XMSG - XML Messaging Specification

W3C Note 13 October 2000

This version:
http://www.w3.org/TR/2000/NOTE-xmsg-20001013/
Latest version:
http://www.w3.org/TR/xmsg/
Editor:
R. Alexander Milowski (Lexica, LLC) <amilowski@lexica.net>

Abstract

XMSG is a specification for using XML to send messages that contain a set of XML documents, embedded non-XML data, and references to non-XML documents in a fashion that supports scalable transactions and operates on a participant model.

Status of this document

This document is a submission to the World Wide Web Consortium from Lexica, LLC (see Submission Request, W3C Staff Comment). For a full list of all acknowledged Submissions, please see Acknowledged Submissions to W3C.

This document is a Note made available by W3C for discussion only. This work does not imply endorsement by, or the consensus of the W3C membership, nor that W3C has, is, or will be allocating any resources to the issues addressed by the Note. This document is a work in progress and may be updated, replaced, or rendered obsolete by other documents at any time.

A list of current W3C technical documents can be found at the Technical Reports page.

Table of Contents

Appendices

---

1. Goals and Requirements

The following are the goals of this specification:

1. To provide the ability to transport multiple documents and references to associated data objects within a single document (a "message") and preserve their identity.
2. To provide the ability to associate metadata with both the documents and the message without modifying the original document or schemas for those

documents.

3. To provide the ability to transport non-XML data as a document within the message.
4. To provide a simple way to accomplish XML messaging.

2. A Messaging Philosophy

This XML messaging specification is based on the basic principle of providing a simple way to transport multiple XML documents within one logical XML construct without dictating any layered semantics of a messaging protocol that might be layered on top. The general philosophy is to provide the general structure upon which messaging protocols for specific business or technological purposes can be layered allowing the identification of that messaging intent but not dictating the exact syntax and semantics of the subject message. In this way, manifests, metadata, and other messaging specific constructs can be tailored to specific vertical markets or technology applications.

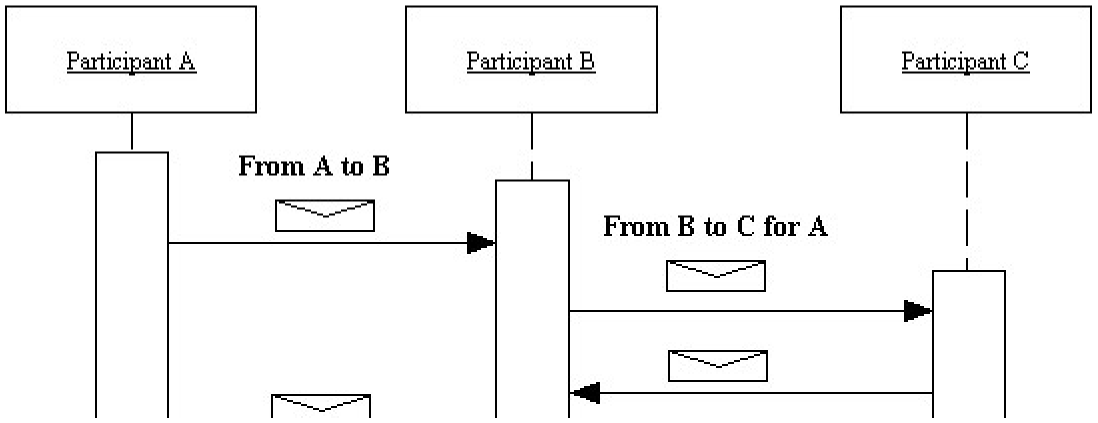
In general, the idea of an XML message presented by this specification is three-fold:

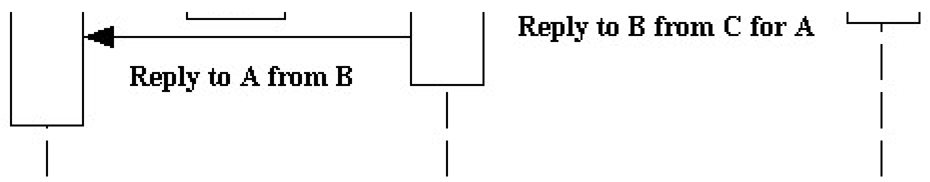
1. A pair or triplet of participants involved in the message are identified by URI values.
2. Metadata may be associated with the message itself.
3. A set of documents is contained and identified by URI allowing for document specific metadata.

These concepts are described later within the specification.

