Meta Data Coalition

Open Information Model
XML Encoding

Version 1.0

Review Draft

September, 1999

TABLE OF CONTENTS

1 Overview

The Meta Data Coalition (MDC) Open Information Model (OIM) is a vendor-neutral and technology-independent specification of core meta data types found in operational and data warehousing environments.

This document describes a recommended format for exchanging instances of the OIM through the use of Extensible Markup Language (XML). It is assumed that the reader is familiar with the concepts represented by the OIM. For an overview of the OIM, please refer to the specification available on-line at http://www.mdcinfo.com.

2 Background

This document describes a set of rules that govern the encoding of metadata objects described by OIM in XML. The XML encoding of OIM types enables the interchange of metadata between heterogeneous repositories. It is also a wire-encoding format to use between repositories and tools. The encoding format defined in this specification is completely driven by the abstract model. As the names of the element and attribute tags used in the representation are derived from the model, documents can be generated and parsed automatically by any implementation of OIM, regardless of technology.

2.1 XML Standards and Drafts

This document is based on Extensible Makeup Language (XML) 1.0 as defined by the W3C. REC-xml-19980210 is the XML specification that was used in the preparation of this document.

XML Namespaces provide a simple method to qualify names in XML documents. The implementation of namespaces in this document is based on the W3C recommendation, Namespaces in XML, REC-xml-names-19990114.

2.2 XML Document Type Definition (DTD)

Accompanying this specification is a set of XML Document Type Definitions (DTDs), which together form a grammar to express the structure of XML instances. DTDs are currently the only approved mechanism to describe the structure of XML documents. In its current form, DTD is not expressive enough to completely cover the semantics of OIM. A correct interpretation of an XML document is therefore only possible based on the OIM specification. However, DTDs have been supplied to make understanding of the XML documents easier and to help with the development of XML import/export functionality based on this encoding format.

3 XML Encoding Definition

XML encodes information as content enclosed in nested begin/end tag elements and name/value pair attributes on these elements. The XML encoding format defined in this document is completely based on this encoding rule.

XML provides the following basic concepts to encode information:

- Character Set – The encoding used for all information in an XML document.
- Top-level Element – The element that encapsulates all transfer information in an XML document.
- Elements – Begin/end tag pairs and the content (transfer information or subordinate element structure) encapsulated between them.
- Attributes – Name/value pairs contained in the begin tag of an element that represent meta-information about the element.
- Namespaces – Scope for the tags of elements to make them unique in a XML document.
- Nested Lists – Ordered or unordered sets of elements that can be used to represent hierarchies of elements.
- Element References – Connections between elements to represent network structures of elements.
- Representation of data types such as strings, dates, etc.

3.1 Character Set and Data Types

The XML encoding proposed in this document relies on the XML character set handling based on Unicode. See the Extensible Markup Language (XML) 1.0 specification document for more information.

Values appear as attribute-tagged values. They are represented using the following rules:

Data type	Encoding
String	Any occurrence of & must be replaced by & Any occurrence of < must be replaced by < Any occurrence of > must be replaced by > Any occurrence of " (double quote) must be replaced by &quote;
Date	Must follow the ISO8601[1] format
Numbers	Punctuation must use US English rules (i.e. period as a decimal separator). Can include exponents
Boolean	0 = False 1 = True
Blob	Use MIME Base64[2] encoding

3.2 Top-level element

XML requires a top-level element (begin/end tag) that encapsulates all information contained in an XML document. DTDs defined or referenced in the document apply to the content of the top-level element.

The OIM to XML mapping defines a Transfer element as the top-level structure. This element encapsulates all structured information that is exchanged between repositories. Additional features of the Transfer element are that it can be nested and that it maintains administrative information, e.g. what exporter generated the data, version, etc.

---

[1] See http://www.cl.cam.ac.uk/~mgk25/iso-time.html for descriptions of the ISO8601 format

[2] See Internet RFC1521 (for example at http://ds.internic.net/rfc/rfc1521.txt) for a definition of the Base64 format.

