



**MISMO Commercial Working Group
Architectural Best Practices Guide**

**<<Version 1.0>>
<<January 28, 2005>>**

Change History

Date & Version	By	Remarks
5/19/03, v 0.1	John McCarthy	Initial Draft
6/8/03, v 0.2	John McCarthy	Revisions and Content Addition
6/25/04, v 0.3	Dan Szparaga	Expanded the template document to include all approved and proposed Best Practice Topics
9/24/04, v 0.4	Dan Szparaga	Edited content throughout, added Introduction and October 2003 Logical Data Model, removed Glossary
11/4/04, v 0.5	Dan Szparaga	Added "Namespaces" and "Naming Conventions" Sections and made minor edits to other sections
11/29/04, v 0.6	Dan Szparaga	Removed full text from "Abstract Complex Types" pending resolution of restriction issue, changed section name of "Occurrence Constraints" to "Element and Attribute Occurrences," made edits reflecting feedback from MISMO XML Architecture Workgroup to "Data Modeling (Three-Way Join," "Namespaces" and "Element and Attribute Occurrences."
12/01/04, v 0.7	Dan Szparaga	Added additional descriptive language to "Attribute Groups" section and modified the URL's for schema validation in "Namespaces"
12/22/04, v 0.8	Dan Szparaga	Changed Namespace declarations throughout, corrected technical errors in "Attribute Groups" and "Data Modeling (Key –KeyRef)" sections, corrected grammar in "Namespaces"
1/23/05, v 0.8.1	Dan Szparaga & John McCarthy	Corrected technical errors in "Three- Way Joins," updated "Occurrences" section both with several grammar corrections and to be clearer on use="prohibited".
1/26/05, v 0.8.2	Dan Szparaga & Jeff Rafter	Corrected several errata in "Namespaces," added clarifications for usage of "Key- KeyRef", corrected the sample schema in "Three-Way Joins", corrected schemas throughout and updated the Logical Data Model.
1/28/05	John McCarthy & Kevin Jarnot	Authorization to Convert v 0.8.2 to v 1.0

Table of Contents

Change History	i
Table of Contents	ii
Introduction	1
Abstract Complex Types and Inheritance	5
Attribute groups	6
Data Modeling (Key-Key Ref)	8
Data Modeling (Three-Way Join)	14
Namespaces	19
Naming Convention	22
Elements and Attributes Occurrences	26
Commercial Logical Data Model	31

Introduction

Purpose and Scope / Mission Statement

The MISMO Commercial Architecture Workgroup is responsible for establishing the design criteria for the use of XML in data standards for the commercial and multifamily mortgage industries. It operates as a subgroup of the MISMO Commercial Working Group and the MISMO XML Architecture workgroup.

MISMO (the Mortgage Industry Standards Maintenance Organization) is a non-profit subsidiary corporation of the Mortgage Bankers Association. It develops open and vendor-neutral XML-based data standards for the American mortgage industry. The data standards consist of:

- a complete set of data transfer protocols for specific message exchange between trading partners (the Specifications);
- a technical framework for the adaptation of the broad capabilities of XML to specific mortgage industry contexts (the Engineering and Implementation Guidelines); and,
- a comprehensive listing of the specific data elements contained in the protocols in name-tag-definition format (the Logical Data Dictionary).

MISMO spans both the residential and the commercial/multifamily mortgage industries. Its Specifications are developed by Workgroups focused on specific process areas (e.g., Credit Reporting or Commercial Servicing). Generally, workgroups addressing data transfer needs of the commercial and multifamily mortgage industry operate under the direction of the Commercial Steering Committee, but interact with parallel workgroups on the residential side. In the case of the Commercial Architecture Workgroup, this interaction is extensive in order to ensure that there is a consistent implementation of XML throughout the entire MISMO effort.

Functionally, the actual XML specifications are created by the process area workgroups. The role of the Commercial Architecture Workgroup is to set forth the technical methods by which the various commercial workgroups can consistently implement XML across their activities. This includes adherence to a Logical Data Model that establishes a reference framework representing a consistent set of relationships between the various data points that constitute a commercial mortgage. The Commercial Core Data Workgroup is responsible for maintaining the Logical Data Model.

The purpose of this Commercial Best Practices Guide is to communicate the technical methodology referred to above. Its primary goal is to describe the design and engineering considerations that will be used in building MISMO's commercial-specific specifications using XML Schemas (W3C XML Schemas). The design of the Best Practices Guide is topical. A specific topic on the use of XML within the commercial/multifamily finance is treated as a chapter of the Guide. Each chapter adheres to a consistent template format in which the various issues are presented. A sample of this Template follows on the next page.

Sample Issue Name and Architecture Workgroup Approval Date

Best Practice

Summary statement of the Best Practice in a concise form. For example, “MISMO’s commercial schema design shall wherever possible use Attribute Groups to group together associated attributes and create reusable components.

Technical Overview

A top-level review of the technical issue (preferably stated in plain English). This is basically a few paragraphs summarizing the conclusion reached and why.

Detailed Technical Discussion

A detailed technical presentation of the issue/concept being reviewed. Where appropriate, it should include references to any source documents (hyperlink reference to web resources). Alternative approaches should be briefly discussed followed by the advantages and disadvantages of the option selected versus the alternatives. If this issue is related to another Best Practice, the interrelationship should be mentioned.

Examples

Where appropriate and beneficial, examples of both schema and instance documents should be included.

Guidance to Core Data and Process Areas

With each Best Practice, a set of instructions to the other non-technical areas should be created to provide a road map to these areas reviewing the impact of the Best Practice on the work that is either in progress or to be done.

If Architecture is retaining control of a document, such as the declaration of simple types, the procedure for making additions to that document needs to be communicated.

Versioning Protocol

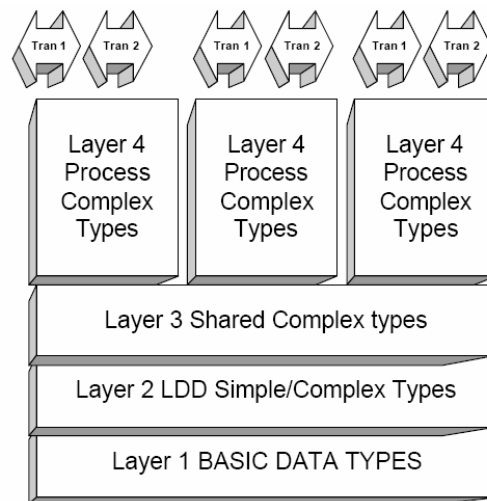
The versioning protocol of this Guide is as follows. The initial release of the Guide contains eight core topics and a representation of the Logical Data Model. Upon its approval by the MISMO XML Architecture Workgroup, this initial release will be titled “Version 1.0.” As additional topics are added, the updated Guide will be referenced by sequential sub-versions (i.e., Version 1.1). Changes to the content of previously-released editions will be noted by a change in the version number (i.e., Version 2.0).

MISMO and Schemas

Since its first specifications were released in 2000, MISMO has relied on DTD's (Document Type Definitions) as its primary specification format. This is largely because DTD's were the only officially-accepted validation format set forth for the XML 1.0 Specification by the World Wide Web Consortium (the W3C, the primary standards body for the Internet's Web framework and the creator of XML).

In May 2001, the W3C endorsed use of the more robust XML Schema specification (known generically as schemas). Schemas differ from DTD's in a number of ways, and provide a number of new capabilities that ideally suit the complex nature of commercial mortgages (mainly they support run-time data type checking and support namespaces). In 2003, the Commercial Architecture Workgroup voted to adopt schemas as its sole method for creating specifications for XML documents. Thus, from the outset (Version 1.0 of the commercial architecture), all commercial and multifamily specifications will be based on W3C XML Schemas. MISMO's XML Architecture Workgroup is in the process of planning a migration of the residential standards to W3C XML Schemas, and specifications released in that context will be referred to as Version 3.0 of the residential architecture.

At an abstract level, there will be commonality between the commercial and residential architectures. They use a consistent naming convention and approach to the use of elements and attributes. They also share a common framework for sharing data items across transactions and process areas. Internally referred to as the "Five Layer Model", this structure is represented in the following diagram:



1st level: MISMO LDD data types are defined as uniquely named xsd schema types (including each enumerated list becoming its own data type)

2nd level: Each MISMO data item in the LDD becomes a uniquely named xsd schema type that builds upon the MISMO LDD data types (level 1)

3rd level: MISMO common structures are defined as complex types that utilize the level 1 and level 2 definitions (e.g. ADDRESS, CONTACT). These structures are NOT "ELE"s they include any child containers (elements) that are a part of the common structure.