3. Participants and Messages

A message is sent from one participant (the Sender) to a second participant (the Recipient). Additionally, it might be sent on behalf of a third participant (the Originator). Essentially, an interaction (message) between two participants might require the Recipient to forward a similar message to some other participant. In this case, it is often necessary for the latter to know for whom the message is being sent. This can be modeled as follows:





Note:

> A very specific business example of this is an insurance marketplace where Insurers and Agencies communicate through a common Market. At certain points in time, the Agency may send messages to the Market which cascade into new messages being sent to the Insurer (request for quote). In that specific case, it is necessary that the Insurer know whom the message is for so that specific business-related relationships can be enforced.

Given this requirement, these participants may have the following roles:

Sender

The participant who sends the message.

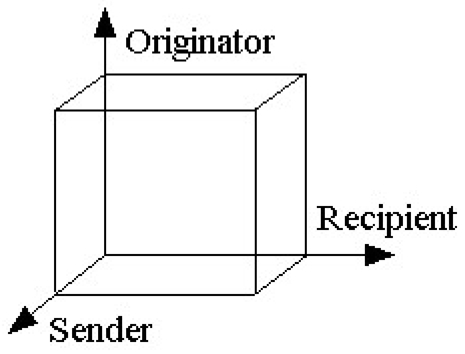
Recipient

The participant who is intended to receive the message.

Originator

The participant for whom the sender has constructed the message.

These participants can be considered to form three axes within which the message can placed.



This space is considered to be a participant space. All messages within a domain must exist within that space.

4. The Structure of a Message

A message is an XML document that utilizes namespaces. The top-level structure of the document belongs to the namespace `urn:x-lexica:xmsg:message:1.0`. Within this document, other documents--which are the content of the message--can be embedded. The message structure itself and the embedded documents are validated via their associated schemas.

4.1. The Message

A message is the top level container for an XML message with the following structure:

```
<message xmlns="urn:x-lexica:xmsg:message:1.0"
          to="{uri}"
          from="{uri}"
          id="{string}"
          generated.on="{datetime}"
          reply.to="{uri}"
          originator="{uri}"
          originator.id="{string}"
          priority="{lowest|low|normal|high|highest}"
          expires="{datetime}"
          tracking.code="{string}"
          action="{string}"
          manifest="{uri}"
          receipt.required="{boolean}"
          for.receipt="{string}"
>
<hop/> ...
<property/> ...
<document>...</document> ... | <failure/> | <receipt/>
</message>
```

The message allows the zero or more `hop` elements that record any transport hops that the message may have taken. A message may have any number of these `hop` elements inserted into the message between the location where the message was generated and where the message was received by the recipient.

Following any `hop` elements, there may be any number of `property` elements. This element provides the ability to associate any number of arbitrary properties with the message, in the form of name/value pairs. This allows application-specific metadata to be associated with the message.

Following all `hop` or `property` elements there must be one or more instances of a `document` element, a `failure` element, or a `receipt` element.

The `message` element may have the following attributes:

Attribute	Description	Required
from	The URI that identifies the sender of the message.	Yes
to	The URI that identifies the recipient of the message.	Yes
reply.to	The URI to use in the reply message for the "to" property.	No
originator	The URI that identifies whom the message was sent for by the actual sender of the message.	No
originator.id	The message identifier of the originator's initial message.	No
priority	A choice of lowest, low, normal, high or highest, with a default of "normal."	No
expires	A date and time that the message expires.	No
id	A message identifer unique to the sender.	Yes
generated.on	The date and time the message was generated.	Yes
tracking.code	An application-specific tracking code.	No
action	An application-specific "subject" or action for the recipient.	No
manifest	A URI that identifies a description of the message's contents.	No
receipt.required	An indicator that receipt message is required.	No
for.receipt	The value of previously received message receipt for which this message is a match.	No

4.2. Recording Message Hops

A message hop occurs when an intervening system handles the transport of a message on the behalf of the sender. Such a hop should be recorded within the message to provide an audit trail of what systems have participated in getting the message to the recipient. Such recording is accomplished via the `hop` element.

The structure of a `hop` tag is:

```
<hop received.on="{datetime}" received.by="{uri}" transport.used="{stri
```

Note:

The `transport.used` attribute typically has as a value common codes for standard transports like "http" or "https".

4.3. Message Properties

A message has metadata encoded as a set of properties. Some of these properties are "first-class" properties and are identified specifically by the message schema--where they are defined as attributes. Arbitrary properties may be associated with the message by using the *property* tag.

