IBM’s Model-Based XML Architecture

Table of Contents

Notices

This information was developed for products and services offered in Europe. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

IBM grants limited permission to licensees to make hardcopy or other reproductions of any machine-readable documentation, provided that each such reproduction shall carry the IBM copyright notices and that use of the reproduction shall be governed by the terms and conditions specified by IBM in the license agreement. Any reproduction or use beyond the limited permission granted herein shall be a breach of the license agreement and an infringement of the applicable copyrights.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The product described in this document and all licensed material available for it are provided by IBM under terms of the IMCL or IMSL agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information on softcopy, the photographs and colour illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or service marks of the IBM Corporation in the United States or other countries or both:

IBM
Insurance Application Architecture
IAA

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About this Book

This document describes IBM's Model-Based XML Architecture (MB-XML), with examples of its use.

Who Should Read this Book?

This book is intended for people who want to understand the MB-XML message architecture, with a view to developing and implementing XML-based messaging designs. MB-XML is a messaging architecture for inter-application communications applicable to any industry

The reader is expected to be familiar with XML and XML Document Type Definitions (DTDs). Knowledge of object modeling notation will assist in understanding the examples contained within this book.

What this Book Contains

The document provides an overview of the architecture and approach adopted by MB-XML, and contains sections on:

- Introduction and Positioning
- Message Architecture
- Message Design

The document also contains a Glossary, a Bibliography, and the following appendices:

- Appendix A: MB-XML and the underlying UML model
- Appendix B: MB-XML Generation from an Interface Design Model in Rational Rose (RR)
- Appendix C: Data Types
- Appendix D: Change History of this Document

Introduction and Positioning

MB-XML is a messaging architecture for inter-application communications applicable to any industry. This section explains why MB-XML was developed, what it is, and where and how it can be used.

Introduction

The MB-XML architecture defines the *architectural elements* that can appear in MB-XML messages, as well as the way they relate to each other. These architectural elements describe the organization of data and commands into messages, and include the concepts of *message*, *command*, *aggregate, aggregate relationship,* and *property*.

Sources: MB-XML

IBM has worked with standards groups, user groups and individual customers over a number of years on Model Based Architectures in many industries. IAA-XML is the direct ancestor of MB-XML. It is a messaging architecture for inter-application communications applicable to any industry. It is also a method that explains how to use this architecture together with a UML model called Interface Design Model (IDM). Different Interface Design Models can be used depending on the subject domain. IAA-XML was developed from the Insurance Industry. IAA-XML together with the first Interface Design Model was published as part of IBM's Insurance Application Architecture in 2001.

The section *"Message Architecture, Mapping to Other Message Standards"* explains the relationships between MB-XML and some of the message standards produced by these groups.

The MB-XML architecture can be used independently of any specific message, by anyone, independently of any technology, vendor, or subject domain.

Messaging Environment

At the heart of an organization are its business processes. Each process is a sequence of steps, some automated by computers and some manual. As computerisation increases the number of automated steps, the need for applications to interoperate grows. Where previously there may have been a number of manual steps between each automated step, with output from one application being massaged prior to being re-entered into another application, now it is common for the processing carried out by one application to be immediately followed by that of another.

The obvious consequence of this increased automation is increased inter-application communication. This would be easy to achieve if every application were built with the same understanding and view of the information that it deals with (for example, based on the same business model), and implemented according to the same technical architecture. However, this is typically not the case. The fact that different applications are written by different people to satisfy different business needs is enough to ensure that analysis and design models differ. Furthermore, incompatible technical architectures arise from the natural technology evolution (through the 1990s we have seen the shift in design emphasis from 2-tier procedural to 3-tier distributed object, to the current flavour of the day, n-tier web-oriented EJBs).

The XML markup language helps to address the demands of inter-application communication by providing a message definition standard designed for easy implementation in commercial and web environments.

XML

The XML messaging standard has achieved a high level of acceptance in business and technical communities, and is an open and public domain standard. Many tools and code libraries exist for handling XML messages. XML is supported by international consortiums such as the Organisation for the Advancement of Structured Information Standards (OASIS) and World Wide Web Consortium (W3C).

The benefits of XML include the following:

Self-defining Messages

Using tags to structure and define the content of messages, XML provides a messaging capability in which the data is non-positional and self-defining.

Technology Independence

XML provides a communication mechanism independent of technology, protocol and middleware. XML is intended to support a wide range of communication protocols, including asynchronous message-oriented inter-application communication, and a conversational command/return protocol.

Shared Syntax

Unlike, for example, COBOL copybooks or C structures, XML does not restrict you to using a particular programming language. This means that it can be used to link applications written in any language.

Extensibility

The XML mark-up language can be easily extended and customized to meet the requirements of specific implementations.

The XML definition therefore provides a good basis for defining standards for communications between applications and disparate systems, but to successfully implement XML, standards for its usage must be agreed. In particular, to achieve reuse of message elements, higher-level standards (e.g., common semantics) for the data being transported must be agreed to. This is addressed in MB-XML.

Why MB-XML?

The strength of MB-XML lies in its being based on a well-structured, normalized model. This provides the following advantages:

Shared Semantic and Content Structure

In order to communicate, two applications must share the same understanding of the world. For example, if one application processes a person's address as a single string of data, and another processes it as comprising street, town, state, and country, they have a built-in impedance to communication.

In order to use the MB-XML approach, a normalized model is needed that guarantees unique semantics and content structure, i.e., a common reference point. For example, when two applications communicate about a PostalAddress, they can agree on what it means, what data it contains, and how it is related to other concepts (such as people and policies).

Structured Extensibility

As discussed above, XML is designed to be extensible. The MB-XML architecture preserves this extensibility, and indeed it is expected that users will take the base content and extend and customize it to support their own requirements. In addition, with the normalized view of data in MB-XML, additional content can be added in a more structured way, with more potential for reuse.

Where Would You Use MB-XML?

MB-XML messages are expected to be used by applications that share and exchange information in a standard and well-structured fashion.

MB-XML is particularly suitable where there are more than two applications involved, and the applications sending the messages are unaware of the characteristics of the applications which will receive and process them.

If there were only ever two applications that would communicate to each other, it is possible that there would be other more efficient and responsive choices for inter-application communication. However, these more specialised approaches are likely to be an inhibitor if new applications are brought into the picture over time. The opportunities for reuse of existing messages would be more limited than with MB-XML.

MB-XML supports different types of communication, for example:

request-response messaging protocols.
publish-subscribe commands.

MB-XML is not appropriate for messaging designs that deal specifically with displaying directly the message data, for example with a browser. The protocols supported by MB-XML are discussed in more detail in the section *Message Architecture, Types of communications supported*.

How Would You Use MB-XML?

One possible use of MB-XML is to make it the XML messaging medium of a message hub. By having a common semantic, structure and mark-up, the amount of inter-application communication code required is minimised. Consider three applications where each application had code to talk to the other two. There would be six sets of inter-application communications code. With a common intermediary set of messages implemented on a middleware architecture, this number reduces to three, through the reuse of a common messaging approach. As the number of applications increase, the benefits increase exponentially, as illustrated in Figure 1.

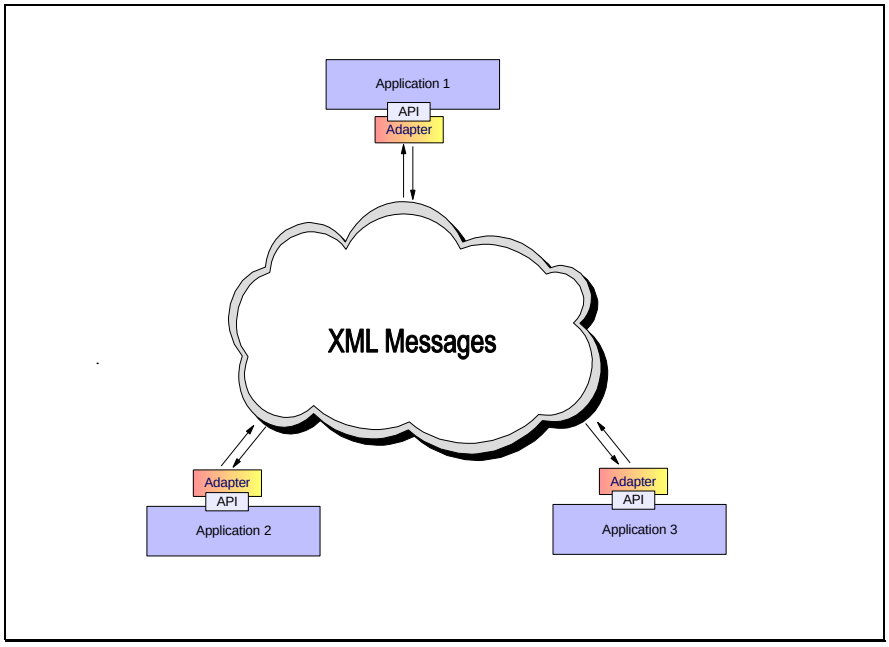

Figure 1. MB-XML Messages in a middleware Architecture

Further information on how to use MB-XML can be found in the next two sections, *Message Architecture* and *Message Design*.

