this makes it much easier to tell whose software was deficient, even if the software at both ends of the communication is deficient.

A DTD represents a contract between information providers, information users, and information processing system vendors. It's a model that users agree serves their needs. Information providers agree to provide information that conforms. Information processing system vendors agree to make their systems able to create and send conforming messages, and to receive and process appropriately incoming messages. Because there is a public model, formally expressed as a DTD, an electronic marketplace of ideas can form around that model. Without such a model, there may be as many different message formats and business vocabularies as there are players in the industry, or even more. This creates a situation that reduces the productive capacity and profitability of the entire industry: purchasers of information may have to buy or build software systems that understand all the formats of every information provider; most of this effort would be avoided by having a single standard format. Information providers may have to create information that conforms to the input requirements of dominant software vendors. Smaller software vendors are squashed because nobody can afford to buy or use software marketed by anyone but the dominant player. Since the dominant software vendor is left with sole responsibility for determining message formats, it is unlikely that diverse business models will be supportable, and it is unlikely that the models that will be used will meet the needs of all players. So, creating an industry-wide DTD that formally expresses the business language used in any given message type is the most essential step to be undertaken by any industry in its efforts to exploit the power of XML. The DTD is the most formal legal and technical expression of the industry's consensus about the nature of its information.

Having understood all that, industry groups often assemble a DTD hastily, so that there could be a standard as soon as possible. They naively believe that the use of the DTD formalism will solve all their information interchange problems. This doesn't work very well, for two reasons:

(1) DTDs are static, while business changes constantly. The DTD becomes less and less appropriate for current business activity, just as an old shoe becomes less and less comfortable for a growing child's foot.
(2) DTDs are monolithic, while business models vary widely.

When we realize these problems, we see immediately that the fundamental function of a DTD—to make the form and substance of business messages predictable and understandable—is at odds with the realities of incessant change and increasing diversity. What, then, are the correct answers to the questions:

(1) How can we detect the need for change in the DTD, and to change the industry-wide DTD without compromising the validity of existing messages and files, and without causing existing systems to be unable to process messages that conform to a more modern version of the industry-wide DTD?

(2) How can we permit individual businesses to deviate from the industry-wide DTD freely, in order to accomplish their business objectives, without compromising the understandability, validatability and interchangeability of the information contained in the deviant messages?

It turns out that both problems can be solved by the practice of allowing syntactic and semantic constraints of business languages ("base architectures") to be "inherited". The formalisms for declaring architectural inheritance are internationally standardized in ISO/IEC 10744:1997, in which inherited element types are termed "architectural forms". The free, open-source, and industry-dominating "SP" parser supports architectural forms and architectural validation. Extendible architectures represent an alternative to solving the problem of extending the MISMO standard for the use within an organization, and have not yet been formally adopted by the MISMO Architectural Work Group.

How architectural inheritance works

When an XML business message (an "XML instance") conforms to a base architecture, such as the MISMO architecture, at least some of its elements, when extracted from the message, form a business message that in every way conforms to the base architecture, as if the base architecture were the DTD.

Here is a fuller explanation. Normally, if a business message is parsed against its own DTD, the result of the parse corresponds exactly to the business message—the parser reports all the elements exactly as it encountered them in the message. However, if that business message declares that it inherits from a base architecture (another DTD, called a "meta-DTD" when it is used as a base architecture), then the parser can be told to ignore all of the elements of the message that are not based on (do not inherit the syntax and semantics of) corresponding element types ("architectural forms") in the meta-DTD. The parts of the message that remain are then validated against the meta-DTD, and the parser reports the parsed information just as if the "architectural instance" (the business message) that was revealed by deleting all the non-architectural features were the entire instance, and as if the meta-DTD were the DTD.

A single architecture-aware parser, such as SP (an open-source SGML parser), can "extract" any of the "architectural instances" from any XML instances that uses one or more base architectures.

Multiple inheritance

A single element in an XML instance can be based on several architectural forms (element types), one from each base architecture. This is very useful when information must be understood in several ways (i.e., "seen through the lens" of multiple architectures); it allows a single XML instance to conform to multiple DTDs, and thus provide input to diverse applications, without having to duplicate the same data content, and without having to maintain separate copies of the data content in order to make the data available under multiple DTDs. The single maintained form of the data can be automatically converted into the form needed by a given application by a single piece of software—an XML parser that does not even have to be specially configured in order to perform the extraction. There is no need for a transformation program to be written; everything needed to perform such a conversion is already inherent in the XML instance.