The structure of the `property` tag is:

```
    <property name="{QName}" value="{string}"/>
```

The value of the name attribute is a QName (qualified name) as defined in Namespaces in XML. In this case, the prefix used must be declared somewhere within the ancestry of the property tag where the qualified name is used.

4.3.1. The "from" Property

The *from* property is a first-class property. It identifies the sender of the message. This value is encoded as an absolute URI value.

This is a required property.

4.3.2. The "id" Property

The *id* property is a first-class property. It is a unique identifier for the instance of the message sent by the sender . The combination of the *from* property and this value should create a unique pair over time for the producer of this message (as identified by the *from* property value).

This is a required property.

4.3.3. The "generated.on" Property

The *generated.on* property is a first-class property. It is a date and timestamp for the message. It is the date and time that the message was generated by the sender.

This is a required property.

4.3.4. The "tracking.code" Property

The *tracking.code* property is a first-class property. It is an application-dependent value that is used as an identifier to track things like transactions. This value may be propagated through multiple messages.

This is an optional property.

4.3.5. The "to" Property

The *to* property is a first-class property. It identifies the intended recipient of the message. This value is encoded as an absolute URI value.

This is a required property.

4.3.6. The "action" Property

The *action* property is a first-class property. It is an application-dependent value used to further qualify the message for the recipient.

This is an optional property.

4.3.7. The "reply.to" Property

The *reply.to* property is a first-class property. It identifies the recipient of the reply to this message. This value is encoded as an absolute URI value.

This is an optional property.

4.3.8. The "originator" Property

The *originator* property is a first-class property. It identifies for whom the message is being sent by the sender. Essentially, this property captures the originator of a message chain.

This value is encoded as an absolute URI value.

This is an optional property.

4.3.9. The "originator.id" Property

The *originator.id* property is a first-class property. It is a unique identifier for the instance of the originator's message. This value is copied from the *id* property when the *from* is copied to the *originator*. The combination of the *originator* property and this value should create a unique pair over time for the originator.

This is a required property if the originator property is specified.

4.3.10. The "priority" Property

The *priority* property is a first-class property. It specifies the priority at which the message should be processed. The allowed values are "lowest", "low", "normal", "high", and "highest".

This is an optional property.

4.3.11. The "manifest" Property

The *manifest* property is a first-class property. It identifies a document in the message that is to be considered the manifest for the message. This value is encoded as an absolute URI value.

This is an optional property.

4.4. Documents

Documents can be embedded within a message by using the *document* tag. This tag allows the encoding of document specific metadata as well as the document content--which may be XML or non-XML data.

The structure of a `document` tag is:

```
<document uri="{uri}" version="{string}">
   <property/>...
   <content>...</content> | <data>...</data> | <data.reference>...</dat
</document>
```

4.4.1. Document Properties

A document has a set of metadata encoded as properties. Some of these properties are considered to be "first-class" and are encoded as attributes. Arbitrary properties may be encoded via the same property tag used for message-level metadata (See **4.3. Message Properties**).

4.4.1.1. The "uri" Attribute

The *uri* property is a first-class property and encodes the absolute URI to be associated with this document. This value must be unique to the message.

This is a required property.

4.4.1.2. The "version" Attribute

The *version* attribute specifies the version of the document. The value is an application-dependent value.

This is an optional property.

4.4.2. XML Documents

An XML document is embedded within a message as a child of the *content* tag. The namespaces of the embedded document must be declared properly such that the document maintains its functionally equivalent information set. It is not legal for the message schema namespace to be used by a contained document unless it is another message instance representing an embedded message.

Effectively, when a namespaced XML document is embedded in a message, the single element child of the *content* child becomes the document element contained within the children property and the *uri* property becomes the base URI of the document information item. In addition, the version property of the document information item defaults to "1.0" and all other properties are empty.

The structure of the *content* tag is:

```
<content>
<!-- any single element here -->
...
</content>
```

The following is an example of an embedded document:

```
<document uri="id:something">
<content>
<request.for.quote xmlns="urn:x-lexica:somemarket:request.for.quote:1.0
<item part.no="1234" quantity="1"/>
<item part.no="2234" quantity="5"/>
</request.for.quote>
</content>
</document>
```

4.4.3. Inline Non-XML Documents

Non-XML documents may be embedded within a message using the *data* tag which identifies the content type and encoding of the data. Essentially, this tag acts like an inline attachment.

The structure of a `data` tag is:

```
<data content.type="{mime-type}" encoding="{string}">...</data>
```

The encoded data must also be escaped according to the XML rules for escaping content. That is, the '<' and '&' characters must be encoded as &lt; and &amp; respectively and unicode characters must be used.