4th level: MISMO Process Area structures are defined as xsd complex types that utilize the level 1, 2 and 3 structures/types.

5th level: MISMO Transactional structures are defined as xsd complex types that utilize the level 1, 2, 3 and 4 structures/types.

In the context of the commercial architecture, Layer 1 data types are consistently defined by the Commercial Architecture Workgroup, along with the MISMO XML Architecture Workgroup. Layer 2 and 3 types are typically defined by both the Commercial Architecture and Core Data Workgroups. Layer 4 and 5 types are typically defined by the commercial process area workgroups, but must adhere to the Logical Data Model maintained by the Commercial Core Data Workgroup.

It is important to note that, because of the fundamental differences between the commercial and residential industries, commercial structures and specifications will not necessarily be analogous to residential structures and specifications. Readers should consider that this Best Practices Guide is specifically designed in a commercial context, for the 1.0 commercial architecture.

- *John McCarthy, Deutsche Bank, MISMO Commercial Architecture Workgroup Co-Chair*
- *Kevin Jarnot, Debt X, MISMO Commercial Architecture Workgroup Co-Chair*

Abstract Complex Types and Inheritance

(Approved by MISMO Commercial Architecture Workgroup May 5, 2004)

Notice

In May 2004, the MISMO Commercial Architecture Workgroup approved the concept of using Abstract Complex Types to define a base container for any container that can have variable content models. Properties of an Abstract Complex Type will be passed on (i.e., inherited) from a higher-level root container. The initial investigation into the use of this mechanism was spurred by a question from the Core Data group. Can a schema dictate the content for different loan types (i.e. Fixed vs. Floating)? The answer is “yes” through the use of Abstract Complex Types.

The Commercial Architecture Workgroup concluded that the optimal method to capture the complex sets of data relationships for commercial mortgages was the use of inheritance by restriction. However, it was noted during the November 2004 review process that there may be implementation issues resulting from the use of restriction – particularly, a number of XML parsers may have difficulty correctly processing schemas.

It was decided that Version 1.0 of this Best Practices Guide would omit this section while the Commercial Architecture Workgroup further analyzed the issue. Once the issue is resolved, the Best Practices Guide will be updated to include this section. This updated Guide (Version 1.1) is planned for release in March 2005.

Attribute Groups

(Approved by MISMO Commercial Architecture Workgroup September 15, 2004)

Best Practice

When common sets of attributes are used across multiple complex types, these attributes shall be declared as an Attribute Group.

Technical Overview

XML schema provides a built-in mechanism that promotes the modularity of a schema. Attribute Groups are used to represent groups of related attributes that appear in one or more complex types. They can be declared in a single location and then be referenced from multiple schemas. Their use helps facilitate *reuse*, *schema readability* and long-term *schema maintenance*. The use of attribute groups follows the overall approach is inline with the “Five Layer” approach adopted by MISMO. The attribute groups will be included in the Layer 3 schema and “included” in the Layer 4 schema that will define the complex type.

The design of schema documents should always consider the effect to the instance document creations. A side benefit of the use of attribute groups is that it does not have any consequence to the instance documents.

Detailed Technical Discussion

An attribute group must be declared as a global element with a unique name attribute. The group is referenced within a complex type definition by including another `xs:attributeGroup` element with a `ref` attribute that matches the top-level attribute group name.

Examples

Basic Use

```
<xs:attributeGroup name="addressGroup">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      An attribute group whose attributes
      constitute a mailing address.
    </xs:documentation>
  </xs:annotation>
  <xs:attribute name="street" type="xs:string" use="required"/>
  <xs:attribute name="city" type="xs:string" use="required"/>
  <xs:attribute name="state" type="xs:string" use="optional"/>
  <xs:attribute name="country" type="xs:string" use="required"/>
  <xs:attribute name="postalCode" type="xs:string" use="optional"/>
</xs:attributeGroup>

<xs:complexType name="Borrower">
  <xs:attribute name="borrowerName" type="xs:string"/>
  <xs:attributeGroup ref="mismo:addressGroup"/>
</xs:complexType>
```

```
<xs:complexType name="Lender">  
  <xs:attribute name="lenderName" type="xs:string"/>  
  <xs:attributeGroup ref="mismo:addressGroup"/>  
</xs:complexType>
```

Nested Attribute Groups

(As the use of nested attribute groups is fully anticipated in the commercial schemas, examples will be added into this Guide at a later time when they are developed.)

Guidance to Core Data and Process Areas

- If a related set of attributes is common to multiple complex types and represent an atomic set of data, an Attribute Group should be declared and used in the complex type definitions.
- Use annotations when defining all Attribute Groups.
- When declaring an Attribute Group, provide a unique *name* attribute.
- Once the data dictionary and logical data model are stabilized, element contents should be reviewed for formation of Attribute Groups prior to schema construction.

Data Modeling (Key-KeyRef)

(Approved by MISMO Commercial Architecture Workgroup May 26, 2004)

Best Practice

MISMO's commercial specifications will be based on the data relationships set forth in the Commercial Logical Data Model.

Technical Overview

In order to consistently represent across process areas the complex data relationships that represent a commercial or multifamily mortgage, the MISMO Commercial Core Data Workgroup has established a Logical Data Model (the LDM). Adherence to the LDM will provide efficiencies in the development of schemas for commercial transactions, since it provides a single, comprehensive reference source for all workgroups to rely upon and eliminates the creation of redundant or contradictory data representations. The Commercial Architecture Workgroup will play a key role in the process of transcribing these relationships to schemas. In summary, the preferred method of modeling relationships is via containment. This can cover all one-to-one and most one-to-many relationships. In the case of many-to-many relationships, the relationships should be represented with the use of the `xs:key` and `xs:keyref`.

Detailed Technical Discussion

One-to-One and One-to-Many Relationships

As XML is a hierarchical design, parent/child relationships are modeled with the parent element including the related child as a child element. If the relationship is one-to-one, this constraint can be imposed by setting the `maxOccurs` attribute to 1. For one-to many relationships, the `maxOccurs` attribute can be set to any number greater than one or left open ended by setting the value to "unbounded".

An example follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="DEAL">
    <xs:complexType >
      <xs:sequence>
        <xs:element name="LOAN" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PREPAYMENT_TERMS" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

This simple example shows both a one-to-one and a one-to-many relationship. The PREPAYMENT_TERMS element is a child of LOAN and the one-to-one relationship is enforced by setting the maxOccurs attribute to 1. In the case of the DEAL > LOAN relationship, a DEAL can have one or more LOAN elements and this is enabled by setting the maxOccurs attribute to "unbounded".

Many-to-Many relationships

Modeling many-to-many relationships, including the enforcement of referential integrity, can be accomplished through the use of the xs:key and xs:keyref. These two elements work in concert to form a bond by establishing a named xs:key that is similar to a database primary key. xs:key is normally accompanied by the use of xs:unique to insure that the values used as the xs:key are unique within its context established by an xpath. This named key is then utilized by the xs:keyref to form the relationship. By establishing this relationship, a validating parser will enforce that each child must point to a valid parent (i.e. no orphans). But this will only create a unidirectional relationship. A second key/keyref pair must be established to create the bidirectional relationship (i.e. no orphans and no childless parents)

In Commercial Loans, each loan may be secured by one or more properties and each property may be collateral for one or more loans. This Loan to Property many-to-many (MTM) relationship will be used in the examples. To extend the key/keyref mechanism to a MTM relationship, the referring side of the relationship must be moved from an attribute to an element to allow for multiple occurrences.

Starting with the following simple structure

```
<SAMPLE
  xmlns="http://www.mismo.org/schema/commercial/yyyy/mm/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mismo.org/schema/commercial/yyyy/mm/ sample.xsd">
  <A A_ID="1"/>
  <A A_ID="2"/>
  <B/>
  <B/>
</SAMPLE>
```

A schema would look like this:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="SAMPLE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="A" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="A_ID" type="xs:string"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="B" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
</xs:schema>
```

The three steps to establishing the foreign key reference is to first establish a unique identifier (i.e. a primary key):

```
<xs:unique name="APrimaryKey" id="APrimaryKey.Key">  
  <xs:selector xpath="/.mismo:A"/>  
  <xs:field xpath="@A_ID"/>  
</xs:unique>
```

This is added to the SAMPLE element to establish that the ID must be unique for all A elements under the root SAMPLE. The <xs:selector> and <xs:field> establish the xpath expression that is evaluated to establish the scope of the <xs:unique>. The placement of the <xs:unique> is critical to establish the proper scope of the uniqueness constraint.

The next step is to establish the fact that this reference will also use a <xs:key> that will be referenced by an <xs:keyref>:

```
<xs:key name="AIDKey" id="AIDKey.Key">  
  <xs:selector xpath="/.mismo:A"/>  
  <xs:field xpath="@A_ID"/>  
</xs:key>
```

Notice that the <xs:selector> and <xs:field> setting are identical.

The final step is to add an attribute within that refers to an A_ID and then create an <xs:keyref> that points to the <xs:key> just created.