Message Architecture

This section presents an overview of the MB-XML message architecture.

MB-XML Four-Layer Architecture

MB-XML messages structure information into four layers: message, command, aggregate, and property. The structure facilitates effective use of the information across software boundaries.

Message Layer

Each MB-XML message contains information common to the entire message. This information allows the communication hub and other systems to identify the message uniquely, interpret the message properly, and determine whether a response is required.

Command Layer

Each command within the message contains information about an action and the subject of that action. The command layer provides enough information about the command for the receiving software to process it accurately.

The actions include, but are not limited to, add, modify, delete, get (an instance) and search for information. These data-centric actions are the minimum required to communicate across software boundaries. While the data-centric actions are defined completely in the MB-XML architecture, MB-XML also applies to business-oriented actions. These business actions can be added to MB-XML as needed, and it is likely that future versions of MB-XML will incorporate more business-oriented command definitions.

The subjects of the commands are the fundamental, abstract entities used by the business. The subjects are independent of one another, although relationships can exist between them.

Three types of commands have been defined in MB-XML.

1. The **request** command requests that the action be performed and may also request for a response. A request command may require a response always or only if an error occurs, or it may not require a response at all. A request may also require the responding software to return certain passed data unchanged, with the response.

2. **Response** is a second command type and signals the response to a request.

3. Besides request and response, the command may be a **notification**, informing other software about an action that has already occurred.

An *AddPartyRequest* asks other software to create a party, while an *AddPartyNotification* informs the other software that the party has been created.

Note that one message may contain commands of different types.

Aggregate Layer

Aggregates provide a convenient way to address groups of properties that are often used together. They also may serve as limiting, functional views of the subjects of the commands.

The identification of aggregates may seem quite arbitrary. Using a UML model makes the identification of aggregates much more logical. Aggregates in MB-XML are identified as follows:

- Each type in the model is the basis for an aggregate. These types may represent logical subsets of the properties of the subject; they may also represent relationships between other types. Thus, there is an aggregate for Person, Organization, and PersonName.
- Aggregates can be referred to through their key as explained in the Message design section.
- Aggregates can also be defined to represent the return of certain operations.

Another element type, *Relationship*, is defined in the aggregate layer. A relationship is an element that defines a relationship between two aggregates. Relationships are used when aggregates could be related to each other in multiple ways. For example, there could be two relationships between a person and a communication: sender and receiver. In some rare cases, relationships can also have properties. A relationship corresponds to an association-end in the Interface Design Model (IDN) MB-XML model.

Property Layer

The property layer of the message represents each indivisible piece of data. Properties may be required or optional in aggregates; they may occur multiple times in an aggregate. These capabilities are standard XML functions.

Example of a Message

The following example of an XML message, based on the MB-XML Message Architecture, describes the creation of a new party - Jane Anderson. Jane's postal address and full name are specified.

Message Layer

```
<Message id='MSG1' version='1.4' bodyType='MB-XML'
timeStampCreated='1999-12-04T12:33:04'
sourceLogicalId='METAUTO1'
destionationLogicalId='CIIS'
authenticationId='SYS'
crfCmdMode='alwaysRespond'>

<Default>
       <DefaultTime zoneOffset='00:00'/>
</Default>

<CrfActionGroup bodyCategory='AddNewPerson'>
       <CommandReference refid='CMD1'/>
       <KeyGroup id='K1' keyGroupType='Person'>
              <UUID>1234</UUID>
       </KeyGroup>
       <KeyGroup id='K2' keyGroupType='PostalAddress'>
              <UUID>1235</UUID>
       </KeyGroup>
       <KeyGroup id='K3' keyGroupType='TelephoneNumber'>
              <UUID>1236</UUID>
       </KeyGroup>
</CrfActionGroup>
```

Command Layer

```
<COMMAND
```

```
   <AddPartyRequest id='CMD1' cmdType='request' cmdMode='always
Respond' defaultState='added'>
```

Aggregate Layer (Person)

```
<Person>
<KeyGroup refid='K1'/>
<birthDate>1962-03-04</birthDate>
<deathDate>2000-02-08</deathDate>
<gender>Female</gender>
<maritalStatus>Single</maritalStatus>
<grossIncome>
   <currencyAmount>32000</CurrencyAmount>
   <currencyCode>USD</CurrencyCode>
</grossIncome>
```

Aggregate Layer (PersonName)

```
<PersonName>
      <usage>Legal</use>
      <firstName>Jane</firstName>
      <lastName>Anderson</lastName>

</PersonName>
```

Aggregate Layer (ContactPreference)

```
<ContactPreference>
      <type>Home</type>

      <preferredLanguage>French</preferredLanguage>
```

Aggregate Layer (PostalAddress)

```
            <PostalAddress>
                  <KeyGroup refid='K2'/>
                  <effectiveFromDate>1943-02-05</effectiveFromDate>
                  <effectiveToDate>2010-01-01</effectiveToDate>
                  <city>Danbury</city>
                  <country>USA</country>
                  <region>New York</region>
                  <street>York Street</street>
                  <houseNumber>21</number>
                  <postalCode>21334</postalCode>
            </PostalAddress>
            <TelephoneNumber>
                <KeyGroup refid='K3'/>
                <contactInformation>+44-4356742-
1434</contactInformation>
            </TelephoneNumber>
      </ContactPreference>
</Person>
```

[End command]

```
   </AddPartyRequest>
</COMMAND>
```

[End message]

```
</Message>
```

Mapping to Other Message Standards

Introduction

Various standards organizations are creating messages in XML format for use in different industries. The standards from ACORD, IFX, and OAG have been considered.

MB-XML, therefore, does not completely adopt any one of these standards. It does, however, integrate concepts from all three organizations and can accommodate all of their messages.

ACORD

The messages produced by the ACORD group apply to insurance, emphasising the messages between brokers and insurance providers. ACORD messages are specifically designed for applications that use those forms. They are also appropriate for displaying data in the forms used.

The ACORD message name could be considered an action, but ACORD messages act directly on properties or aggregates. An ACORD message corresponds to an MB-XML command together with all its parameters (all the aggregates and properties defined as being part of the command).

IFX

Interactive Financial Exchange (IFX) messages are oriented towards business-to-business transactions for banking. They are structured the same as MB-XML: an MB-XML message is called a "service" in IFX; a command is an IFX "message. The IFX actions are data-oriented; they map easily to MB-XML commands.

OAG

The participants in the Open Application Group (OAG) come primarily from the manufacturing industry.

OAG messages contain a single command. This structure can be seen as a subset of the MB-XML structure. OAG messages support the concepts of *"verb,"* which corresponds to the MB-XML command, and *"noun,"* which corresponds to the subject of the command. And OAG defines aggregates, but like ACORD aggregates, some of them are somewhat arbitrary. OAG *"verbs"* tend to be data-oriented and can be mapped to MB-XML commands.

CP Exchange

CP Exchange is an emerging XML standard in the area of Party information. There has been a strong collaboration between MB-XML and CP Exchange, both in terms of business content and architecture. The model used for CP Exchange is the Party area of the IAA model, with very few modifications.

Types of Communications Supported

There are several modes of software communication, and these modes may have an impact on the content of the messages used, as shown in Table 1.

Table 1. Types of Communication Supported

Type	Description	MB-XML
Conversational	The parties to the conversation are "connected" and have access to a single parameter set.	Not supported.
Request / Response	The sender of the message requests some action to be performed by the receiver and expects some response back. The response may be requested only when an error occurs, or it may ALWAYS be required. The request by the sender is not synchronised with the receiver, although the sender may wait for the reply before proceeding and thus simulate conversational communication.	Two commands: 1. Command type *request* with response mode of *always* or *error only* 2. Command type *response*.
Fire and Forget	The sender of the message requests some action to be performed by the receiver but does not expect a response.	Command type *request* with response mode of *never*. There is no response command.
Publish / Subscribe	The sender of the message notifies others that some action has occurred, generally in response to a business event. Software interested in the action must subscribe to it through the messaging facility.	Command type *notification*. Response mode of *always* or *on error* only refers to responses from the *outgoing* adapter or the message hub. Responses from the subscriber end of the message are not reported.