Recursive (or "nested") inheritance

An architectural instance, after it has been extracted, may itself have one or more base architectures. Thus, an architectural instance can be extracted from an architectural instance. Looking at the same thing from the perspective of the DTD and the meta-DTDs, the DTD declares a base architecture, which is itself a DTD that declares a base architecture. This allows enormous flexibility, on account of the fact that there is no limit on the number of recursions. If, in the course of maintaining an architecture, it becomes necessary or desirable to insert a new layer at any level of inheritance, it is easy to do so, and the net effect is to add one more possible base architecture on the basis of which a corresponding XML architectural instance can be automatically extracted.

Overview of the overall MISMO information architecture

Top level:

Certain element types, such as an element type intended to contain a human being's surname, or the name of a city or municipality, are used in more than one part of the MISMO architecture (e.g. Party or Contact). These "common element types" are declared as common element types and aggregated in a base architecture on which all corresponding elements, in all DTDs throughout the MISMO architecture, are based. These have also been called "global" DTD's.

Second level:

Each of the DTDs at the second level from the top is created and maintained by an editorial committee of persons who are expert in the knowledge domain that corresponds to the message type defined by that DTD. For example, the underwriting DTD is designed to serve the needs of underwriting activities, and it is under the editorial control of a committee of underwriting experts. The committees are required to cause any element types that they create to inherit and conform to the architectural forms of the common meta-DTD wherever there is a match. The common meta-DTD (top level) is also under the control of an editorial committee. One of the responsibilities of the common DTD committee is to review the DTDs of the other committees, and, wherever two committees have created elements with similar semantics, to create a common architectural form to which the two committees will be able to conform.

The second level consists of distinct DTDs to permit responsibility for maintaining these DTDs to be delegated to those who best understand the information being communicated, as well as the industrial requirements that must be met. It may turn out that fewer or more committees are needed; the specific organization of the committees and their responsibilities is not a system-design problem; it is a political and economic problem to be resolved by the community. The MISMO information architecture is to be flexible, and to avoid imposing technical constraints on a matter that can only be decided by human beings on the basis of non-engineering factors like the available talent pool, governing business processes, etc. This level is known as the process level of the architecture and contains the process area workgroups work product.

Third level:

The MISMO_Union DTD is the base architecture that provides a standard way for any MISMO message to contain any number of any of the message types defined by committees in the second level. The MISMO Union DTD inherits all of the committee architectures comprehensively: nothing is left out. The MISMO_Union architecture is the base architecture that comprises the essential work product of MISMO. The purpose of this level of the architecture is to gather all the data for the industry in one DTD, so that people can derive from it, and this overall DTD is known as the "release" DTD. This industry-encompassing DTD may be used to derive process specific DTDs. For example, one could generate a Credit Reporting DTD or a Service Request DTD; alternatively, one could create a DTD for the entire mortgage industry.

Fourth level:

At the fourth level are extensions to the MISMO_Union architecture that are contributed by various mortgage industry players, in order to serve their own needs, and the needs of the business partners with whom they will communicate by means of messages that conform to the MISMO_Union architecture. This is the level that is known as the "application translation layer" or ATL. Each company utilizing the standard would extend it from this level, and may or may not choose to provide these extensions to the industry or their trading partners.

OTHER XML ARCHITECTURE WORK GROUP TASKS

- **Document Maintenance and Operation Procedures for the MISMO XML Database –** All of the procedures for using and maintaining the MISMO XML Database should be clearly documented on the MISMO web site.

- **Develop Common Format For Mortgage Industry Implementation Guides –** Even though it will eventually be possible to "self-document" the XML data structures when we migrate to the XML Schema format, there is still an immediate need to provide guidance for implementation issues and details with each XML transaction set, in a common document format or template. The guides should supply numerous examples and sample data that illustrate the features of the DTD formats

- **Select or Develop an XML Routing "Framework" –** Almost simultaneous with the implementation of the first MISMO DTDs, there will be a need for some type of XML data structure to handle the routing of "business-to-business" (B2B) requests and reports. The "BizTalk Framework" has a simple data structure that is similar to the X12 ISA envelope that contains the necessary information for automated routing of documents within a intranet or internet environment. One of the tasks of the XML Architecture work group would be to examine existing routing "frameworks" and decide whether to use an existing "framework" (such as BizTalk), develop our own, or leave the choice of "framework" to the individual companies implementing XML B2B transactions.

- **Develop a common XML Error Format** – Most X12 implementations use the X12 824, 997 or similar transaction for reporting errors in the B2B environment. There is a need for a similar standard structure for the XML transactions used in the Mortgage Industry.

- **Continue extending these guidelines** – As we move forward there will be additional discussions about specifications and MISMO XML design and implementations, we will need to keep those decisions coming into this guidelines specification. It is intended that this document will evolve over time and itself be posted to the MISMO website under revision control.