```
<xs:attribute name="A_IDRef" type="xs:string" use="required"/>
```

The keyref looks like this:

```
<xs:keyref name="BtoARef" refer="mismo:AIDKey">  
  <xs:selector xpath="/.mismo:B"/>  
  <xs:field xpath="@A_IDRef"/>  
</xs:keyref>
```

And the complete revised schema now looks like this:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"  
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"  
  elementFormDefault="qualified" attributeFormDefault="unqualified">  
  <xs:element name="SAMPLE">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="A" minOccurs="0" maxOccurs="unbounded">  
          <xs:complexType>  
            <xs:attribute name="A_ID" type="xs:string"/>  
          </xs:complexType>  
        </xs:element>  
        <xs:element name="B" minOccurs="0" maxOccurs="unbounded">  
          <xs:complexType>  
            <xs:attribute name="A_IDRef" type="xs:string" use="required"/>  
          </xs:complexType>  
        </xs:element>  
      </xs:sequence>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

```
</xs:sequence>
</xs:complexType>
<xs:unique name="APrimaryKey" id="APrimaryKey.Key">
  <xs:selector xpath="/mismo:A"/>
  <xs:field xpath="@A_ID"/>
</xs:unique>
<xs:key name="AIDKey" id="AIDKey.Key">
  <xs:selector xpath="/mismo:A"/>
  <xs:field xpath="@A_ID"/>
</xs:key>
<xs:keyref name="BtoARef" refer="mismo:AIDKey">
  <xs:selector xpath="/mismo:B"/>
  <xs:field xpath="@A_IDRef"/>
</xs:keyref>
</xs:element>
</xs:schema>
```

and an instance document conforming to the revision would look like this:

```
<SAMPLE
  xmlns="http://www.mismo.org/schema/commercial/yyyy/mm/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mismo.org/schema/commercial/yyyy/mm/ sample2.xsd">
  <A A_ID="1"/>
  <A A_ID="2"/>
  <B A_IDRef="1"/>
  <B A_IDRef="2"/>
</SAMPLE>
```

The referential integrity that has been created is that the A_ID must be unique across all <A> elements contained in an instance and each element must have an "A_IDRef" attribute (use="required") that must refer to a valid A_ID attribute of the <A> element.

There are two methods to model this dual key/keyref relationship.

Method 1 - Dual pointing: Child(ren) elements from both sides of the relationship point to each other;

Method 2 - Foreign Key Linking Elements: Similar in design to the standard linking table (also called a zipper table) in traditional RDMS environments.

In the Dual Pointing Method (see *keyrefSample1.xsd* and *keyrefinstance1.xml* below), both sides of the relationship (LOAN and PROPERTY) have a child element (PROP_REF and LOAN_REF, respectively) that may have multiple (unbounded) occurrences. The unique key and keyref mechanism is implemented to establish LoanID and PropertyID as the respective primary keys and establish the referential integrity constraint.

An alternate model to accomplish the same referential integrity constraint is to create a linking element structure (at the root in this example) that acts in the same capacity as a linking table used to model MTM relationship in a relational database. For convenience, this is called "Foreign Key Linking Elements" (FKLE).

Following is a comparison of the 2 methods.

The Dual Pointing method has the following advantages over FKLE :

- 1) The cross reference to the linked element is bound to the source element by containment. There is no need to search the document to find the link as with FKLE.
- 2) In creating instance documents, the creation of the FKLE would require the compilation of all references from both sides, checking for duplicates, before writing out these elements.

The FKLE method has the following advantages over Dual Pointing:

- 1) Both sides of the cross reference are coupled, as in the RDBMS environment. Determining the scope of the relationship is simple using xpath statements.
- 2) Many implementing companies will have a structure similar to FKLE in their DB systems.

By assessing the comparative advantages, the Commercial Architecture Workgroup has determined that the recommended method for modeling data with a many-to-many relationship is to establish an element to hold both sides of the relationship and bind them with key/keyref relations for both sides of the equation (i.e., Dual Pointing).

Guidance to Core Data and Process Areas

To be clear, the use of key/keyref is an advanced concept in XML and its use is recommended primarily for those relationships that cannot be modeled with more traditional methods. Proposed uses of key/keyref should be vetted with the Commercial Architecture Workgroup to ensure that unnecessary complexity is not introduced into the schema. Readers should also note that this section and the following one (“Data Modeling: Three-Way Joins”) are closely related and the two sections should be read together.

keyrefSample1.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="SAMPLE">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="A" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="A_ID" type="xs:string"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="B" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="A_IDRef" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:unique name="APrimaryKey" id="APrimaryKey.Key">
      <xs:selector xpath="./mismo:A"/>
      <xs:field xpath="@A_ID"/>
    </xs:unique>
    <xs:key name="AIDKey" id="AIDKey.Key">
      <xs:selector xpath="./mismo:A"/>
      <xs:field xpath="@A_ID"/>
    </xs:key>
    <xs:keyref name="BtoARef" refer="mismo:AIDKey">
      <xs:selector xpath="./mismo:B"/>
      <xs:field xpath="@A_IDRef"/>
    </xs:keyref>
  </xs:element>
</xs:schema>
```

keyrefinstance1.xml

```
<DEAL
  xmlns="http://www.mismo.org/schema/commercial/yyyy/mm/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mismo.org/schema/commercial/yyyy/mm/ keyrefSample1.xsd">
  <LOAN LoanID="999999">
    <PROPERTY_REF>111111</PROPERTY_REF>
    <PROPERTY_REF>111112</PROPERTY_REF>
    <PROPERTY_REF>111113</PROPERTY_REF>
  </LOAN>
  <LOAN LoanID="999998">
    <PROPERTY_REF>111112</PROPERTY_REF>
    <PROPERTY_REF>111113</PROPERTY_REF>
  </LOAN>
  <COLLATERAL>
    <PROPERTY PropertyID="111111">
      <LOAN_REF>999999</LOAN_REF>
    </PROPERTY>
    <PROPERTY PropertyID="111112">
      <LOAN_REF>999999</LOAN_REF>
      <LOAN_REF>999998</LOAN_REF>
    </PROPERTY>
    <PROPERTY PropertyID="111113">
      <LOAN_REF>999998</LOAN_REF>
      <LOAN_REF>999999</LOAN_REF>
    </PROPERTY>
  </COLLATERAL>
</DEAL>
```

Data Modeling (Three-Way Joins)

(Approved by MISMO Commercial Architecture Workgroup May 26, 2004)

Best Practice

Relationships between unrelated data elements will be defined using multiple-way joins.

Technical Overview

Multiple-way joins are a way to encapsulate information about relationships between multiple data, information which doesn't apply to any of the data individually. An example that will be used in this section of the Best Practices Guide is a three-way join between a role, a party, and a real property element. We'll assume that there are additional data that apply to this three-way combination of data for which this three-way join is necessary.

Detailed Technical Discussion

Following is the full schema for our example. There are three keys, one for each relationship between the three-way joins attributes (PartyIDRef, RoleIDRef, and PropertyIDRef). The name of each key is some value that identifies this key. The refer attribute should have a value that corresponds to the name value of the key of the element with which you want to establish a relationship with the key. Finally, the xpath attribute of the field particle refers to the attribute of the 3-way join that is the foreign key to the key specified in the refer attribute.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="DEAL">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="LOAN">
          <xs:complexType>
            <xs:attribute name="ID" type="xs:string" use="required"/>
          </xs:complexType>
          <xs:key name="LoanID">
            <xs:selector xpath="."/>
            <xs:field xpath="@ID"/>
          </xs:key>
        </xs:element>
        <xs:element name="PARTY" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="ID" type="xs:string" use="required"/>
          </xs:complexType>
          <xs:unique name="uniquePartyID">
            <xs:selector xpath="."/>
            <xs:field xpath="@ID"/>
          </xs:unique>
        </xs:element>
        <xs:element name="COLLATERAL">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="REAL_PROPERTY">
                <xs:complexType>
```