The command types *Notification*, *Request*, and *Response* are all in some ways a notification. All messages are intended to communicate information to something. The differences in command types are merely ways to make the style of communication more obvious. In all cases, the middleware determines where to send the message, based on rules that examine the message content.

Note that the message middleware facilities determine how messages are routed. Products such as *WebSphere MQ Integrator* enable users to specify rules for subscriptions that examine the XML message elements. The details of how that specification is performed, however, are outside the scope of this document.

Message Design

In addition to the architecture described above there are a number of general design principles that underpin MB-XML as well as more specific implementation rules for each of the four MB-XML message layers.

General Design Principles

Focus on messages that support the business
Use industry standard terminology wherever possible
Do not repeat business data within a message
Reduce the amount and depth of tag nesting where possible
Aid reuse by limiting the number of alternative MB-XML representations for the same instantiated graph of type instances
Express the specification in data type definitions (DTD)s

The Use of Types

It is useful at this point to understand the general Type concepts that underlie the MB-XML Type tag definitions, and which are referred to later in this document. These are based on established Object-Oriented class structuring concepts.

Abstract and Concrete Types

Every type can be qualified as either abstract or concrete. An abstract type is never directly instantiated but is used to identify commonalties between different business concepts. For example, Party is an abstract type, while Organization and Person are the concrete subtypes of Party.

Inheritance

In order to reduce the complexity of the DTD, all the type hierarchies from the UML model have been "flattened" in the DTD. A property or association defined at the level of a superclass is duplicated in the definitions of all the subclasses. [1]

For example, in the model, PostalAddress is defined as a subtype of the abstract type ContactPoint.

There will be no element for ContactPoint (because it is an abstract type) and PostalAddress will look like:

```
<!ELEMENT PostalAddress
(KeyGroup?,effectiveFromDate?,effectiveToDate?,city?,country?,region?,street?,houseNu
mber?,addressLines*,postalCode?,boxNumber?,unitNumber?,floorNumber?,buildingName?,t
ype?,contactInformation?,Place?)>
```

With the part in bold being all the attributes and associations inherited from ContactPoint.

While this approach implies that there will be some redundancy in the DTD, it has absolutely no impact on the XML messages. The redundancy in the DTD is itself not a problem in our

[1] Originally, XML entities were used to represent the type hierarchies but most of the existing tools on the market could not support multiple levels of entities nesting so we had to back up to a simpler DTD.

approach since the repository for the content of the message definitions is still the UML model.

Finally, if you want to define an abstract association to ContactPoint, which means an association to any concrete subtype of ContactPoint, or if you want to specify the super-type as a parameter in a command, you must define an entity for the abstract super-type to make the link to the concrete subtypes:

```
<!ENTITY % InheritanceContactPoint
"(TelephoneNumber|EMailAddress|PostalAddress|UnstructuredContactPoint|CareOfAddress)">
```

This ensures that every time a ContactPoint is referenced (via InheritanceContactPoint), the XML is valid when any one of the concrete subtypes of ContactPoint is provided.

Use of Identifiers

When it comes to referring to an existing aggregate (as with a request to *modify* or *delete* an aggregate), some mechanism is required to identify the target data of the modification or deletion. Identifiers are also necessary to reference aggregates, as explained in the following section.

Key

A key identifies an aggregate uniquely across the whole space of messages.

Typically, when a piece of data is created, one system takes responsibility for owning it and managing it. This system is referred to as its system of record and has the right to allocate its unique universal id (UUID). Sometimes this UUID may be allocated by special management software, and this is the case for WMQI Enabler, where the hub allocates UUIDs that can be used across systems. The key information (which includes the unique universal id), or a reference to it, appears as the first element within an aggregate.

Keys are represented in a special structure called *KeyGroup*.

The following is an example of a key defined for a Person aggregate.

```
<Person >
    <KeyGroup>
        <UUID> PERS00432342 </UUID>
    </KeyGroup>    …
</Person>
```

Alternate ids

An alternate id is a key by which a specific system would know an aggregate. Alternate ids are represented as part of the KeyGroup structure in MB-XML:

```
<Person >
    <KeyGroup>
        <UUID> PERS00432342 </UUID>
        <AlternateId value='rowId123' sourceLogicalId='Siebel' state='added'/>
    </KeyGroup>    …
</Person>
```


Using Keys in the Context of a Message Hub

If messages are sent via a message hub, all key information must be placed in the message header, and the aggregates contain only references to entries in the key section of the message header:

```
<Message>
        …
        <KeyGroup id='K1'>
                <UUID> PERS00432342 </UUID>
                <AlternateId value='rowId123' sourceLogicalId='Siebel' state='added'/>

        </KeyGroup>     …
        …
    <COMMAND >
        <AddPartyRequest id='C1'>

                <Person >
                        <KeyGroup refid='K1'/>
                        …
                </Person>

        </AddPartyRequest>
    </COMMAND>
</Message>
```

Representing Relationships between Aggregates

There are several ways to represent relationships between aggregates: embedding (direct or indirect) and referencing (internal or external).

Embedding

The simplest way is direct embedding. Embedding is used when there is no possible ambiguity in the nature of the relationship between two aggregates.

```
<Person>
    …
    <PersonName>
        …
    </PersonName>
</Person>
```

A slightly more complex way is embedding through a relationship construct, as shown below.

```
<Communication>
    …
    <sender>
        <Person>
                …
        </Person>
    </sender>
</Communication >
```

A relationship construct is used when the same aggregate (in this case Person) could be used in multiple ways by the referencing aggregate (sender or receiver of the Communication, for example).

Referencing

In order to avoid data redundancy in the XML documents, referencing is used when the same information is needed in multiple places within the same XML message.

The standard way of referencing is by using id (id attribute of type ID: unique identifier for the element instance within the document/message) and references to the id (refid attribute of type IDREF: reference to an id within the document/message).

```
<Person>
    <ownedOrganization>
        <Organization refid='123'/>
    </ownedOrganization>
</Person>
.. and somewhere else in the message
<Organization id='123'>
    …
</Organization>
```

The final case for referencing is when you want to reference something not defined in the document. Using IDREFs is not appropriate in that case because IDREFs can only reference an id within the document. In this case, the reference is expressed as element content of the <UUID> property in the KeyGroup structure, and the *externalRefid* is set to true.

The following example shows a message where you want to pass the information that a person owns an organization without passing the organization itself. The UUID of the organization is passed as content of the Organization aggregate.

```
<Person>
    <ownedOrganization>
        <Organization  externalRefid='true'/>
            <KeyGroup>
                <UUID>12345B568135131</UUID>
            </KeyGroup>
        </Organization>
    </ownedOrganization>
</Person>
```

The DTD reflects both cases, embedding and referencing. One limitation of this approach is that all properties and relationships must become optional in the DTD, since if the element is referenced and not embedded, its properties will not be present.

Allocating Elements to Layers

Every layer has a corresponding entity that contains all the XML attributes defined at this level. For example, the %Aggregate entity defines all the XML attributes for MB-XML aggregates. Every MB-XML aggregate will include these attributes as part of its definition. In order to recognise that an element is part of a certain layer (and thereby has the attributes of that layer defined for it), it requires a link to the entity through the ATTLIST construct of XML.

For example, the following DTD expresses the fact that a postal address is an aggregate:

<!ATTLIST PostalAddress %Aggregate;>

Specifying Null Values

The distinction between property values that are blank and those that have been omitted or unknown must be supported by MB-XML messages.

MB-XML recognises null values, signifying omission, in one of two ways. If an aggregate is being added initially and the tag is optional, the sender can simply omit the tag to signify null values. Or the sender can signify null values on an *add* or a *modify* by specifying no spaces between the begin and end tags. If a response requests a tag that is nulls, the same mechanism is used. To specify blank values rather than null, at least one space must exist between the begin tags and end tags.

The following examples show this mechanism:

Adding a null value:
<tag state='added'></tag>
(If the aggregate to which the tag is being added does not exist prior to the request, the tag can simply be omitted from the request, as well).

Update to a null value:
<tag state='modified'></tag>
or <tag state='deleted'></tag>

Return nulls from inquiry:
<tag></tag>

Adding a blank value:
<tag state='added'> </tag>

Updating to a blank value:
<tag state='modified'> </tag>

Returning a blank value:
<tag> </tag>

Message Layer

The Message represents the top layer in the MB-XML message architecture. Messages include one or more commands that are related; for example, as part of a business transaction or even just grouped for performance reasons.

Message and Transaction Scope

An XML attribute, *txnScope* tells you whether the commands of a given message are transaction-independent. If the *txnScope* attribute is set to *all*, each command in a message is part of one single business transaction and if one fails, they all must be rolled back. If this attribute is set to *any*, the commands are transaction-independent.