The content type of the encoded data is specified via the *content.type* attribute. This value must be a valid MIME type.

In addition, the encoding of the data is specified via the *encoding* attribute. The value of this attribute is the name of the encoding (e.g. base64) and must be understood by the receiving system.

The *content.type* and *encoding* attributes are required.

4.4.4. Out-of-Band Documents

Documents not containted within the message may also be referenced by the message. This is accomplished by use of the *data.reference* tag. This tag has the following structure:

```
<data.reference href="{uri}" content.type="{mime-type}" encoding="{stri
```

The *href* attribute is a URI value--relative or absolute--that specifies the document. The resolution of the location of this content may be transport dependent. For example, it might be the relative URI of an attached document in a MIME message (e.g. the attached filename).

The content type of the referred data is encoded via the *content.type* attribute. This value must be a valid MIME type. It is specified such that the processing system can identify the content type before traversing to the referred data.

The encoding of the referred document may be specified as well via the *encoding* attribute. Although specified, typically an transport will identify the encoding of data when the referred data is accessed. In such a case, this value may be ignored.

The *href* and *content.type* attributes are required but the *encoding* is optional.

4.5. Failures

A *failure* element encodes a messaging level failure to process a message. Typically this element is used to encode a response to a message received if that message could not be processed.

The structure of the *failure* tag is:

```
<failure on="{timeInstant}" reason.code="{string}">...</failure>
```

The content of this element is a textual message associated with the failure. The time the failure was detected is encoded in the *on*attribute--which is required.

Optionally, the sending system can encode a specific application-dependent reason code in the *reason.code* attribute.

4.6. Receipts

Receipts are returned as the reply to a message when requested by setting the *receipt.required* property to *true*. This receipt can be used to match a message received asynchronously at some later date which has a that matches the *message.tracking.code* supplied in the receipt.

The structure of the *receipt* tag is:

```
   <receipt message.tracking.code="{string}" timestamp="{timeInstant}"/>
```

When the receipt is generated, the time instant is recorded into the *timestamp* attribute. The message tracking code--as generated by the receiving system--is set in the *message.tracking.code* attribute. Both of these attributes are required.

It is necessary that the message tracking code value and the receiving system--as identified by the *to* message property--create a unique pair for the sending system which will receive the asynchronous reply.

4.7. Forwarding and Originators

When a message is forwarded on behalf of a participant, the *from* and *id* properties must be copied into the *originator* and *originator.id* properties of the forwarded message. This allows the receiving system to create a reply which can be matched via these same property values to the original message.

4.8. Creating Replies

A reply message must conform to the following rules to be considered a well-formed reply:

1. The *from* property of the reply message is set to the *to* property of the received message.
2. The *to* property of the reply message is set to the *reply.to* property of the received message if specified, otherwise it is set to the *from* property of the received message.
3. If specified, the *originator*, *originator.id*, and *tracking.code* properties of the received message are copied to the reply message.

5. Validating Messages

Messages are validated via a namespace and schema aware processor. The message itself must conform to the DTD or XML Schema provided. Each document embedded through the use of the *content* tag must be validated against the schema mapped to the namespace.

Note:

It is possible for a namespace to be validated against a DTD instead of an XML Schema. See **Appendix D. DTD Validation of Namespaces** for more information.

Appendix A. Examples

A single document message requesting a quote:

```
<message xmlns="urn:x-lexica:xmsg:message:1.0" from="urn:x-marketplace:
<document uri="id:request">
<content>
<request.for.quote xmlns="urn:x-someone:request.for.quote:1.0">
<item part.no="1234" quantity="1"/>
<item part.no="2234" quantity="5"/>
</request.for.quote>
</content>
</document>
</message>
```

A multiple document message requesting storage of the documents.

```
<message xmlns="urn:x-lexica:xmsg:message:1.0" from="urn:x-marketplace:
<document uri="id:request">
<content>
<store.to xmlns="urn:x-lexica:marketplace:verbs:store:1.0" account="A12
</content>
</document>
<document uri="id:1">
<content>
<index xmlns="urn:x-marketplace:participant:a:index:1.0">
<term href="id:2">Document</term>
<term href="id:3">XML</term>
<term href="id:4">Schema</term>
</index>
</content>
</document>
<document uri="id:2">
<content>
<termdef xmlns="urn:x-marketplace:participant:a:termdef:1.0">
<title>Document</title>
<p>...</p>
</termdef>
</content>
</document>
<document uri="id:3">
<content>
<termdef xmlns="urn:x-marketplace:participant:a:termdef:1.0">
<title>XML</title>
<p>...</p>
</termdef>
</content>
</document>
<document uri="id:4">
```

```
<content>
<termdef xmlns="urn:x-marketplace:participant:a:termdef:1.0">
<title>Schema</title>
<p>...</p>
</termdef>
</content>
</document>
</message>
```