```
        <xs:attribute name="ID" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ID" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ROLE">
  <xs:complexType>
    <xs:attribute name="ID" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="PARTY_ROLE_PROPERTY_JOIN">
  <xs:complexType>
    <xs:attribute name="PartyRolePropertyJoinID" type="xs:string" use="required"/>
    <xs:attribute name="PartyIDRef" type="xs:string" use="required"/>
    <xs:attribute name="RoleIDRef" type="xs:string" use="required"/>
    <xs:attribute name="PropertyIDRef" type="xs:string" use="required"/>
    <xs:attribute name="AdditionalData" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:key name="PartyID">
  <xs:selector xpath="./mismo:PARTY"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:key name="PropertyID">
  <xs:selector xpath="./mismo:COLLATERAL/mismo:REAL_PROPERTY"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:key name="RoleID">
  <xs:selector xpath="./mismo:ROLE"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:keyref name="fkPartyID" refer="mismo:PartyID">
  <xs:selector xpath="./mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@PartyIDRef"/>
</xs:keyref>
<xs:keyref name="fkRoleID" refer="mismo:RoleID">
  <xs:selector xpath="./mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@RoleIDRef"/>
</xs:keyref>
<xs:keyref name="fkPropertyID" refer="mismo:PropertyID">
  <xs:selector xpath="./mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@PropertyIDRef"/>
</xs:keyref>
</xs:element>
</xs:schema>
```

We'll look at it piece by piece. The root element of this schema, as with all XSD schemas, is the Schema element:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  ...
</xs:schema>
```

The next element is the one that will be the root element for the instance documents, DEAL:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="DEAL">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

The DEAL element is a complexType with five elements: LOAN, PARTY, COLLATERAL, ROLE, and PARTY_ROLE_PROPERTY_JOIN. The LOAN element is not used in the three-way join; it is there as would be any other element, whether or not it's involved in a three-way join.

The actual three-way join is implemented in the DEAL element based on the relationships defined as attributes in the PARTY_ROLE_PROPERTY_JOIN element. Before we look at that, though, let's take a brief look at each of the three elements involved in the three-way join.

Party

```
<xs:element name="PARTY" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="ID" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:unique name="uniquePartyID">
    <xs:selector xpath="."/>
    <xs:field xpath="@ID"/>
  </xs:unique>
</xs:element>
```

The PARTY element consists of an attribute named ID and a unique descriptor. The attribute ID identifies this party. The unique descriptor is there to ensure that the value for the ID attribute is unique in the instance document

Real Property

```
<xs:element name="COLLATERAL">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="REAL_PROPERTY">
        <xs:complexType>
          <xs:attribute name="ID" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

The REAL_PROPERTY element – a child element of COLLATERAL – is specified this way (as a child element of a sibling element to the other two elements involved in the 3-way join) in this example schema to demonstrate that 3-way joins do not need to be composed

solely of sibling elements. The REAL_PROPERTY element consists of an attribute, ID. This is identical functionality to that of the PARTY element.

Role

The ROLE element consists of the same things as REAL_PROPERTY (an ID attribute).

The Three-Way Join

```
<xs:element name="PARTY_ROLE_PROPERTY_JOIN">
  <xs:complexType>
    <xs:attribute name="PartyRolePropertyJoinID" type="xs:string" use="required"/>
    <xs:attribute name="PartyIDRef" type="xs:string" use="required"/>
    <xs:attribute name="RoleIDRef" type="xs:string" use="required"/>
    <xs:attribute name="PropertyIDRef" type="xs:string" use="required"/>
    <xs:attribute name="AdditionalData" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:key name="PartyID">
  <xs:selector xpath="/mismo:PARTY"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:key name="PropertyID">
  <xs:selector xpath="/mismo:COLLATERAL/mismo:REAL_PROPERTY"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:key name="RoleID">
  <xs:selector xpath="/mismo:ROLE"/>
  <xs:field xpath="@ID"/>
</xs:key>
<xs:keyref name="fkPartyID" refer="mismo:PartyID">
  <xs:selector xpath="/mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@PartyIDRef"/>
</xs:keyref>
<xs:keyref name="fkRoleID" refer="mismo:RoleID">
  <xs:selector xpath="/mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@RoleIDRef"/>
</xs:keyref>
<xs:keyref name="fkPropertyID" refer="mismo:PropertyID">
  <xs:selector xpath="/mismo:PARTY_ROLE_PROPERTY_JOIN"/>
  <xs:field xpath="@PropertyIDRef"/>
</xs:keyref>
</xs:element>
</xs:schema>
```

The PARTY_ROLE_PROPERTY_JOIN element is a complexType particle that describes five attributes. The first attribute, PartyRolePropertyJoinID, is used to identify this 3-way join element. The second attribute, PartyIDRef, will hold the value of the party ID that is taking part in this particular 3-way join. The third attribute, RoleIDRef, will hold the value of the role ID that is taking part in the 3-way join. The fourth attribute, PropertyIDRef, will hold the value of the real property ID that is part of this 3-way join. The fifth attribute is some other piece of data for which this whole construction is necessary; that is, this is the data that applies to all three elements together, rather than to any of them separately. Any number of additional attributes can be appended to this complexType definition of the 3-way join element.

Following the closing tag for the PARTY_ROLE_PROPERTY_JOIN element is the end tag closing the complexType particle for the DEAL element. This places the actual join at the DEAL element level. The join is established by the key/keyref relationships for the attributes contained in the PARTY_ROLE_PROPERTY_JOIN element.

The Implementation

An implementation of the 3-way join schema is fairly simple, as shown in the example XML instance document below.

```
<DEAL
  xmlns="http://www.mismo.org/schema/commercial/yyyy/mm/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mismo.org/schema/commercial/yyyy/mm/ file:///c:/doc-1.xsd">
  <LOAN ID="LoanID1" />
  <PARTY ID="PartyID1"/>
  <COLLATERAL ID="CollateralID1">
    <REAL_PROPERTY ID="PropertyID1"/>
  </COLLATERAL>
  <ROLE ID="RoleID1"/>
  <PARTY_ROLE_PROPERTY_JOIN
    PartyRolePropertyJoinID="PartyRolePropertyJoinID1"
    PartyIDRef="PartyID1"
    RoleIDRef="RoleID1"
    PropertyIDRef="PropertyID1"
    AdditionalData="Some other data" />
</DEAL>
```

Guidance to Core Data and Process Areas

Key/keyref declarations should always appear within the lowest common denominator element separately for each key/keyref pair used within the multiple-way join. In the example presented here, the lowest common denominator is the DEAL element (which by coincidence is the root element). Another example might link together a Rent Roll used in an Operating Statement of an Appraisal. In this context, the lowest common denominator would be the THIRD PARTY REPORTS element. The use of the three-way join is fundamentally linked to and reinforces the Commercial Logical Data Model. As the commercial schemas are rolled out, it will be necessary for both the Core Data and Process Area workgroups to provide feedback to Architecture on the market acceptance and ease in implementation of this method. Likewise, the Commercial Architecture Workgroup welcomes questions from the industry on implementing this construct. It is anticipated that, because of the complexity inherent in many commercial loans, there will be a number of unique situations that require communication with the Workgroup in implementing multiple-way joins. Questions should be sent to the workgroup's listserv at mismo_comarch_sg@listman.disa.org.

Namespaces

(Approved by MISMO Commercial Architecture Workgroup October 27, 2004)

Best Practice

The Namespace for the MISMO commercial schemas issued will be in the form `http://www.mismo.org/schema/commercial/yyyy/mm/`. This namespace shall also be the default namespace for the schema, alleviating the need to include a qualifying namespace prefix for all elements in XML Documents. A Namespace designation will only be present in the top level transactional schema files and the supporting schemas (containing simple and complex type definitions) will be “namespaceless” - thus inheriting the top-level namespace..

Technical Overview

The Namespace is a concept to define domain boundaries for schemas issued by various organizations to provide built-in conflict resolution while still allowing elements or attributes of the same name to coexist in the same XML document. In other words, each schema issued creates its own vocabulary and different vocabularies may include the same element name but the element and its contents will be different. Two trading partners may each define an *Account* element which may have different structures, but with proper use of namespace both versions of the *Account* element can exist in the same XML document without any ambiguity or conflict.

Detailed Technical Discussion

The W3C issued as a recommendation Namespaces in XML in January 1999. The W3C recognized that there would be multiple markup vocabularies, that this would pose problems of recognition and collision and that it was necessary for software to be able to recognize the tags and attributes which they are designed to process (even in the face of “collisions occurring when markup intended for some other software package uses the same element type or attribute name).

A Namespace is defined as a collection of names. A Namespace identifier in the form of a Uniform Resource Identifier (URI) can be used to uniquely identify any element within an XML document. This URI does not have to specify the location of the schema. It only serves as a unique identifier although a URL can be used as a URI.

Examples

Declaring a Namespace in a Schema

A typical schema with a namespace will look like the following:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mismo="http://www.mismo.org/schema/commercial/yyyy/mm/"
  targetNamespace="http://www.mismo.org/schema/commercial/yyyy/mm/"
  elementFormDefault="qualified" attributeFormDefault="unqualified"></xs:schema>
```

The components of this declaration are 1) the target namespace and 2) the namespace prefix. The target namespace provides the declaration that all elements -- unless otherwise designated as specifically coming from other namespaces -- are to be validated against schemas coming from that namespace. In this example, there is a target namespace of <http://www.mismo.org/schema/commercial/2005/05/> and a second namespace from the XML schema namespace.