Note: The stringency attached to database transactions does not need to apply to business transactions. If one of a sequence of commands fails, it is permissible for an alert to be generated on a console and for a human operator to regress the applied changes. A full two-phase commit, if it can be implemented, is the ideal way to manage business transactions.

Message Layer Rules

A Message can be seen as a container for a set of business information (Commands, Aggregates and Properties).

A Message can enclose 1:m *Commands.*

A Message can also contain default values, message origination information, and message processing information.

The Message contains header information for message management and authentication. The header defines default routing and message processing (acknowledgement and publication). The header also contains one action group for each MB-XML command, connected via an XML reference to a COMMAND element wrapping each MB-XML command. This allows the message hub to deal with each MB-XML command separately for routing and message response purposes.

At the level of the message header, a default command mode (type of acknowledgement requested) and publication information must be defined. These can be overridden at the level of an individual command inside the action group.

Within the action group, optional routing, message acknowledgement, and publication information can be specified at a command level. If this information is not present, the top (Message) level values are used.

Within each action group is a key group. The key group holds the actual key entries for aggregates contained in a command. If it is to be processed by the hub, keys for MB-XML aggregates are not stored in the aggregates, but in the key group inside the message header and are accessed via the XML reference mechanism.

WMQI Enabler will process information in the header and use it to update its internal state information.

Note that the primary focus of MB-XML is to provide a basis for representing the business content of a message. The message layer specification in MB-XML provides an example of the type of information that should be specified at this level, e.g. elements and attributes are provided to represent system and routing related information, and information about system defaults.

These definitions may be extended in the future to provide consistency with other adopted XML standards.

Message Element and Attribute Meaning

Table 2 outlines the meaning of the elements and attributes defined at the *message* layer.

Table 2. Message Elements and Attribute Meanings[2]

Element	Structure	Attribute	Data Type	Usage	Default	Description
Message	(top level)[3]			Required		A Message is a container for Commands. It can also contain

[2] WMQI Enabler-specific elements and attributes are pre-fixed with *crf*, which stands for Cross-Reference File.

[3] If text is shown in parenthesis, it is not interpreted literally but refers to an MB-XML construct that can be inferred from the context. In this particular case, (top level) indicates that the Message element should appear as the highest level tag inside an MB-XML message.

						special elements such as Defaults, ErrorInfo and CrfActionGroup (described below).
		id	ID	Required	-	A unique id of the message. Every instance of every message must have its own unique id.
		sessionId	String	Optional	-	A number of Messages may be sent on the same topic OR a chain of messages may originate from one Message. The sessionId attribute is used to associate these messages together. Associated messages (such as a the messages containing a Command and the one containing its Response) will have the same sessionId value.
		version	String	Required	-	The version of the MB-XML architecture / message structure. Only applies to MB-XML if bodyType='MB-XML'. For this version of the MB-XML specification, version='1.5[4]'
		bodyType	Enumeration	Required	-	Identifier that describes the architecture of the message content. Defined values:

[4] Although, this is the first officially published version, this is actually the fifth iteration of the specification.

						MB-XML EID …
		timeStampCreated	Timestamp	Required	-	The time at which the message was created in the issuing system. Specified as a full yyyy-mm-ddThh:mm:ss.ffffff timestamp.
		timeStampExpired	Timestamp	Optional	-	The time at which the message is no longer valid / expires.
		sourceLogicalId	String	Required	-	An identifier of the system that issued the message.
		destinationLogicalId	String	Optional	-	An identifier of the system that should receive the message. In the case of a notification, the DestinationLogicalId attribute will usually not be taken into account (it might be taken into account if the notification does not use the publish-subscribe mechanism)
		authenticationId	String	Optional	-	An identifier of the end user using the system that issued the message. In most cases messages to modify or query data will need some form of credential to ensure the sender is authorized. This credential can be stored in the *authenticationId* attribute.

		crfPublish	Boolean	Optional	false	Defines whether to publish the hub activity to subscribers. Permitted values: true false Defines the default value for the crfPublish attribute inside CrfActionGroup.
		crfCmdMode	String	Optional	onlyRespondInError	Defines what type of response the hub will generate. Permitted values: alwaysRespond neverRespond onlyRespondInError Defines the default value for the crfCmdMode attribute inside CrfActionGroup.
		txnScope	String	Optional	all	Permitted values: all any all means that all Commands in the Message are considered part of a transaction. any indicates that Commands are transaction independent.
ErrorInfo	in Message					Structure describing the error condition for this action group. See section 'Command layer (Command elements)' for details on the ErrorInfo structure.

Default	in Message			Optional		There are a number of XML attributes listed

						below which can appear (mostly as XML elements) many times in a message but which typically take the same value. The Defaults section allows them to be predefined so to reduce verbosity and reduce message size.
DefaultTime	in Default			Optional		Pertains to properties of type Timestamp
		zoneOffset	UTCOffset	Optional	-	The default time zone offset from UTC. Values must be within –720 through +720. Value is typically a multiple of 60 (an exact number of hours) but the offset may also include half and quarter hours. If this attribute is not included, an offset of 0 is assumed.
DefaultCurrency	in Default			Optional		Pertains to properties of type CurrencyAmount
		currencyCode	Enumeration	Required	-	The default currency code value for all the currency amounts not specifying a value for the currency code. The currency codes use the ISO-4217 values.
DefaultUnit	in Default		String	Optional, Repeating		Defines the unit to be used by default for measuring

						something corresponding to the measured concept. An example would be kg for weight.
		measuredConcept	String	Required	-	The measured concept such as weight, distance, temperature…
CrfAction Group	in Message			Optional, Repeating		Grouping of hub activity related to a command section. Must be defined per command contained in the message. Required if message to be processed by the hub.
		bodyCategory	String	Required	-	The name of the Workflow script that the adapter wants used to process a message.
		crfPublish	Boolean	Optional	false	Determines if the messages need to be published specific to this action group. Permitted values: true false
		crfCmdMode	Enumeration	Optional	onlyRespondInError	The type of response the action group will need. Permitted values: alwaysRespond neverRespond onlyRespondInError This attribute represents a specialised version of cmdType

						attribute defined for MB-XML commands.
		destinationLogicalId	String	Optional	-	Who will receive the message specific to this action group.
CommandReference	in CrfActionGroup			Optional		Refers to a (command) in the core (business content) section of the message.
		refid	IDREF	Optional	-	Reference to a COMMAND element which corresponds to this action group.
ErrorInfo	in CrfActionGroup			Optional		Structure describing the error condition for this action group. See section “Command layer (Command elements)” for details on the ErrorInfo structure.
KeyGroup	in CrfActionGroup			Optional, Repeating		Must be defined per aggregate contained in a particular command. For the WMQI Enabler hub, the Keygroup information is provided in the message header instead of in the aggregate. The aggregate refers to its keygroup by a refid. If an aggregate has multiple keys, multiple alternate ids will show within KeyGroup. Assumes all keys are related by UUID.

		id	ID	Optional	-	Identifier to allow referencing of key blocks within a message.
		refid	IDREF	Optional	-	Reference to key block (used at the level of the aggregates to reference to their keys in the header).
		keyGroupType	String	Optional	-	Identifies the type of key. Corresponds to the aggregate type. For example: 'Person', 'PostalAddress'
UUID	in KeyGroup		String	Optional		Universal unique identifier used to access all keys for all systems corresponding to this entry.
AlternateId	in KeyGroup			Optional, Repeating		Alternative / system specific key used by a particular system to identify an aggregate.
		value	String	Required	-	Contains a key value stored in a particular system (if different from UUID).
		sourceLogicalId	String	Required	-	Identification of the system owning this key.
		attributeString	String	Optional	-	Holds a (set of) string(s) which may be required by certain systems to be held in the hub cross reference file (CRF). If a separator is used, it should be a comma.
		state	Enumeration	Required	referenced	Action to be performed / that was performed

						on the key where: referenced => means the key is just information sent from one system to another and that CRF should not have an entry for it add => means the key was just created in a system and needs to be added to CRF, only for message into hub added => means the key was just created in the originating system and in the CRF, only for message coming from hub exists => means the key is in the CRF & should be translated for another system modify => means the key was modified in its system and needs to be changed in the CRF, only for message into hub modified => means the key was modified in the originating system and in the CRF, only for message coming from hub delete => means the key needs to be removed from the CRF, only for message into hub

		newValue	String	Optional, only valid for state='modify'	-	Value the key should be modified to. If a newValue is present, then the atttributeString attribute (if present) contains the new attributeString.