Example

```
<?xml version="1.0"?>
<oim:Transfer version="1.0"
      xmlns:oim="http://www.mdcinfo.com/oim/oim.dtd">
      <oim:TransferHeader
            Exporter="MSRXML"
            ExporterVersion="2.0"
            TransferDateTime="19980804T08:15:00"
      >
            . . . user-defined information . . .
      </oim:TransferHeader>
      . . . objects . . .
</oim:Transfer>
```

All structures defined in the remainder of this section are valid only within the begin (<oim:Transfer>) and end (</oim:Transfer>) tags of the Transfer element. The TransferHeader element is used to contain information about the component that generated the transfer.

3.3 Elements

Elements in XML are represented by an opening tag and closing tag pair, which encapsulate the content of the element. The content can be either a structure of sub-elements or an unstructured data representation. The following table shows how the different UML types used to model the Open Information Model are mapped into XML:

UML Element	XML representation
Information Model	No mapping. See Namespaces section for more detail.
Concrete Class	*<class_name>…</class_name>*
Attribute	Included as an XML attribute on an XML element, e.g *attribute=value*
Association	*<association_name>…</association_name>*

The tag identifies the type of an element. Additional meta-information about the element can be represented by predefined attributes. The following attributes are currently predefined:

Attribute Name	Defined for:	Mandatory/ Optional	Description
Id	Object, Association	Optional	Transfer ID, which is used to uniquely identify an element in an XML document. The id has no meaning outside of a transfer. The Id is mandatory on objects, but optional on object references.
objid	Object, Association	Optional	Unique identifier of an element in the source or target repository.
seqno	Object	Optional	Number that defines the position of an element in a collection.
href	Objects	Optional	Hyperlink mechanism to reference objects.
label	Objects	Optional	The name of object within the encapsulating association
supertype	Object	Optional	Used by extensions to indicate the OIM type that can be used for importing an object.

The OIM to XML mapping separates the transfer ID and object ID and, in addition, treats the object ID as an attribute of the element. This XML encoding is designed to enable the interchange of objects between heterogeneous repositories. There is no common format for object identifiers and furthermore there is no common agreement how to implement object identity (name based, GUID, disk pointer, etc.).

To provide a generic solution, an ID is defined that uniquely identifies an object within a transfer; i.e. an ID can serve as the target of a reference in the transfer. The structure of the ID is unspecified, except that it must be unique in a transfer and must contain an underscore as first character. Examples of valid transfer IDs are a running number ("_007") or the name of an object ("_Invoice007").

Note that object identity is necessary to allow a meaningful synchronization of objects between repositories. In a heterogeneous environment, this requires the XML encoding to maintain a cross-reference between the globally unique identifiers of objects maintained by different repository products. To exchange object IDs as attributes of objects requires that exchanging repositories agree on the semantics of the exchange mechanism. The attribute *objid* has been included in the encoding format to simplify this process. The first source of a transfer, if necessary, can generate the object ID. Each successive transfer step must maintain the whole object ID and pass it on.

3.4 Namespaces

In the OIM, classes and associations share the same namespace for a single sub-model. What that means is that there cannot be more than one class and/or association within the same sub-model that have the same name. The following shows the basic structure of the OIM namespace hierarchy:

Level 1	Level 2	Description
Sub-model		Corresponds to a sub-model in OIM.
	Class	Class name and the associated attributes.
	Association	A collection of nested or linked classes.

The OIM to XML mapping combines XML Namespaces and a naming convention to provide the following solution.

<	Namespace		Name	>
<	Sub-Model Prefix	:	Class Name	>
<	Sub-Model Prefix	:	Association Name	>

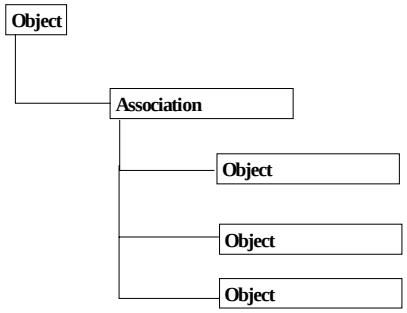
For example, <x:y> is a element tag for an object of class y in sub-model x. Note that because attributes are represented as XML attributes and XML attributes are scoped as part of the element, the attribute names only need to be unique within the class, not the whole sub-model.