Declaring a Namespace in an Instance Document

In each instance document a namespace can be declared as the default namespace (i.e. without a namespace prefix declaration)

```
<DEAL xmlns="http://www.mismo.org/schema/commercial/2005/05/">  
  <LOAN LoanID="1234" />  
</DEAL>
```

Alternatively, a namespace prefix can be included. If this is done, then each element must be qualified with the namespace prefix as shown in this example.

```
<mismo:DEAL xmlns:mismo="http://www.mismo.org/schema/commercial/2005/05/">  
  <mismo:LOAN LoanID="1234"/>  
</mismo:DEAL>
```

The obvious detriment of this second method is the additional namespace prefix that must be added for every start and end tag. Therefore, it is preferable to declare the predominant namespace as the default as shown above.

It should be noted that although the use of default namespaces in the *instance* document is the preferred method, within the *XML Schema definitions* the commercial namespace is always declared with the prefix "mismo". This is necessary because the XML Schema key-keyref feature requires the use of a namespace prefix in the key references..

MISMO will maintain an RDDDL file at the URL used as the namespace string that will contain a list of pointers to documents and files to help understand the release. Going to that URL with a browser will present the documentation links. Following some of the links may require agreeing to an IPR agreement.

One of the resources documented in the RDDDL file located at the URL of the namespace string will be the location of the schema for any published transaction. If a user's application has the ability to use RDDDL to resolve references, then the validation source schema will be found and used from that location. If, on the other hand, the user is in an environment that is not aware of how to use RDDDL, then browsing to that location will show the user the value.

Example:

Say that the namespace <http://www.mismo.org/schema/commercial/2005/05/> is assigned to a published MISMO commercial standard. Following that link with a browser will show that the location of the schema for validation is at <http://www.mismo.org/schema/commercial/transaction.xsd>

If an instance document is operating in an environment that can discover the schema location using RDDL, only xmlns="<http://www.mismo.org/schema/commercial/2005/05/>" is needed. It would likely be the case that in this scenario, the resolution system would also have a way to look locally for the schema before looking to the www.mismo.org location.

If an instance document is operating in an environment that does not support RDDL, resource resolution users would also need to insert the following link into the instance document:

```
xsi:schemaLocation="http://www.mismo.org/ schema/commercial/transaction.xsd"
```

Namespace Design Patterns

The three design patterns for employing namespace names in a project where there are multiple schema documents:

1. Heterogeneous Namespace Design:
gives each schema a different target namespace
2. Homogeneous Namespace Design:
gives all schemas the same target namespace
3. Chameleon Namespace Design:
gives the “main” schema a target namespace and give no target namespace to the “supporting” schemas (the no-namespace supporting schemas will take-on the target namespace of the main schema, just like a Chameleon)

Based on the building block approach employed throughout MISMO and the division of responsibility between Core Data and the process areas (Origination and Servicing), it is recommended that a Chameleon Namespace be used. As it is intended that Core Data will “own” the declaration of the data types that will be utilized by the process areas in the composition of the transactional schemas, the Chameleon Namespace pattern is generally preferable to the Heterogeneous and the Homogeneous design methods.

Guidance to Core Data and Process Areas

Each transactional schema that will be issued as a MISMO standard shall contain the commercial MISMO namespace declaration as the default namespace. The MISMO commercial namespace of <http://www.mismo.org/schema/commercial/yyyy/mm/> will tentatively be targeted to the planned release date of the first transactions which makes the version 1.0 namespace as <http://www.mismo.org/schema/commercial/2005/05/>.

Architecture will, as part of its review of each schema, ensure proper and consistent use of namespaces.

Naming Convention

(Approved by MISMO Commercial Architecture Workgroup October 27, 2004)

Best Practice

MISMO's commercial specifications will be based on the naming conventions previously established by the MISMO XML Architecture Workgroup. These naming conventions will be used throughout the Commercial Logical Data Dictionary and the commercial schemas.

Technical Overview

The MISMO XML Architecture Workgroup has developed a systematic method for establishing the names of data points used throughout the existing body of MISMO DTD's. These names allow for multiple entities to map their individual data points to the content of the MISMO Logical Data Dictionary. A similarly rigorous naming convention allows the content of the Logical Data Dictionary to directly lead into the establishment of the XML tag names.

The Commercial Architecture Workgroup has decided to implement the same general naming convention methodology to the data points used in commercial specifications. The key benefit resulting from this is that it applies a logical framework for naming data points that has been successfully implemented in an industry similar to the commercial mortgage industry. When one considers that there will be thousands of elements in the Commercial LDD, the application of the MISMO naming convention methodology reduces the complexity in naming and tagging the commercial content.

A secondary benefit is that it allows for reuse of the content between the Residential and Commercial LDD's. Given that there will be crossover points of use for some of the existing residential specifications in the commercial space, there will also be value derived from naming data elements with a consistent methodology (e.g., the potential for duplicate names for the same element is somewhat reduced). But it should be noted that while applying a consistent naming convention reduces the degree of redundancy, it is not a "silver bullet." In this regard, there is tension between the underlying principles of the MISMO naming convention. One principle is that a data element name should stand on its own, independent of context for interpretation. Another is that names should be derived from a data model. The residential and commercial sides of MISMO have implemented different data models (reflecting the underlying differences and complexities between the two industries). Since the MISMO naming convention inherently refers to the relationship of data elements both to each other and to a larger data model, this ultimately means that some data points that have conceptual similarity between the two industries will have different names (creating some degree of redundancy at an abstract, overall MISMO level). Although the actual consequences of this are mitigated by the fact that the implementation of schemas and namespaces for MISMO specifications – particularly for the commercial standards – minimizes the potential for name-based collision of data elements, it increases the level of complexity in developing and deploying specifications across both industries.

Detailed Technical Discussion

The MISMO naming convention applies to both identification of data elements and the construction of XML tag names for those elements. It starts with the establishment of a data element as a single entry in an overall Logical Data Dictionary. Once an entry is created, it is assigned a name that is unambiguous and precise. As a consequence, this name can have a length of several words. In the Residential LDD, the entry also contains:

- a Data Point Definition
- identification of the Process Area(s) that utilize the element
- the XML Container in which the element is used
- specification of the Data Type
- a list of valid values if the Data Type is “Enumerated”
- the XML tag name

and optionally,

- the data source for the data element
- decimal place information
- definitions for the valid values if Data Type is enumerated
- any other information deemed necessary.

The data element name also becomes the XML tag for the data element. Element tag names are represented in capital letters with underscores (`_`) separating words (e.g., `LOAN_APPLICATION`). Attribute names are represented in Upper Camel Case, with word separation noted by the change in case (e.g., `MortgageScoreType`). In cases where there is an exact match between the first words of an attribute name and the container name in which they appear, the MISMO naming convention allows for the replacement of the words with a leading underscore, as noted below for the attributes `MortgageScoreDate` and `MortgageScoreType`:

```
<!ELEMENT MORTGAGE_SCORE EMPTY>  
<!ATTLIST MORTGAGE_SCORE _Date CDATA #IMPLIED>  
<!ATTLIST MORTGAGE_SCORE _Type CDATA #IMPLIED>
```

Because the usage of acronyms and abbreviations abound throughout the industry, MISMO prohibits the use of these in all but a limited number of approved cases. Thus, “LIBOR” is allowed in place of “London Interbank Offered Rate” but “Dt” would be spelled out as “Date.” In cases where numbers appear in a data element name, the following rules apply. Ordinal numbers are spelled out (i.e., “First”). If a cardinal number used in a data element name is less than or equal to twenty (20), it is spelled out. If the number is greater than twenty (20), Arabic numerals are used. The following examples illustrate this: “Three Year Treasury” and “Credit Liability Late 120 Days Count.” Because of the prevalence of forms in the residential industry, an interesting exception to this rule on numbers occurs when a number is included in the form name. Thus, reference to a field on settlement statement required by law at the time of loan closing is allowed as “HUD1 Settlement Date” and not as “HUD One Settlement Date.”