Message Example

```
<Message id='M1' version='1.4' bodyType='MB-XML'
      timeStampCreated='2000-08-22-T23:59:00'
      timeStampExpired='2000-10-14-T23:59:00'
      sourceLogicalId='YourSystem'
      authenticationId='MyUserId''
      -> this is a required attribute
      crfPublish='true'>
    <Default>
          <DefaultTime zoneOffset='00:00'/>
          <DefaultCurrency currencyCode='USD'/>
          <DefaultUnit measuredConcept='weight'>kg</DefaultUnit
    </Default>
    <CrfActionGroup bodyCategory='AddNewPerson'>
          <CommandReference refid='xx'/>
          <KeyGroup id='K1' keyGroupType='Person'>
                <AlternateId value='rowId123'
                      sourceLogicalId='SIEBEL'
                      attributeString = 'the agent number'
                      state='add'

          </KeyGroup>
          …
    </CrfActionGroup>
    <COMMAND>
          <AddPartyRequest id='xx' cmdType='request'>
                <Person>
                      <KeyGroup refid = 'k1'/>
                …
                </Person>
          </AddPartyRequest>
    </COMMAND>
    …
</Message>
```

Command Layer

Properties and aggregates specify data content and relationships; they do not describe what to do with the data. The command layer provides this information. While the elements of the aggregate and property layers are derived from the types defined in the model, and are generally applicable to several areas, the commands represent the particular reason this message was sent. Each command is unique, but the aggregates and properties may be reused many times across multiple commands.

Commands are the third layer of the MB-XML architecture. A command encloses a number of aggregates. The exact set of aggregates enclosed depends on the command. Think of commands as a procedure call, or an operation on an object, which takes parameters. The command is the procedure (or operation) name and the enclosed aggregates are the parameters of the command (in the case of the object analogy, one of the aggregates may be considered to be the object on which the command executes while the remaining aggregates are parameters).

Commands can only appear inside a *Message* element as defined in the message layer.

Command Types

Three types of commands are defined in the MB-XML architecture: *Request*, *Response*, and *Notification*.

A **Notification** command corresponds to publishing an event that has already happened. Such a command does not expect any response from any system. It is an informative command. Each application must decide what events are to be broadcast. A CRM application may decide to notify the world each time a party entry is created (*AddParty notification)*. All applications that have subscribed to that particular event are informed and will take whatever action they consider necessary. The notification command usually implies a Publish/Subscribe mechanism.

A **Request** command is used when the originating application expects a particular system to take an action. An order management system can issue an AddParty request command to create a new customer. It is assumed that one and only one system has registered to this event and will process the request. In such a case, the order management expects the Party system to respond to the request with a confirmation such as *party created successfully* and possibly the party UUID (using a Response command).

A **Response** command is used to respond to a Request command. A response is sent only to the originating system. Other applications are notified by the notification mechanism.

Response to a Command

A *notification* or *response* command never expects a response. A *request* command must specify the type of response it expects. It must also specify if the original data must be echoed back in the response message.

The attribute *cmdMode* is used to specify the type of response a command expects. It has three possible values:

alwaysRespond
neverRespond
onlyRespondInError

A response message includes the attribute *cmdStatus* to indicate if the request could be executed successfully.

Message and Command Id

A response command must match a request and must therefore contain a reference to (the id of) both the requesting message (attribute refidMsg) and the requesting command in this message (attribute refidRequest).

Naming Conventions for the Commands

Every command name consists of three elements:

a verb
a noun
a suffix

The verb defines the action to be applied to the data passed in a command. Verbs encompass Create/Update/Delete (CRUD) type verbs and also more business-oriented ones.

The noun specifies the data the action should be applies to.

The suffix specifies Request, Response, or Notification depending on the command type.

An example of a command is *AddPartyRequest*.

List of Verbs

A minimum list of verbs must include the usual data-centric actions to add, modify, delete, get (an instance) and search for information. The MB-XML terminology is aligned to IFX for this purpose:

Add
Modify
Delete
Cancel
Inquiry

As we move into more transactional commands, we will add other verbs such as Submit, Accept, Issue, and so on.

Inquiry Command Guidelines

For very simple inquiries, the aggregate defined for a corresponding *Add* command could be reused. For example, if you search for a particular postal address (independent from its relationship to a person, etc.), the PostalAddress aggregate could be used in an inquiryPostalAddress command. However, in most cases a different aggregate is defined to specify the search criteria for an inquiry command. It assembles properties found on a set of different aggregates, such as Person, ContactPreference and PostalAddress for an inquiryPartyRequest command. The properties in this aggregate mean the same as properties used in any other command, but their values specify what the property matches for the search. Thus <lastName>Smith</lastName> specifies that the last name is Smith in all persons retrieved by the command.

As the logical operator for the commands is *matches*, the values can include wildcards in the same way that SQL “matches” values can use wildcards. An asterisk (*), for instance, matches any character. If more than one element is specified, the elements are combined with the *and* logical connector. Thus:

```
<lastName>Smith</lastName>
<firstName>Betty</firstName>
```

retrieves all persons whose last name is “Smith” and whose first name is “Betty.”

Note that the XML specification allows these criteria, but it does not implement the search. For the search to be successful, the receiving program must interpret and use the criteria as specified.

Processing Commands in the Message Hub

For processing within the message hub, the MB-XML command must be enclosed in <COMMAND> </COMMAND> tags. The COMMAND element is not considered as a separate layer in the MB-XML architecture. It is part of the command layer and only represents a technical wrapper around the actual command. Command-related information, such as the command type or command mode, are defined as part of the actual command element.[5]

Command Tag and Attribute Meaning

Table 3 defines the elements and attributes used in the command layer.

Table 3. Command Tags and Attribute Meanings

Element	Structure	Attribute	Data Type	Usage	Default	Description
COMMAND	in Message			Required if message is to be processed by the WMQI Enabler hub.		A wrapper defining that the enclosed element is an MB-XML command.
(command)	in COMMAND			Required		A command specifies an operation on one or more nested Aggregates.
		id	ID	Required	-	Id is a unique reference to the Command within the scope of the message.
		cmdType	Enumeration	Optional	request	The type of command as defined by the issuing system. Permitted values: request response notification
		cmdMode	Enumeration	Optional	onlyRespondInError	Permitted values: alwaysRespond neverRespond onlyRespondInError The type of response expected by the sending system.
		echoBack	Boolean	Optional, only applies to cmdType “request”	false	Permitted value: true false Defines if the content of a

[5] The <COMMAND> wrapper might be integrated with the actual command tag in future versions of MB-XML.

				"request"		request command should be contained in a response command.
		refidRequest	String	Required, only applies to cmdType "response"	-	refidRequest contains the id value of the request command that the response is issued in respect of
		refidMsg	String	Required, only applies to cmdType "response"	-	refidMsg contains the id value of the Message that the response is issued in respect of.
		cmdStatus	Enumeration	Required, only applies to cmdType "response"	-	Depending on whether the Command succeeded or failed. Permitted values: *ok* *fail* *notApplied* A value of *notApplied* indicates that a previous Command in the message failed or was never executed and as a consequence the transaction was halted.
		defaultState	Enumeration	Optional	referenced	The default for the state attribute at the aggregate and property levels of this command. If defaultState is not specified, it is derived from the verb part of the command name. Permitted values: added modified deleted referenced

Command Elements

An important design concept in MB-XML is the separation of business data from the system-specific attributes in a message. The message header already holds some system attributes, which are generic to all messages regardless of content. However, there may also be a need to hold system- and transaction-specific information that varies depending on the processing needs of the sending or receiving application. To support this, a generic structure has been created in each command, called *SystemInfo*:

```
<SystemInfo>
    <ErrorInfo>
        <errorMessageType>text</errorMessageType>
        <errorCode>text</errorCode>
        <errorMessageText>text</errorMessageText>
```

```
        <errorStatus>text</errorStatus>
    </ErrorInfo>

    <CodeValue>
        <code>text</code>
        <value>text</value>
    </CodeValue>

</SystemInfo>
```

SystemInfo is intended to hold system-specific information, without business value but required by legacy applications. It includes:

System error codes
Error information related to the processing

The ErrorInfo structure contains information related to the physical processing of the data, such as data return codes or message text.

The CodeValue structure allows you to record, for example, sequence codes and values, which help the receiving application to interpret and rebuild a series of related messages.

A sending or receiving application may not have the ability to process the complete message at one time, and may send or receive different parts of one logical message within several physical messages. For example, an XML structure may support an insurance agreement and all the vehicles and drivers insured, but the sending application may only provide this information as a series of messages by individual driver, while the receiving application needs sequence codes in order to rebuild the information. The CodeValue structure, with the *code* and *value* element pair, may be used to describe the type of sequence and the value.

SystemInfo is intended as a generic structure that can hold multiple error messages or system codes and values defined by the applications. It is important to ensure that any information targeted for the SystemInfo aggregate has been through a rigorous analysis to ensure it is truly system dependent, and has no business value.