If more than one attribute in the inheritance chain of the class has the same name, then both attributes have their names expanded to **ClassName.AttributeName**. If the class name is not unique, then it is prefixed with the namespace.

3.5 Nested Lists

XML represents information as nested lists of elements or references to elements. Lists can be either ordered or unordered and the occurrence of element types might be optional or mandatory.

The following diagram shows the representation of the OIM "class has associations" and "associations contain objects" in XML:



Example:

```
<object attribute="…">
     <association>
        <object label="C" name="Lisa" seqno="1">
        …
        </object>
        <object label="A" name="John" seqno="2">
          …
        </object>
        <object label="B" name="Tom" seqno="3">
          …
        </object>
     </association>
     ...
</object>
```

Object elements contain lists of association elements, which in turn contain lists of object elements.

3.6 Element References

Nested lists of XML elements enable the representation of hierarchical structures of objects. References are used to link objects into a general network of associations. The XML hyperlink mechanism is used to reference objects defined internal to a transfer. An internal object is simply referenced by its transfer id.

An object reference is represented by the *href* attribute of the element tag:

<object_type_name href="#_123"/ >

The object reference indicates the type of the referred-to object; to learn the object type, you need not navigate the object reference to the target object.

3.7 Extensibility

The Open Information Model can be extended with vendor-specific meta data types. New classes, attributes, and associations are added using the UML representation of the OIM and should be grouped into a vendor or tool-specific package. New elements may be either created from scratch or based on existing OIM types using specialization (inheritance). A vendor may choose to publish the model extensions in order to share the meta data with other vendors, or treat the extension as tool specific (private).

Using the OIM to XML mapping rules described in this document, an XML DTD for the model extension can be created from its UML representation. However, the XML DTD does not provide enough information for other vendors to interpret the model. This is a limitation of the current XML standard with DTD as schema description language. DTDs do not capture type inheritance and other sophisticated modeling structures. The W3C, as XML standard body, has an effort under way to standardize a new schema definition language called XML Schema.

Until the XML Schema specification becomes available, the OIM XML encoding format will support the use of the optional *supertype* attribute. This attribute is used to define which OIM type a new meta data type specializes. In the case where multiple OIM types are specialized, it is responsibility of the exporting tool to choose one of the types.

The following example shows an instance of a new meta data type extending the Table class in the Database Schema Model:

```
<Ext:MyTable supertype="DBM:Table" name="xxx" size="yyy" myVal="123"/>
```

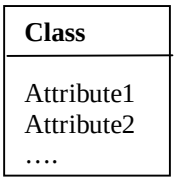
An importer can decode the element structure even if the new subtype is unknown. It simply uses the schema of the known OIM type specified by the *supertype* attribute. Note that the attribute must contain a fully qualified class name (including namespace). It is also necessary to resolve attribute name conflicts using the same rules as described in the following sections.

4 OIM to XML Mapping

This section provides a set of basic diagrams that show the mapping of the core concepts Class, Attribute, and Association as well as class inheritance from OIM into XML. An UML diagram representing the OIM concepts is provided with the XML encoding.

4.1 Classes and Attributes

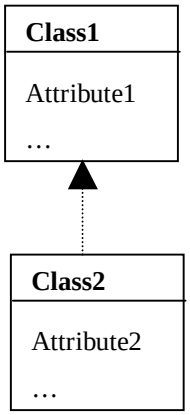
The following example shows how the attributes of an OIM class are mapped into XML.




```
<Class Attribute1="…" Attribute2="…">
        …
</Class>
```

4.2 Classes and Single Inheritance

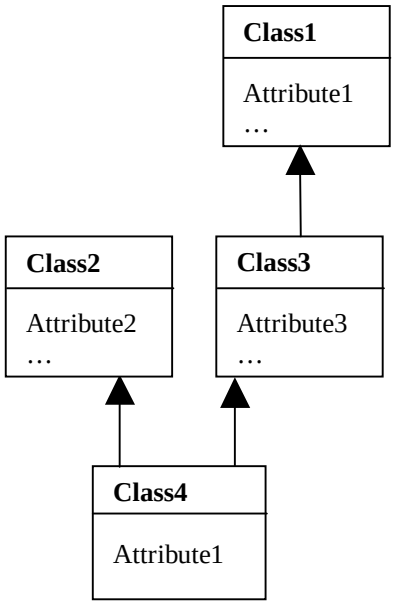
The following example shows how attributes and inherited attributes of a class are mapped into XML.




```
<Class2 Attribute1="…" Attribute2="…">
        …
</Class2>
```

4.3 Classes and Multiple Inheritance

A class can inherit attributes from multiple other classes. The following example shows how such a class is represented.