The preceding is a brief, high-level summary of the MISMO naming convention. Complete details on both the establishment of data elements and tag names are contained in the “MISMO XML Design Rules and Guidelines” which is maintained by the MISMO XML Architecture Workgroup and posted on the MISMO web site.

As the Commercial Architecture Workgroup has endorsed the use of these design guidelines, the content of that document will be the authoritative reference document for the Commercial LDD. The remainder of this “*Detailed Technical Discussion*” section will focus on areas where the existing MISMO naming convention is modified to accommodate the needs of the commercial industry. These modifications will be contained in both the “MISMO XML Design Rules and Guidelines” document and this “Best Practices Guide.”

Commercial Logical Data Dictionary

The Commercial Working Group’s first deliverable was an Exposure Draft of its Logical Data Dictionary released in January 2002. This document reflects the work product of the commercial effort at *that point in time*. Since its release, the Commercial Working Group has developed a number of additional deliverables (including a comprehensive Commercial Logical Data Model - LDM) that largely supersede the 2002 document. While the core content of the 2002 document will be preserved in subsequent LDD volumes, its format and construction will change dramatically and take on the appearance of the Residential LDD.

The format of the Residential LDD generally carries over well to the Commercial LDD, with the exception of two key areas: the identification of Process Area and the identification of the XML Container in which the data element is found. Regarding Process Areas, the organizational structure of the Commercial Working Group does not currently mirror that of the residential side where there are numerous workgroups built around Process Areas (i.e., Secondary, Credit, Title, etc.) Functionally, there are Commercial Originations and Commercial Servicing, and the content of the Commercial LDD apply equally to both. Thus, the Commercial LDD will not identify Process Area (until changes in the commercial organization require otherwise).

Regarding identification of the XML Container, the Commercial LDM establishes a set of nested containers that in which the data elements will ultimately be located. The Commercial LDD will explicitly identify these containers and sub-containers, which requires the use of a modified format (for the additional containers) or “Container” than currently used in the Residential LDD.

Data Element Names – Abbreviations and Acronyms

The commercial industry will identify and compile a list of its commonly-used acronyms and append them to the list of approved acronyms contained in the “XML Design Guidelines” document.

Additionally, the Commercial Architecture Workgroup recognizes that while it is commonplace on a business case, the use of abbreviations for data element and XML tag names creates ambiguity and confusion in the case of data transfer. Correspondingly, the use of abbreviations is generally prohibited in the Commercial LDD. Exceptions to this prohibition will be reviewed on a case-by-case basis and any approved abbreviations noted in the list of approved commercial acronyms.

Guidance to Core Data and Process Areas

By its use of detailed names for data elements and XML tags, the MISMO naming convention can adequately meet the needs of the commercial industry. The downside risk is that the verbosity and redundancy of long tag names can add to file size. As the first commercial specifications are implemented, it will be necessary for the Core Data and Process Area workgroups to determine the effect of this on commercial transactions (both in the creation of the schemas by the industry and in the efficiency of data transmission between trading partners).

Element and Attribute Occurrences

(Approved by MISMO Commercial Architecture Workgroup May 5, 2004)

Best Practice

XML schemas shall specify the number of occurrences of elements and attributes in a conforming XML instance document through the use of the minOccurs and maxOccurs attributes (for element nodes) and the use attribute (for attribute nodes).

Technical Overview

When elements and attributes are specified in an XML schema they can have optional attributes that determine the exact range of possible occurrences in a conforming instance XML document (by “conforming instance document” is meant simply an XML document that uses the XML schema document against which to validate its XML document structure). For element nodes, the names of the attributes in an XML schema to do this are “minOccurs” and “maxOccurs”. For attribute nodes the name of the attribute is “use”.

These occurrence constraint attributes are easy to implement in XML schemas, to be demonstrated in examples below. It is important to note that though it is not required to specify these attributes in an XML schema, there are default occurrence constraints on element nodes and attribute nodes. In lieu of explicitly stating these attributes the default behavior will occur automatically. Since the default occurrence behavior for element nodes is both a minimum and a maximum of one occurrence in the conforming instance XML document, and the default occurrence behavior for attribute nodes is that they are optional, one may need explicitly to specify the desired behavior in the schema.

Detailed Technical Discussion

We will take a look first at examples of the use of occurrence constraint declarations for element nodes in XML schemas and then will look at attribute node examples and attribute groups.

Elements - Valid Values: minOccurs and maxOccurs

Any positive integer is a valid value for both minOccurs and maxOccurs. maxOccurs has an additional valid value of “unbounded”, which means that there is no limit on the number of occurrences of this element in the conforming instance XML document. Further, the value of minOccurs is never allowed to exceed the value of maxOccurs.

Element Node Example 1: Element required, max occurrences of one; default behavior

Minimum occurrences: 1

Maximum occurrences: 1

In this example, the default behavior of element node declaration in a schema is demonstrated and explained:

```
<xs:element name="MONEY_TERMS"/>
```

Since neither minOccurs nor maxOccurs is specified, the conforming instance XML document must have one occurrence of an element named “MONEY_TERMS” and may have no more than one occurrence of this element.

Element Node Example 2: Element required once, max occurrences of one

Minimum occurrences: 1

Maximum occurrences: 1

In this example, the value of minOccurs is 1 and the value of maxOccurs is 1. This means that the conforming instance XML document must have one occurrence of an element named “MONEY_TERMS” and may have no more than one occurrence of this element. This happens to be the same specification as the default behavior of element nodes in an XML schema, so it does not matter technically whether the behavior is specified as in Example 1 or as in this current example.

```
<xs:element name="MONEY_TERMS" minOccurs="1" maxOccurs="1" />
```

Element Node Example 3: Element optional, max occurrences of one

Minimum occurrences: 0

Maximum occurrences: 1

In this example, the value of minOccurs is 0 and the value of maxOccurs is 1. This means that the conforming instance XML document does not need to have an occurrence of an element named “MONEY_TERMS”, but if it does exist in the XML document it may have no more than one occurrence of this element.

```
<xs:element name="MONEY_TERMS" minOccurs="0" maxOccurs="1" />
```

Element Node Example 4: Element optional, max occurrences of two

Minimum occurrences: 0

Maximum occurrences: 2

In this example, the value of minOccurs is 0 and the value of maxOccurs is 2. This means that the conforming instance XML document does not need to have an occurrence of an element named “MONEY_TERMS”, but if it does it may have no more than two occurrences of this element.

```
<xs:element name="MONEY_TERMS" minOccurs="0" maxOccurs="2" />
```

Element Node Example 5: Element required twice, max occurrences of three

Minimum occurrences: 2

Maximum occurrences: 3

In this example, the value of minOccurs is 2 and the value of maxOccurs is 3. This means that the conforming instance XML document must have at least two occurrences of an element named “MONEY_TERMS”, but may have no more than three occurrences of this element.

```
<xs:element name="MONEY_TERMS" minOccurs="2" maxOccurs="3" />
```

Element Node Example 6: Element optional, infinite max occurrences

Minimum occurrences: 0

Maximum occurrences: unbounded (no limit)

In this example, the value of minOccurs is 0 and the value of maxOccurs is unbounded. This means that the conforming instance XML document need not have an occurrence of an element named “MONEY_TERMS”, and may have as many occurrences as desired.

```
<xs:element name="MONEY_TERMS" minOccurs="0" maxOccurs="unbounded" />
```

Attributes - Valid Values: Use attribute

Valid values for the attribute node’s occurrence constraint, “use”, are:

- Attribute is Required – ‘required’
- Attribute is Optional – ‘optional’
- Attribute is Prohibited – ‘prohibited’

When ‘required’ is specified for the ‘use’ attribute, the attribute node whose occurrence is being constrained is required to appear in a conforming instance XML document, though never more than once for any one element (which is true of all attributes). If the value is ‘optional’, it is simply an explicit declaration of the default behavior of attributes that they need not appear in a conforming instance XML document.

The value ‘prohibited’ is applicable only in the case of extension by restriction. The use attribute can only be changed in a derived type to restrict the base type by specifying tighter occurrence constraints. That is, if an attribute is optional (the default) in the base type, it can be restricted to be either required or prohibited in the derived type. In any other situation, use=“prohibited” is ignored by the validating parser.

NOTE:

The applicability of the prohibited value in the MISMO standard is subject to the to the Complex Type and Inheritance recommendation that has been deferred to version 1.1 of then guide.