Table 4 defines the System info element and its structural parts.

Table 4. SystemInfo Elements and Attributes

Element	Structure	Data Type	Usage	Default	Description
SystemInfo	in (command)		Optional		Holds information that a system may want to expose to the command. It is associated with each command. The information does not contain business content, but may contain information about the transaction or internal identifications that apply to the business information in the command.
ErrorInfo	in SystemInfo, in Message or in CrfActionGroup		Optional, Repeating		Contains information about errors that may be reflected by this command (this message).
errorMessageType	in ErrorInfo	String	Optional		Designates the severity of the message – sever error, error, warning, information.
errorCode	in ErrorInfo	String	Optional	-	Encodes the message for easier

					tracking.
errorMessageText	in ErrorInfo	Text	Optional	-	Specifies exactly what the error is.
errorStatus	in ErrorInfo	String	Optional	-	Specifies the Status of the command.
CodeValue	in SystemInfo		Optional, Repeating		Contains a pair of a name and its value.
code	in CodeValue	String	Required	-	Specifies what type of information the associated value contains.
value	in CodeValue	String	Required	-	Contains a value the system wants to pass in the message.

Parameters for Requests and Responses

Response commands contain structures (aggregates) that can hold the original data from the request command so that they can echo back the request data.

Command Example

```
<COMMAND>
    <ModifyPartyRequest id='id1'     cmdType='request'>
          <SystemInfo/>…</SystemInfo>
          <Person/>…</Person>
   </ModifyPartyRequest >
</COMMAND>
<COMMAND>
   <AddPartyRequest id='id2' cmdType='request' echoBack='true'>
          <SystemInfo/>…</SystemInfo>
          <Person/>…</Person>
   </AddPartyRequest >
</COMMAND>
```

Aggregate Layer

The *aggregate layer* defines groups of related elements to provide a mechanism for coding *composition rules,* and a convenient method to refer to related information using a *single name*. The single name used for an aggregate represents a *type*.

For example, the composition rules can define that “element 1 or element 2 must be provided”, and the aggregate also provides a convenient way for programmers to specify all of the related information by using a single name.

An *aggregate* represents one type of information or a well-defined set of related types of information that are commonly used together.

An *aggregate* groups together property elements (see Property Layer) and other aggregates.

Aggregate examples include, among others, <Person/> and <PersonName/>.

```
<PersonName id='PN00454'>
      <KeyGroup refid='K1'/>
       <usage>Legal</usage>
      <firstName>Jane</firstName>
       <lastName>Anderson</lastName>
</PersonName>
```

Note: In IFX, this layer is called the *Aggregate* building block.

Aggregate Layer Rules

Aggregate names are case sensitive. They are defined with first character as a capital letter (e.g. **P**erson, **C**ontactPreference). They should not contain special characters such as colons or spaces.

Aggregates can be sparsely populated. Not all property elements and aggregate elements defined in the specification must be specified when constructing a message.

Elements contained in an aggregate have a pre-defined order. Properties appear first in prescribed order, related aggregates appear next, also in a prescribed order.

In an MB-XML message, aggregates can only appear inside a command.

Elements contained in an aggregate have a pre-defined order. Properties appear first in prescribed order, related aggregates appear next, also in prescribed order.

Extending Aggregates

The *supertype* attribute allows you to extend the MB-XML specification by defining additional aggregates, while keeping a reference to the aggregate they extend.

For example, if you want to extend Organization by creating a new subtype called *SportsClub*, the XML will look like:

```
<SportsClub supertype='Organization'>
    <startDate>...</startDate>
    <numberOfMembers>...</numberOfMembers>
    <league>...</league>
</SportsClub>
```

So, someone who does not understand *SportClub* but does understand *Organization* could convert the XML into:

```
<Organization>
    <startDate>...</startDate>
    <numberOfMembers>...</numberOfMembers>
    <league>...</league>
<Organization>
```

The properties defined on *Organization* (foundationDate and numberOfMembers) would be understood, while the ones added on *SportsClub* (league) would be ignored.

Aggregate Tag and Attribute Meaning

Table 5 defines the elements and attributes defined at the aggregate level.

Table 5. Elements and Attributes of the Aggregate Level

Element	Structure	Attribute	Data Type	Usage	Default	Description
(aggregate)	in command OR in (aggregate) OR in (relationship)-			Required		Specifies the name of the aggregate.
		id	ID	Optional	-	A unique identifier for the aggregate within the scope of

						the message. If defined, the aggregate can be referenced by an attribute of type idref on another aggregate, somewhere else in the document.
		refid	IDREF	Optional	-	Refers to an aggregate already defined in the message. Uniquely identifies the referenced aggregate within the scope of the message.
		externalRefid	Boolean	Optional	-	Defines that the information provided in the KeyGroup structure refers to an aggregate not defined in the message. The UUID or AlternateId of the referenced aggregate has to be specified in the KeyGroup structure. Permitted values: true false
		state	Enumeration	Optional	referenced	An attribute indicating what the state of the information contained in the aggregate is. Permitted values: added modified deleted referenced State defaults to the value of the defaultState attribute in the command if it is not specified at the aggregate level.
		supertype	String	Optional	-	Defines the super type of an aggregate. To be used for extensions of the MB-XML specification.
KeyGroup	in (aggregate), must appear as first item inside aggregate			Optional	-	Definition see “Message layer” section. The KeyGroup is defined as part of the aggregate if and only if it has not already been defined in the message layer.
(relationship)	In *command* OR in (aggregate)			Optional		Specifies the name (type of) relationship between the enclosing and the enclosed aggregate.

When an aggregate is first defined in a command, then either the KeyGroup is present and the refid is not, or the KeyGroup is given in the message header and the refid here refers to the id defined in that KeyGroup. If the aggregate is defined elsewhere in the parameters of this MB-

XML command, then the refid of the aggregate (as opposed to the refid on KeyGroup) refers to the id associated with that aggregate.

Property Layer

The *Property layer* defines the atomic elements of the MB-XML Message specification. A *Property* is the most basic unit of data in the MB-XML Specification.

A *Property* defines a single piece of information passed between applications. The format of the information is specified in the property's *data type.* The definition provided with the property describes more precisely the purpose of this information.

Properties can only appear inside an aggregate or a relationship as defined in the aggregate layer.

Property examples include a *Person's* <birthDate/>, or <street/> as part of *PostalAddress*.

Note: In IFX, this layer is referred to as an *Element* Building Block. This name is not used in MB-XML because of the specific meaning of element in an XML environment.

Property Layer Rules

Property names are case sensitive. They are defined with first character as a small letter (e.g., **g**ender, **f**irstName). They should not contain special characters such as colons or spaces.

Properties are contained in aggregates or relationships.

```
<Person>
    <birthDate>1965-05-15</birthDate>
</Person>
```

Properties and Application Data

All 'application data' is represented by the values between the property start and end tags.

```
<referenceNumber>5077-512F</referenceNumber>
```

Application data is typed. Each data type has a well-defined and unambiguous format. The data type for each property is specified in the in the DTDs generated from the model.

<birthDate>1965-05-15</birthDate> holds data of type 'Date' and is formatted accordingly

Data Types

The MB-XML data type definitions are based on:

- XML Schema data type definitions, as defined in *XML schema draft specification of data types*, found at http://www.w3c.org/TR/2000/WD-xmlschema-2-20000225

  Java 2 built-in data types, as defined in *Mastering Java 2*, *John Zukowski*, *Sybex*
- IFX data type definitions as defined in *Interactive Financial Exchange Business Message Specification*, Version 1.0.1, Public Review Draft January 3, 2000[6].

[6] The latest version of the IFX specification can be downloaded from the Internet: http://www.ifxforum.org.

The following data types are supported in MB-XML:

String
Text
Binary
Boolean
Date
Time
Timestamp
Time duration
Number
Byte
Integer
Short
Decimal
Percentage
Amount
Currency amount
Enumeration
Identifier

For the definition of the MB-XML data types, please refer to the section *MB-XML Data Types*, page 53.

Units of Measure

A specific data type, *Amount*, is used in MB-XML to represent anything that is measurable. It contains a required element for the value (*theAmount*) and an optional element for the units (*unit*). The DefaultUnit element contains an XML attribute, *measuredConcept*, so that defaults can be specified for every measured concept at the level of the message (see Message layer).