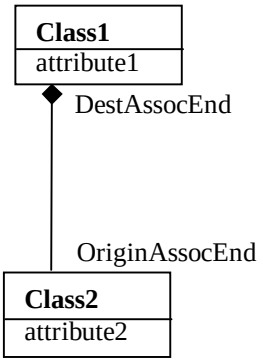



```
<class4 id="_123"
        Class4.Attribute1="…"
        Class1.Attribute1="…"
        Attribute2="…"
        Attribute3="…">
              …
</class4>
```

Because there is a naming conflict between the two attributes called Attribute1, the name of the class they are defined on is added as prefix.

4.4 Associations

The next example shows how OIM associations are encoded in XML.




```
<class1 id="_1" attrib1="…">
  <Class1OriginAssocEnd id="_3> <!-- assoc starts -->
    <class2 id="_2" seqno="1" label="A"
            name="Alpha" attribute2="…"/>
    <class2 id="_3" seqno="2" label="B"
            name="Beta" attribute2="…"/>
  </Class1OriginAssocEnd>  <!-- association ends -->
</class1>
```

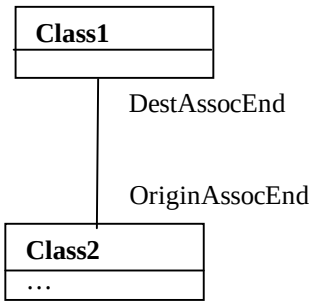
If an association name is not specified, the name is generated using the following rule:

OriginClassName + OriginAssociationEndName

Given this rule, the association in the above example is named **<Class1OriginAssocEnd>**.

4.5 Object References

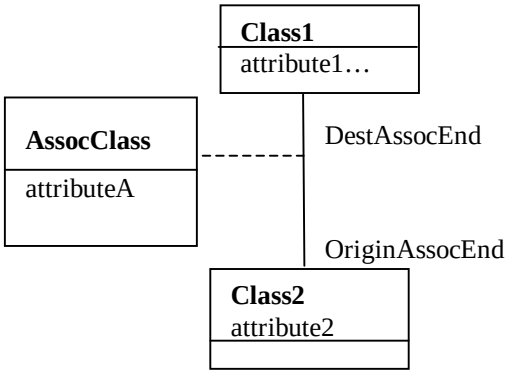
The following shows an association structure in which the destination object has already been defined and therefore needs to be referenced.




```
<class2 id="_2">
      …
</class2>
<class1 id="_1">
      <Class1OriginAssocEnd>
            <class2 href="#_2"/>
            …
      </Class1OriginAssocEnd>
      …
</class1>
```

4.6 Association Classes (Many to Many)

The following shows how an association class is represented in XML:

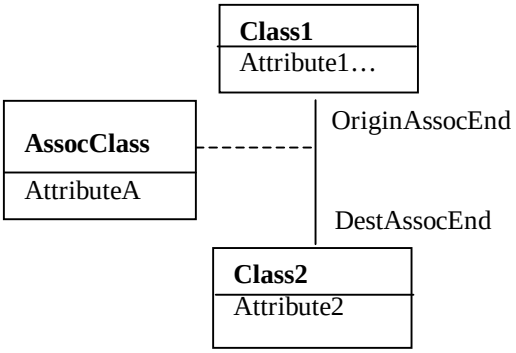



```
<class1 id="_2">
      <class1OriginAssocEnd>
            <AssocClass attributeA="…">
      <AssocClassDestAssocEnd>
                        <Class2 />
      </AssocClassDestAssocEnd >
            </AssocClass>
      </class1OriginAssocEnd>
</class1>
```

This effectively changes the association class into a junction class between the other two classes and uses the association name generation rule to establish the two association names.