Attribute Node Example 1: Attribute required

Minimum occurrences: 1

Maximum occurrences: 1

In this example, the value of ‘use’ is ‘required’. This means that the conforming instance XML document must have an attribute named “LoanName”, but -- as is true for any attribute -- is not allowed to exist more than once in any one element.

```
<xs:attribute name="LoanName" use="required" />
```

Attribute Node Example 2: Attribute optional

Minimum occurrences: 0

Maximum occurrences: 1

In this example, the value of 'use' is 'optional'. This means that the conforming instance XML document need not have an attribute named "LoanName" but may have one instance.

```
<xs:attribute name="LoanName" use="optional" />
```

Attribute Node Example 3: Attribute optional (default behavior)

Minimum occurrences: 0

Maximum occurrences: 1

In this example, the 'use' attribute in the XML schema is not specified. As the default behavior, this means that the conforming instance XML document need not have an attribute named "LoanName" but may have one instance.

```
<xs:attribute name="LoanName" />
```

Attribute Groups and Inheritance

Occurrence constraints for attributes that are part of an attribute group are defined at the time that the attribute group itself is defined. Look, for example, at the attribute group below:

```
<xs:attributeGroup name="BorrowerGroup">  
  <xs:attribute name="Attr1" type="xs:string" use="optional"/>  
  <xs:attribute name="Attr2" type="xs:string" use="optional"/>  
  <xs:attribute name="Attr3" type="xs:string" use="optional"/>  
  <xs:attribute name="Attr4" type="xs:string" use="required"/>  
</xs:attributeGroup>
```

When this attribute group is referenced there normally is no opportunity to change the occurrence constraints of the various attributes that have been defined in that attribute group. For example, following is a typical way to reference an attribute group:

```
<xs:complexType name="LenderType">  
  <xs:attributeGroup ref="mismo:BorrowerGroup"/>  
</xs:complexType>
```

By this reference, any element of type LenderType optionally can use Attr1, Attr2, and Attr3 attributes and must use the Attr4 attribute. In the complexType definition above there is no way to change the occurrence constraint values for the attributes contained within the attribute group that LenderType references.

Summary

There are both advantages and disadvantages to the use of occurrence constraints, particularly within the context of an industry like commercial mortgage where there are many perspectives to consider, and particularly concerning the requirement that an element or attribute exist in an instance document.

Advantages

- **Validation:** Without the use of a standard document among organizations' systems that can enforce occurrence integrity, this integrity would be left up to each organization's

development group to take care of separately. XML schemas have occurrence constraints like minOccurs, maxOccurs and the use attribute so that there is a standard way for systems to communicate without forcing each organization to come up with a solution to implement validation techniques.

- **Control:** Since element nodes in an XML schema have a default behavior of being required (but not more than once) in the conforming instance XML document, without the use of minOccurs and maxOccurs one would be forced always to include these element nodes in the conforming instance XML document, but never be able to include them multiple times. The rules for defining occurrence constraints in element nodes give complete control over the occurrence specification.

Disadvantages

- **Garbage Data:** When elements or attributes are required in a schema, but when a system has no available value to fill the element or attribute, the system (or user of the system) would be forced to give the element or attribute some value simply so that the conforming instance XML document validates against the schema. This is problematic in two ways: one, this puts an additional burden on the system or user, and two, the consumer of the XML document would not know if the value is intended to be understood as a “real” value or whether it is simply some “throw-away” value. Having said that, as discussed above, elements have a default behavior of being required; this particular disadvantage, then, applies whether or not the occurrence constraints are specified explicitly or not. Attributes, though, have a potential vulnerability to this disadvantage, since their default Use state is optional.
- **Appearance of Exclusivity:** Making certain items required may cause potential implementers to feel that the standard is too restrictive and possibly viewed as proprietary to meet the needs of a few large companies. This is the main reason that Residential MISMO is not using required elements/attributes. Once again it may be good to point out that simply the inclusion of an element node in a schema, without explicitly stating the values for minOccurs and maxOccurs, implies that the element must be included in conforming instance XML documents.

Guidance to Core Data and Process Areas

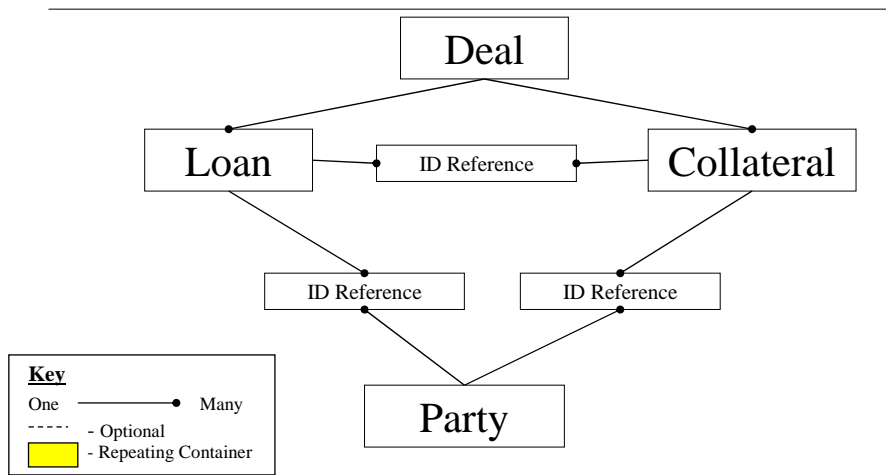
The use of required elements and attributes in a schema should be limited to those elements or attributes without which the loan would make little or no sense. For all elements that are not required, the minOccurs attribute should be set to 0. Attribute nodes, on the other hand, are optional by default; it is only in the case where an attribute is deemed to be required that the use attribute would need to be specified (with a value of “required”).

Commercial Logical Data Model

(Finalized by MISMO Commercial Working Group September 30, 2004)

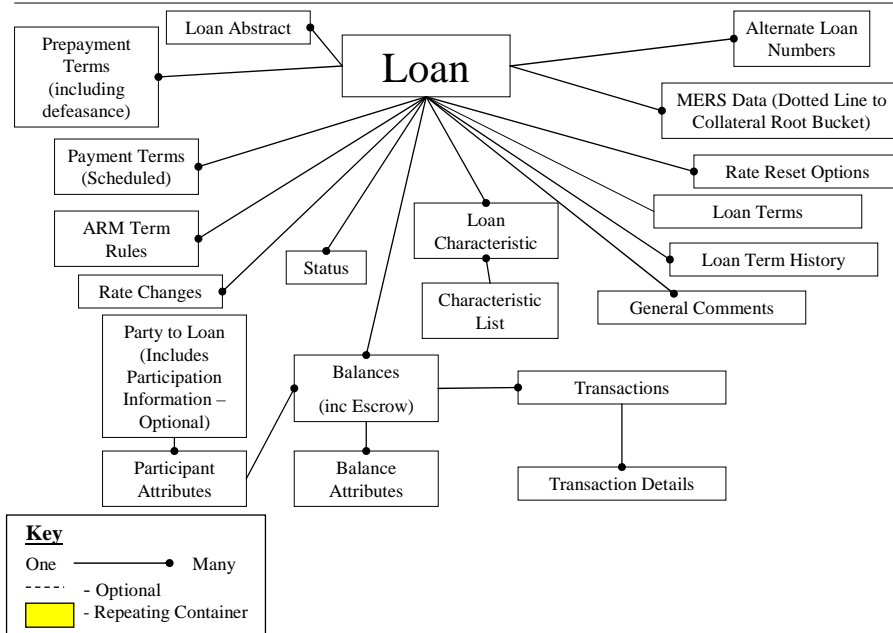
Top Level LDM for Commercial Mortgages

NOTE 1: The Inclusion of a Deal container allows multiple Loans to be associated with the same property or borrower (i.e., a first-lien and a mezzanine position). It also allows the passing of Collateral Data without corresponding Loan Data (e.g., for REO).