Non-XML data encoded with base64:

```
<message xmlns="urn:x-lexica:xmsg:message:1.0" from="urn:x-marketplace:
<document uri="id:request">
<content>
<store.to xmlns="urn:x-lexica:marketplace:verbs:store:1.0" account="A12
</content>
</document>
<document uri="id:foo.txt">
<data content.type="text/plain" encoding="base64">SSBhbSB0ZXh0Lg0KSGVhc
</document>
</message>
```

Appendix B. XML 1.0 Compatibility.

XML 1.0 documents that are valid contain document types. This syntax is, obviously, incompatible with being embedded in the `document` element of the message schema. If such a document is to be transported in a message, one of the following schemes must be used:

Note:

Assume the following XML 1.0 document:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "xhtml11.dtd" >
<html>
<head><title>An XML 1.0 document</title></head>
<body>
<p>Hello world!</p>
</body>
</html>
```

1. The document is included with the doctype declaration and internal subset. Properties are set for the document using the *property* element for the system and public identifiers for the corresponding DTD. The default namespace is undeclared using xmlns="".

```
<!-- a message fragment -->
<document uri="id:something">
<property name="public-id" value="-//W3C//DTD XHTML 1.1//EN"/>
<property name="system-id" value="xhtml11.dtd"/>
<content>
<html xmlns="">
<head><title>An XML 1.0 document</title></head>
<body>
<p>Hello world!</p>
</body>
</html>
</content>
</document>
```

2. The document is included without the doctype declaration and internal subset. The default namespace is set to the system identifier of the DTD.

```
<!-- a message fragment -->
<document uri="id:something">
<content>
<!-- Note the problem in that the namespace must be mapped to the
<html xmlns="http://www.w3.org/TR/xhtml1">
<head><title>An XML 1.0 document</title></head>
<body>
<p>Hello world!</p>
</body>
</html>
</content>
</document>
```

3. The document is included without the doctype declaration and internal subset. An XML encoding is used to transport both.

```
<!-- a message fragment -->
<document uri="id:something">
<content>
<xml1.0 xmlns="urn:x-lexica:examples:xml1.0:encoding:1.0">
<doctype public.id="-//W3C//DTD XHTML 1.1//EN" system.id="xhtml11.
<internal.subset/>
</doctype>
<root>
<html xmlns="">
<head><title>An XML 1.0 document</title></head>
<body>
<p>Hello world!</p>
</body>
</html>
</root>
</xml1.0>
</content>
</document>
```

Appendix C. The Message Schema

The message schema has been specified in both an XML Schema and DTD format. These schemas are intended to be used by a processor that can associate the namespaced elements within the infoset of the message with the appropiate schema for validation purposes.

The DTD and XML Schema are avaliable online and associated documentation is provided with this specification.

Appendix D. DTD Validation of Namespaces

A DTD can be used to validate a namespace if you make the assumption that tag and type are conflated. That is, an element name is also the name of the complex type by which the element is being validated.

If, when the DTD is loaded, the namespace to be associated with the DTD is used to build qualified names out of all element names and element name references in content models, then the model can be used externally to validate the namespace. For example, given the following element declaration:

```
<!ELEMENT container (thing+) >
<!ELEMENT thing ANY >
```

when read associated with namespace `urn:x-lexica:examples:container:1.0` you would end up with the following interpretation:

```
{urn:x-lexica:examples:container:1.0}container -> {urn:x-lexica:example
{urn:x-lexica:examples:container:1.0}thing -> ANY
```

In addition, the use of ANY allows mixing of namespaces. The keyword "ANY" is defined in XML 1.0 as:

"The declaration matches ANY, and the types of any child elements have been declared."

Thus, if another namespace is included as a child of an element with a declaration of ANY and that maps to another DTD, then that child is also legal. The only restriction is that you cannot mixed namespaces within a content model. Essentially, the ANY keyword is the only construct you have to mix namespaces.

An example of using this methodology is as follows:

```
<container xmlns="urn:x-lexica:examples:container:1.0">
<thing>
<body xmlns="http://www.w3.org/1999/xhtml">
<p>Hello world!</p>
</body>
</thing>
</container>
```