For example, you can specify the following default at the level of the message:

```
<DefaultUnit measuredConcept='weight'>kg<DefaultUnit>
```

The effect is that the following element:

```
<bodyWeight>
        <theAmount>65</theAmount>
</bodyWeight>
```

is interpreted as follows:

```
<bodyWeight>
        <theAmount>65</theAmount>
        <unit>kg</unit>
</bodyWeight>
```

In the DTD, the following entity is defined:

```
<!ENTITY % Amount ' theAmount, unit?'>
<!ELEMENT theAmount(#PCDATA)>
<!ELEMENT unit (#PCDATA)>
<!ATTLIST theAmount measuredConcept>
```

Amount can be used in the same way as any other data types by specifying:

```
<!ELEMENT bodyWeight (%Amount;)>
```

Property Tag and Attribute Meaning

Table 6 defines the XML attributes defined at the level of the property layer.

Table 6. Attribute Definitions at the Property Layer

Element	Structure	Attribute	Usage	Default	Description
(property)	in (aggregate) OR in (relationship)		Required		Specifies the name of the property. The actual data (value) is defined as element content.
		state	Optional	referenced	An attribute indicating the state of the information contained in the property. Permitted values: added modified deleted referenced Defaults to state from the aggregate level to which this property belongs, if aggregate state exists. Defaults to defaultState at the command level if there is no state defined at the aggregate and property levels.

Note that certain combinations of the *state* property at the aggregate and property levels are not logical. For instance, it makes no sense to delete an aggregate but at the same time add or modify one of its properties. We cannot prevent such requests through XML, but we also cannot predict the action of applications receiving such a request.

Bibliography

This section contains a list of the books and other reference sources used to prepare this document.

David Megginson, *Structuring XML Documents* (from the Series on Open Information Management from Charles F Glodfarb) (Prentice Hall)

Luis Ennser, Christophe Chuvan, Paul Fremantle, Ramani Routray, Jouko Ruuskanen, *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, IBM Red Books

ACORD: http://www.acord.org

CPEX: Customer Profile Exchange

IFX: (Interactive Financial exchange): http://www.IFXForum.org

OAG: http://www.openapplications.org

OASIS: http://www.oasis-open.org

XML.ORG: http://www.xml.org

W3C (World Wide Web Consortium): http://www.w3.org

Glossary

This section contains a list of terms and abbreviations used in this document.

ACORD

A US standards body. ACORD have produced the ACORD Insurance Service Business Message Specification for Property and Casualty.

API

Application Programming Interface.

CIIS

Client Information Integration Solution, an implementation of a Party Management System based on the IAA model.

Class

A UML class. A description of an object.

DTD

Document Type Definition.

EJB

Enterprise Java Bean.

IAA

Insurance Application Architecture. IBM's business model for the insurance and financial services industry.

IFX

Interactive Financial eXchange. A cooperative industry effort among major financial institutions produced by the IFX Business Message Specification for the financial services industry.

OAG

Open Applications Group. A non-profit industry consortium comprised of many of the most prominent stakeholders in the business software component interoperability arena in the world. It was formed in February, 1995 in response to the rapidly expanding problem of tying disparate software applications together.

OASIS

(Organisation for the Advancement of Structured Information Standards). A non-profit international consortium founded in 1993 to advance the open interchange of documents and structured information objects. Originally focused on SGML, OASIS has evolved to more actively support XML.

Object

Instance of a class.

Party

Any person or organization that the insurance company has, or had, or may have a business interest in.

Property

A data value of a type.

Operation

A function defined on a class and executable on the object.

Property tag

An XML tag representing a property of an IAA type (represented as an UML attribute).

Root type

A type which is (one of) the main type(s) of an IAA component. The type to which types outside of the component are related to, and on which most of the types of the component depend on.

Tag

An XML construct <Tag....>.

Type tag

An XML tag representing an IAA type.

UML

Unified Modeling Language.

W3C

World Wide Web Consortium.

XML

eXtensible Markup Language. XML is a markup language for message definition, and is an open and public domain standard. XML is a subset of SGML designed for easy implementation in commercial and web environments.

XML attribute (or just attribute)

Appears in an opening tag, used to specify values in the tag. <Tag attribute='val'...> .

Appendix A: MB-XML and the underlying UML model

Benefits of Model-driven Messages

Clear Definitions

In order to fully benefit from the MB-XML approach, it is advised to use an underlying IUML model as the basis for the generation of the DTDs. This model could vary depending on the problem domain. Any model playing this role is commonly referred to as an Interface Design Model (IDM).

Every concept identified in the Interface Design Model must be defined precisely so that messages defined based on this model are consistent in structure and have a clear meaning. A type such as *PersonName* is defined unambiguously in the interface design model; the corresponding XML aggregate benefits from this definition. The same holds for attributes of the Interface Design Model.

Normalisation of Data

An Interface Design Model should be a normalized model. Every piece of data is defined once at the right place. Some compromises have to be made to guarantee performance from an implementation perspective, the structure of the XML messages can only benefit from being normalized.

Basing MB-XML on a normalized Interface Design Model allows for defining message sets around subject areas with no data redundancy.

Command Layer

The command layer is closely related to the Interface Design Model interfaces. An operation on an interface translates to a request/response command pair in MB-XML.

Appendix B: MB-XML Generation from an Interface Design Model in Rational Rose (RR)

Legend:

In the Rational Rose column, words in **bold** represent defined terms (constructs) within the Rational Rose tool
In the DTD column, words in *italics* represent dynamic parts of the generated DTD Normal text represents static parts of the DTD

Rational Rose	DTD	Comments
	"Hard-coded" ELEMENTs of DTD	
	ENTITY Command	
	ENTITY CommandReuqest	
	ENTITY CommandResponse	
	ENTITY Aggregate	
	ENTITY Relationship	
	ENTITY Property	
defined in Interface Design Model, but not used for generation	ENTITY *DataTypes*	includes definitions for all MB-XML data types defined in this document
	ELEMENT Message	
	ELEMENT Command	
	ELEMENT Default (Time, Currency, DefaultUnit)	
	ELEMENT CrfActionGroup	
	ELEMENT KeyGroup (AlternateId, UUID)	
	ELEMENT SystemInfo, CodeValue and ErrorInfo	
GENERAL RULES		
Component defined in RR **Component View**	defines scope of DTD	
OPERATIONS	**COMMAND LAYER**	
Operation defined on an **interface** assigned to component in RR Component View, no stereotype assigned	Command pair of Request and Response (and Notification) <!ELEMENT *OperationName*Request (%Command; , *OperationInputParameter*) > <!ATTLIST *operationName*Request %CommandRequest;> <!ELEMENT *OperationName*Response (%Command; *operationReturnType)*> <!ATTLIST *operationName*Response %CommandResponse;>	for MB-XML 1.4, the 'Notification' commands are not generated

CLASSES	**AGGREGATE LAYER – AGGREGATES**	
Any **class** with subclasses assigned to component in RR Component View	abstract superclass: <!ENTITIY % Inheritance*SuperClassName* '*SubClassName1* | *SubClassName2*'> concrete superclass: <!ENTITIY % Inheritance*SuperClassName* "*ClassName* | *SubClassName1* | *SubClassName2*"> if *SubClass* is not assigned to component, an 'empty / stub' %Aggregate is generated for it	Used for - associations that exist between this abstract class and others - abstract class as parameter in a command
Any concrete **class** assigned to component in RR Component View	<!ELEMENT *ClassName* (KeyGroup, *attributeName1, attributeName2, relationship1, relationship2)*> <!ATTLIST *ClassName* %Aggregate;>	
ATTRIBUTES	**PROPERTY LAYER**	
Attribute defined on a class where class assigned to Component View in RR no stereotype or stereotype <<derived>>	<!ELEMENT *attributeName* (%*DataType*;)> <!ATTLIST *attributeName* %Property; > order: same as found in RR model; generated as first set of ELEMENTs included in aggregate The order in which the properties appear as elements does not matter in the DTD. It matters only in the definition of the aggregate.	- generated as ELEMENTs after KeyGroup - all MB-XML properties are optional
RR **Type**	maps to *DataType* where *DataType* must refer to a data type as defined in hard-coded section	
RR **Initial value**; permitted values for data type Enumeration	not defined in DTD, only in MB-XML documentation	
Operation stereotype <<attnav>>	generated as %Property (see above); *operation name* = *property name* *return type* = *property* data type	
ASSOCIATIONS	**AGGREGATE LAYER – RELATIONSHIP**	
Navigable **association** from SourceClass to TargetClass - both ends of association assigned to component in RR Component View - stereotype <<generate>>	<!ELEMENT *associationEndName* (*TargetClass)*> <!ATTLIST *associationEndName* %Relationship;> the %Relationship (*associationEndName*) is included in the *SourceClass* %Aggregate if *TargetClass* is not assigned to the component, an "empty / stub" %Aggregate is generated for it	generated as sub-sequent ELEMENTs after properties
Operation, stereotype <<typenav>>	generated as %Relationship (see above); *name* = *associationEndName* *return type* = *TargetClass* embedded in	

	%Relationship [] used to express 0:m	

Appendix C: Data Types

Definition of Data Types

String

A string of characters (optionally containing blanks) for which a maximum length can be specified.