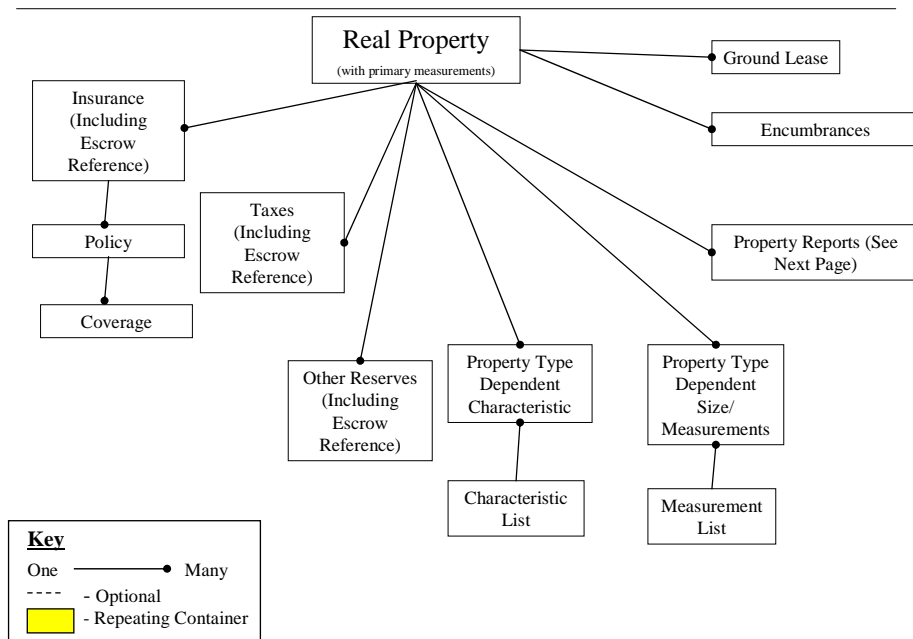
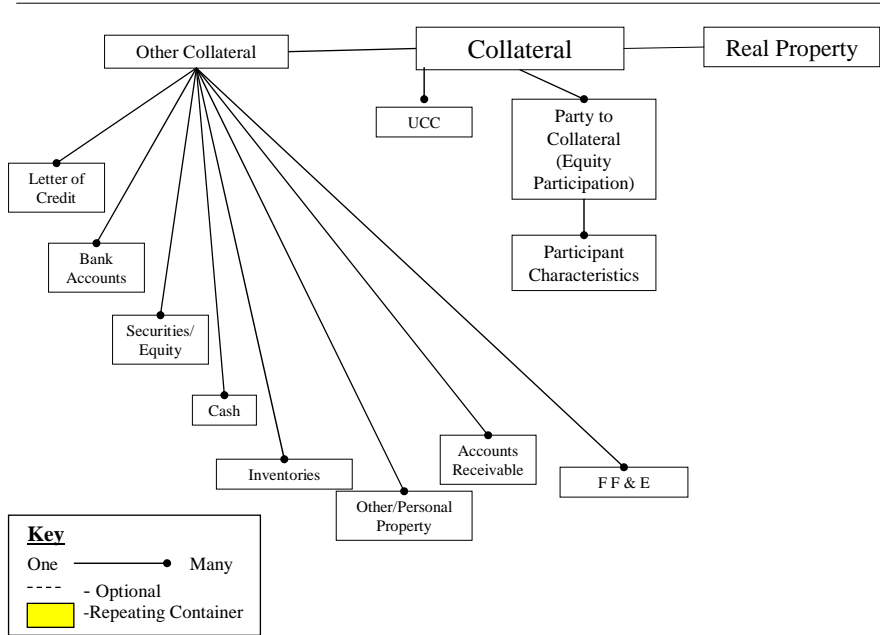
Loan Containers

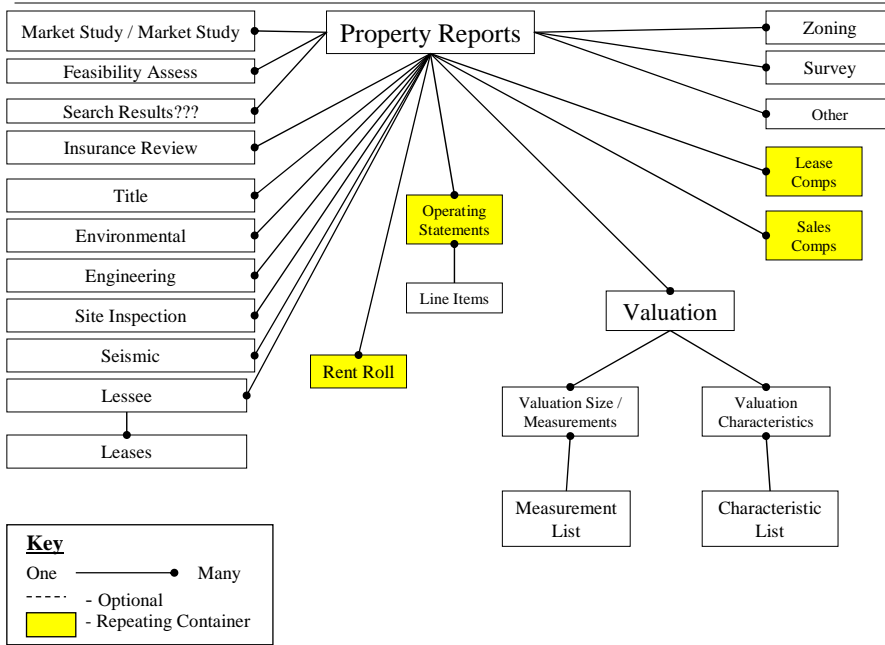
(NOTE: Party includes borrower, servicer, owners, guarantors, etc & will be tied to other containers.)



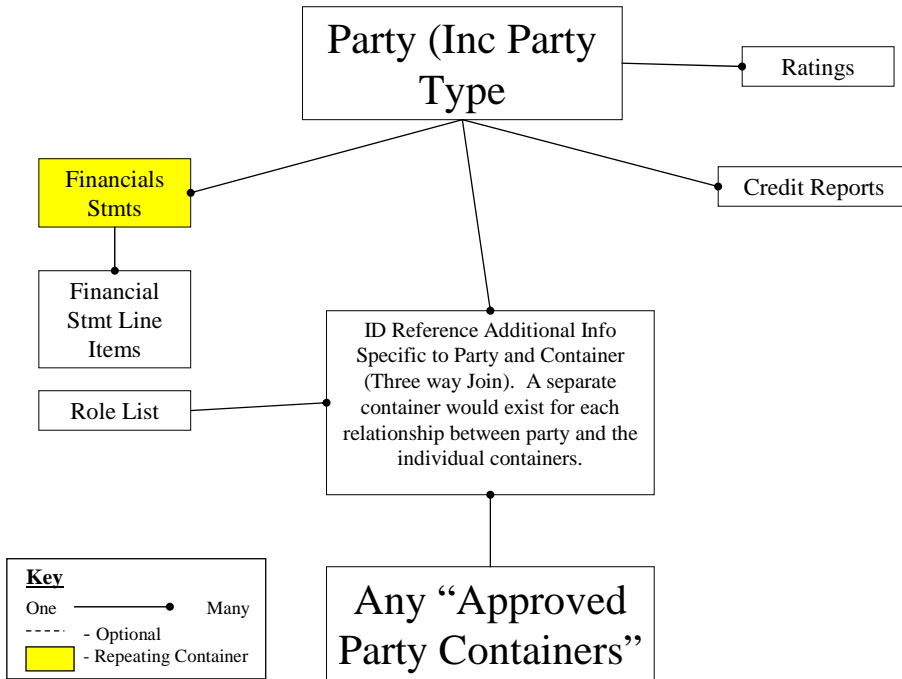
Collateral Containers

(NOTE: Party can be tied to each container)





Party Containers



Proposed “Approved Party Containers”

- Party to Party
- Party to Loan
- Party to Collateral
- Party to Participation – This is Participation
- Party to Property Reports (inc. Party/Rent Roll Information)
- Party to Other Collateral (*TBD on how Other Collateral Works)
- Party to Tax
- Party to Insurance
- Party to Lessee (Not yet discussed)

Other Party Related Information

- Within each ID Reference table, we would have the fields for Start Date and End Date so that that the relationship of the Party to the other containers could be expired and NOT deleted.
- Additional information that is in the context of a particular container and a party will be contained in the appropriate ID Reference table.

Repeating Container Information

Container Name	Required Info.	Possible Other Locations (Not Required)
Rent Roll	Property	Could be contained in a Valuation, Referenced in a Lease
Financial Statements	Property or Party	Could be contained in a Valuation
Lease Comps	Property	Could be contained in a Valuation
Rent Comps	Property	Could be contained in a Valuation
Entity	None	Could be contained in as part of any of the "Approved Party Containers"

Specific Deal Transmissions

- All Deal Transmissions
 - Require a Deal
- Participations
 - Information transmitted in the Party to Loan bucket.
- Equity Based Deals or REO
 - A deal that utilizes the Property container without a loan or reference to a loan.

Mezzanine Debt

- In the Loan container identify the loan as type of “Mezzanine.”
- Use the Party to Loan container to identify the relationship (Lien/Interest) on the Party.
- Use the Party to Collateral container to identify the Collateral as an “Asset” of the Party.
- Utilizes the Property container to move the property information.

High Level Master LDM for Servicing Transfers
(Approved as Draft by Servicing Workgroup on 1/24/03)