4.7 Association Classes (One to Many or One to One)

The following shows how a one-to-many or one-to-one association class is represented in XML:




```
<class1 id="_2" attribute1="…">
      <class1OriginAssocEnd>
            <class2 attributeA="…"
                    attribute2="…"/>
            …
      </class1OriginAssocEnd>
</class1>
```

This mapping represents all the attributes of the association class as attributes on the destination class.

Appendix A – Sample Encoding

```
<?xml version="1.0" ?>
  <oim:Transfer xmlns:oim="http://www.mdcinfo.com/oim/oim.dtd"
                xmlns:dbm="http://www.mdcinfo.com/oim/dbm.dtd">
    <dbm:Catalog id="_1" name="sales" comments="Sample catalog">
      <dbm:CatalogSchemas>
        <dbm:Schema id="_2" name="dbo">
          <dbm:SchemaTables>
            <dbm:Table id="_3" name="Customer">
              <dbm:ColumnSetColumns>
                <dbm:Column id="_6" name="CustomerID" IsNullable="0" />
                <dbm:Column id="_7" name="Name" IsNullable="0" />
                <dbm:Column id="_8" name="Address" IsNullable="1" />
                <dbm:Column id="_9" name="Phone" IsNullable="1" />
              </dbm:ColumnSetColumns>
            </dbm:Table>
            <dbm:Table id="_4" name="Order" EstimatedRows="10000">
              <dbm:ColumnSetColumns>
                <dbm:Column id="_10" name="CustomerID" IsNullable="0" />
                <dbm:Column id="_11" name="OrderID" IsNullable="0" />
                <dbm:Column id="_12" name="Date" IsNullable="1" />
              </dbm:ColumnSetColumns>
            </dbm:Table>
            <dbm:Table id="_5" name="OrderItem" EstimatedRows="100000">
              <dbm:ColumnSetColumns>
                <dbm:Column id="_13" name="CustomerID" IsNullable="0" />
                <dbm:Column id="_14" name="OrderID" IsNullable="0" />
                <dbm:Column id="_15" name="LineNo" IsNullable="0" />
                <dbm:Column id="_16" name="Description" IsNullable="1" />
                <dbm:Column id="_17" name="Quantity" IsNullable="0" />
                <dbm:Column id="_18" name="UnitPrice" IsNullable="0" />
              </dbm:ColumnSetColumns>
              <dbm:TableUniqueKeys>
                <dbm:UniqueKey id="_19" name="PK_OrderItem" IsPrimary="1" />
                  <dbm:KeyColumns>
                    <dbm:Column href="_14" />
                    <dbm:Column href="_15" />
                  </dbm:KeyColumns>
                </dbm:UniqueKey>
              </dbm:TableUniqueKeys>
            </dbm:Table>
          </dbm:SchemaTables>
        </dbm:Schema>
      </dbm:CatalogSchemas>
    </dbm:Catalog>
  </oim:Transfer>
```

Appendix B – EBNF Representation

The following defines the grammar of the OIM XML encoding in EBNF:

```
xmlHdr          ::=     '<?xml version="1.0">'

oimDoc          ::=     xmlHdr S Transfer

Transfer        ::=     '<oim:Transfer' [S 'version="1.0"'] NameSpaceDecl '>' S
                                [TransferHeader]
                                (Object | Transfer)*
                        '</oim:Transfer>'

TransferHeader  ::=     '<oim:TransferHeader'
                                ['Exporter="' ExporterName '"']
                                ['ExporterVersion="' ExporterVersion '"']
                                ['TransferDateTime="' CurrentDate '"']
                        '/>'

oimNameSpace    ::=     'xmlns:oim="http://www.mdcinfo.com/oim/oim.dtd"'

oimPrefix       ::=     'oim:'

NameSpaceDecl   ::=     oimNameSpace (S ModelSpaceDecl)*

ModelSpaceDecl  ::=     'xmlns:' modelAbbr '="http://www.mdcinfo.com/oim/' nsPrefix '.dtd"'

nsPrefix        ::=     modelAbbr (of the sub-model that the class is defined in)

objTransID      ::=     '_' uniquifier (where uniquifier is a running number)

objID           ::=     unique identifier for the element (repository dependent)

seqno           ::=     sequence number within an association

label           ::=     name of object within the association

object          ::=     '<' nsPrefix ':' elementName S
                                'oim:id="' objTransID '"'
                                [S 'oim:objid="' objID '"']
                                [S 'oim:seqno="' seqno  '"']
                                [S 'oim:label="' label '"']
                                [(S Attribute)*]
                        '>'
                                [(S Association)*]
                        '</' nsPrefix ':' elementName '>'

Attribute       ::=     [[nsPrefix ':'] ClassName '.'] AttributeName S? '="' S?
                                attributeValue S? '"'

Association     ::=     '<' [nsPrefix ':'] AssociationName '>' S?
                                (Object S)*
                        '</' [nsPrefix ':'] AssociationName '>'
```