String indicates an element that allows character data up to a maximum number of characters. The number after the hyphen specifies the maximum number of characters. For example, S-12 specifies an element of characters with a maximum length 12 characters. S-8indicates an element with no maximum length. It is expected that *character* type elements may contain multibyte representations of characters in some implementations, depending on the allowable character sets.

Text

A string of characters (optionally containing blanks) for which a maximum length cannot realistically be fixed.

Binary

A finite sequence of binary octets. The definition consists of three logical elements: content type, binary data and binary data length.

The *Binary* data type is a compound type consisting of three logical elements:

Tag	*Type*	*Usage*	*Description*
<contentType>	Enumeration	Optional	Specified in IETF RFC 2046.
<binLength>	Integer	Required	Identifies the size of the binary data in number of bytes.
<binData>	Raw Binary Data	Required	Binary data.

Boolean

A logical TRUE or FALSE condition.

Date

An indication of a particular day in the Gregorian calendar.

Elements of data type *Date* contain an indication of a particular day. This data type describes a unique period of time, normally 24 hours (not a repeating portion of every year).

Logically, this data type must contain a 4-digit year, and may contain a month and day number..

Time

An indication of a particular time in a day expressed with a maximum precision of one microsecond.

Elements of data type *Time* contain an indication of a particular time during a date. This data type describes a repeating portion of a day. That is, each time described (ignoring leap seconds)

occurs once per calendar date. Based on the IFX specification, it is required that a *time* data type be able to represent a specific period with indefinite precision. Milliseconds are the minimum required precision of the data type *time*.

A time represented using this data type must not be ambiguous with respect to morning and afternoon. That is, the time must occur once and only once each 24-hour period.

In addition, the *Time* data type must not be ambiguous with respect to location at which the time occurs. If unspecified, the time zone defaults to Coordinated Universal Time (UTC). Generally, use of a specific time zone in the representation is preferred. The time zone should always be specified to avoid ambiguous communication between clients and servers.

Timestamp

An indication of a particular date and time expressed with a precision of one microsecond.

Elements of data type *Timestamp* contain year, month, day, hour, minute, second, fraction and utcOffset. *Timestamp* information is not intended to be meaningful at the other end of the communication. In addition, microseconds are the minimum required precision of the time portion of this data type.

For example, a *Timestamp* value may be generated at a server when creating an audit response. The client application may return that value to the server in later requests, but the client software should not interpret the information.

TimeDuration

A duration of time expressed in years, months, days, hours, minutes, and seconds.

Number

A numeric count not requiring any units.

Byte

A signed integer between –128 and +127, of type *Number*.

Integer

A signed integer between –2147483648 and +2147483647, of type *Number*.

Short

A signed integer between –32768 and +32767, of type *Number*.

Decimal

A numeric value that is up to fifteen digits long, excluding any punctuation (e.g., sign, decimal, currency symbol, etc.); or is not restricted to integer values and has a decimal point that may be placed anywhere from the left.

Decimal indicates a numeric value up to fifteen (15) digits in length, excluding any punctuation (e.g., sign, decimal, currency symbol, etc.); orlue is not restricted to integer values and has a decimal point that may be placed anywhere from the left of the leftmost digit to the right of the rightmost digit (e.g., +.12345678901234 is acceptable while 12345678901234567 is not).

The sign is always optional. If it is absent, the value is assumed to be positive.

Percentage

A percentage.

Amount

A numeric count including units, such as litres, inches, or kilometres per litre. For example: 150 km/h.

An *Amount* is a compound data type consisting of two logical elements:

Tag	*Type*	*Usage*	*Description*
<theAmount>	Decimal	Required	Amount.
<unit>	String	Required	Unit.

TheAmount has an attribute called *measuredConcept* that specifies what the amount measures (weight or distance for example).

Currency Amount

A monetary amount including the currency.

A *Currency amount* is a compound data type consisting of two logical elements:

Tag	*Type*	*Usage*	*Description*
<currencyAmount>	Decimal	Required	Amount.
<currencyCode>	String	Required	Currency code.

All monetary amounts in MB-XML are handled with the *Currency amount* data type. When included, this data type contains a decimal value for the amount, and an optional three-letter currency code defined in ISO-4217.

Enumeration

A value out of a limited set, each with a specific, mutually exclusive meaning.

Enumeration is a Value type that has a limited number of specified valid values, each of which is represented by a tag of up to 80 characters each.

At present, the MB-XML specification does not provide a syntax to define permitted values for an Enumeration type. Where defined, the permitted values will be found in the description of a property.

Identifier

A value without business meaning that uniquely distinguishes an occurrence.

Object reference

An identifier, unique across both space and time, with respect to the space of all Object references.

Date and Time Formats

Time is specified at the hub in a character format at the moment, with no validation expected. We highly recommend, however, that users follow the ISO8601 standard, which is the proposed standard for dates and times in XML schemas. The ISO8601 specifies time as follows:

CCYY-MM-DDThh:mm:ss.sssZ

Where CCYY is the year; MM, month; DD, day; T is the literal 'T' used as a date-time separator; hh, hours using a 24 hour clock, mm, minutes; and ss.sss, seconds.

The 'Z' indicates that the time is in Universal Time Coordinated (UTC). To indicate a timezone different than UTC, follow the time with ±hh:mm to signify the difference from UTC and omit the 'Z'.

So the following times are valid and equivalent:

2000-04-06T19:30:40Z signifying GMT and

2000-04-06T20:30:40+1:00 signifying CET and

2000-04-06T13:30:40-6:00 signifying CT (US)

ISO8601 also specifies the format for time duration. It allows the omissions of parts of the date and time. This can be useful, for instance, when we are only interested in the underwriting year or the renewal month and year. The representation allowing these omissions follows:

PnYnMnDTnHnMnS

An optional preceding '-' (before the P) may be specified to signal a negative duration. For example, P1Y2M3DT10H30M signifies 1year, 2 months, 3 days, 10 hours and 30 minutes; -P10Y means 10 years ago. The specification may be truncated on the right.

Time periods can be specified using several combinations of times, either the start instant and a duration, the start instant and end instant, or the end instant and a negative duration.

Please refer to the *ISO Standard* document for more details.

XML Schema Data Types vs. IFX Data Types

The W3C committee is defining a schema to be used for XML message definition and validation. One of the advantages of using the XML schema over DTDs is the ability to use and validate data types. The schema implementation is not yet available, however, which requires the first implementation of MB-XML to use DTDs.

We have examined the XML schema draft specification of data types, found at http://www.w3c.org/TR/2000/WD-xmlschema-2-20000225, and compared them with the IFX data types, as follows:

MB-XML Data Types	**IFX Data Type**	**XML Schema Primitive Data Type**
String	Character	String
Text		
Not supported	Narrow Character	String with a length attribute specified
Binary	Binary	Binary
Boolean	Boolean	Boolean

	YrMon	TimeInstant with the with the day, hour, minute, and second omitted
Date	Date	TimeInstant with the hour, minute, and second omitted
Time	Time	TimeInstant with the year, month, and day omitted
	DateTime	TimeInstant with seconds expressed as integers
Timestamp	Timestamp	TimeInstant with decimal values included in seconds
Amount		
Currency Amount	Currency Amount	Not supported
Not supported	Closed Enum	Supported by enumeration attribute of each data type
Enumeration	Open Enum	Supported by enumeration attribute of each data type
Short	Long	Supported by length attribute on decimal, with scale attribute of 0
Identifier	Identifier	ID
Not supported	Phone Number	Not supported
Decimal	Decimal	Decimal
Not supported	Universally Unique Identifier (UUID)	Not supported
Not Supported	URL	URI-reference

The XML schema has additional primitive data types for float (floating point number), double (double precision floating point number), time instant, time duration, recurring instant, IDREF (like IDREF in DTDs), ENTITY (like ENTITY in DTDs), and NOTATION (like NOTATION in DTDs). In addition, the XML schema contains/includes derived data types that are specialisations of primitive data types, such as language, integer, date, time, and name. It also supports user-defined data types.

The MB-XML DTDs include definitions for all valid MB-XML data types. We recognise that in future we will move from DTD specifications to using XML schema for message definitions. We recommend, therefore, that those MB-XML data types that XML schema does not support are used with great care – only on message types that are expected to be short-lived.

Furthermore, the date and time formats for IFX, although functionally equivalent, are implemented differently. The IFX timestamp specification is the same as the XML schema format. If MB-XML users choose to implement IFX conventions for other date and time formats in adapters, those adapters will have to change when the XML schema is adopted. MB-XML DTDs do not use the IFX date and time data types at this time.