Appendix C – Namespaces in the OIM

Using XML Namespaces, each sub-model of the OIM encoding defines a separate namespace for its XML tags; i.e. each sub-model is described by an individual DTD. Sub-models depend on each other and form a well-defined (acyclic) dependency graph. MDC OIM 1.0 has the following sub-models:

OIM Submodel	Namespace Identifier
Analysis and Design Model	
Unified Modeling Language	uml
UML Extensions	umx
Common Data Types	dtm
Generic Elements	gen
Object and Component Description Model	
Component Descriptions	cde
Database and Data Warehousing	
Database Schema Elements	dbm
Data Transformation Elements	tfm
OLAP Schema Elements	olp
Record Oriented Legacy Databases	rec
Knowledge Management Model	
Semantic Definition Elements	sim

The XML namespaces respect the extensibility mechanism of OIM. Users are able to add sub-models with new elements without causing name conflicts with existing sub-models or future extensions.

Appendix D – DTD for the OIM Namespace

```
<!-- ____________________________________________________________ -->
<!--                          XML Encoding                        -->
<!--                for the Open Information Model                -->
<!-- ____________________________________________________________ -->

<!-- ____________________________________________________________ -->
<!-- Transfer                                                     -->
<!-- ____________________________________________________________ -->
<!-- A transfer is a unit of exchange in OIM. Transfers might be  -->
<!-- nested.                                                      -->
<!ELEMENT Transfer ( TransferHeader?, ( ANY | Transfer )* ) >
<!ATTLIST Transfer
                 version CDATA #FIXED "1.0"
>
<!-- ____________________________________________________________ -->
<!-- TransferHeader                                               -->
<!-- ____________________________________________________________ -->
<!-- A transfer header allows to specify all necessary information -->
<!-- to define the origin of a transfer in a structured way.      -->
<!-- Exporter         Name of software that generated the transfer -->
<!-- ExporterVersion  Version of software that generated transfer  -->
<!-- TransferDateTime  Date and time that the transfer was created -->
<!ELEMENT TransferHeader (ANY)>
<!ATTLIST TransferHeader
                 Exporter CDATA #IMPLIED
                 ExporterVersion CDATA #IMPLIED
                 TransferDateTime CDATA #IMPLIED
>
<!-- ____________________________________________________________ -->
<!-- Classes                                                      -->
<!-- ____________________________________________________________ -->
<!-- Classes are output as XML elements. They should all have id, -->
<!-- objid, href and sequence number as predefined XML attributes. -->
<!-- Unfortunately the DTD grammar is not capable to specify this -->
<!-- so these attributes are shown here as an example. The oim:   -->
<!-- namespace qualifier for the predefined attribute is only     -->
<!-- included when one of the predefined attribute has a naming   -->
<!-- conflict with the attributes on the class                    -->
<!-- <!ATTLIST typename                                           -->
<!--                     [oim:]id ID #REQUIRED                    -->
<!--                     [oim:]objid CDATA #IMPLIED               -->
<!--                     [oim:]href CDATA #IMPLIED                -->
<!--                     [oim:]seqno CDATA #IMPLIED               -->
<!--                     [oim:]label CDATA #IMPLIED               -->
<!--                     [oim:]supertype CDATA #IMPLIED           -->
<!--                                                              -->
<!-- End of DTD _________________________________________________ -->
```