AdsML® Framework for E-Commerce Business Standards for Advertising

AdsMLEnvelope 1.1.4
Part 1
Processing Model, Usage Rules & Guidelines

Document Authors: AdsML Technical Working Group

Document ID: AdsMLEnvelope-1.1.4-SpecP1Usage-AS-5

Document File Name: AdsMLEnvelope-1.1-SpecP1Usage-AS.pdf

Document Status: Approved Specification

Document Date: 30 June 2009

Draft Number: 5

Table of Contents

1 AdsML Standard Documentation

1.1 Document status and copyright

This is the Approved Specification of the AdsML® Envelope Processing Model, Usage Rules & Guidelines. It is a draft document and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use AdsML Proposed Specifications as reference material or to cite them as other than "work in progress".

1.2 Non-Exclusive License Agreement for AdsML Consortium Specifications

USER LICENSE

IMPORTANT: AdsML Consortium specifications and related documents, whether the document be in a paper or electronic format, are made available to you subject to the terms stated below. Please read the following carefully.

1. All AdsML Consortium Copyrightable Works are licensed for use only on the condition that the users agree to this license, and this work has been provided according to such an agreement. Subject to these and other licensing requirements contained herein, you may, on a non-exclusive basis, use the Specification.

2. The AdsML Consortium openly provides this specification for voluntary use by individuals, partnerships, companies, corporations, organizations and any other entity for use at the entity's own risk. This disclaimer, license and release is intended to apply to the AdsML Consortium, its officers, directors, agents, representatives, members, contributors, affiliates, contractors, or coventurers (collectively the AdsML Consortium) acting jointly or severally.

3. This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its

implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this Usage License are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the AdsML Consortium, except as needed for the purpose of developing AdsML specifications, in which case the procedures for copyrights defined in the AdsML Process document must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and will not be revoked by AdsML or its successors or assigns.

4. Any use, duplication, distribution, or exploitation of the Specification in any manner is at your own risk.

5. NO WARRANTY, EXPRESSED OR IMPLIED, IS MADE REGARDING THE ACCURACY, ADEQUACY, COMPLETENESS, LEGALITY, RELIABILITY OR USEFULNESS OF ANY INFORMATION CONTAINED IN THIS DOCUMENT OR IN ANY SPECIFICATION OR OTHER PRODUCT OR SERVICE PRODUCED OR SPONSORED BY THE ADSML CONSORTIUM. THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN AND INCLUDED IN ANY SPECIFICATION OR OTHER PRODUCT OR SERVICE OF THE ADSML CONSORTIUM IS PROVIDED ON AN "AS IS" BASIS. THE ADSML CONSORTIUM DISCLAIMS ALL WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY ACTUAL OR ASSERTED WARRANTY OF NON-INFRINGEMENT OF PROPRIETARY RIGHTS, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. NEITHER THE ADSML CONSORTIUM NOR ITS CONTRIBUTORS SHALL BE HELD LIABLE FOR ANY IMPROPER OR INCORRECT USE OF INFORMATION. NEITHER THE ADSML CONSORTIUM NOR ITS CONTRIBUTORS ASSUME ANY RESPONSIBILITY FOR ANYONE'S USE OF INFORMATION PROVIDED BY THE ADSML CONSORTIUM. IN NO EVENT SHALL THE ADSML CONSORTIUM OR ITS CONTRIBUTORS BE LIABLE TO ANYONE FOR DAMAGES OF ANY KIND, INCLUDING BUT NOT LIMITED TO, COMPENSATORY DAMAGES, LOST PROFITS, LOST DATA OR ANY FORM OF SPECIAL, INCIDENTAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES OF ANY KIND WHETHER BASED ON BREACH OF CONTRACT OR WARRANTY, TORT, PRODUCT LIABILITY OR OTHERWISE.

6. The AdsML Consortium takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available. The AdsML Consortium does not represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication, assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the Secretariat of the AdsML Consortium.

7. By using this specification in any manner or for any purpose, you release the AdsML Consortium from all liabilities, claims, causes of action, allegations, losses, injuries, damages, or detriments of any nature arising from or relating to the use of the Specification or any portion thereof. You further agree not to file a lawsuit, make a claim, or take any other formal or informal legal action

against the AdsML Consortium, resulting from your acquisition, use, duplication, distribution, or exploitation of the Specification or any portion thereof. Finally, you hereby agree that the AdsML Consortium is not liable for any direct, indirect, special or consequential damages arising from or relating to your acquisition, use, duplication, distribution, or exploitation of the Specification or any portion thereof.

8. This User License is perpetual subject to your conformance to the terms of this User License. The AdsML Consortium may terminate this User License immediately upon your breach of this agreement and, upon such termination you will cease all use duplication, distribution, and/or exploitation in any manner of the Specification.

9. This User License reflects the entire agreement of the parties regarding the subject matter hereof and supercedes all prior agreements or representations regarding such matters, whether written or oral. To the extent any portion or provision of this User License is found to be illegal or unenforceable, then the remaining provisions of this User License will remain in full force and effect and the illegal or unenforceable provision will be construed to give it such effect as it may properly have that is consistent with the intentions of the parties. This User License may only be modified in writing signed by an authorized representative of the AdsML Consortium. This User License will be governed by the law of Darmstadt (Federal Republic of Germany), as such law is applied to contracts made and fully performed in Darmstadt (Federal Republic of Germany). Any disputes arising from or relating to this User License will be resolved in the courts of Darmstadt (Federal Republic of Germany). You consent to the jurisdiction of such courts over you and covenant not to assert before such courts any objection to proceeding in such forums.

10. Except as expressly provided herein, you may not use the name of the AdsML Consortium, or any of its marks, for any purpose without the prior consent of an authorized representative of the owner of such name or mark.

IF YOU DO NOT AGREE TO THESE TERMS PLEASE CEASE ALL USE OF THIS SPECIFICATION NOW. IF YOU HAVE ANY QUESTIONS ABOUT THESE TERMS, PLEASE CONTACT THE SECRETARIAT OF THE ADSML CONSORTIUM.

AS OF THE DATE OF THIS REVISION OF THE SPECIFICATION YOU MAY CONTACT THE AdsML Consortium at www.adsml.org.

1.3 AdsML Code of Conduct

The AdsML Code of Conduct governs AdsML Consortium activities. A reading or reference to the AdsML Code of Conduct begins every AdsML activity, whether a meeting of the AdsML Consortium, AdsML Working Groups, or AdsML conference calls to resolve a technical issue. The AdsML Code of Conduct says:

Trade associations are perfectly lawful organizations. However, since a trade association is, by definition, an organization of competitors, AdsML Consortium members must take precautions to ensure that we do not engage in activities which can be interpreted as violating anti-trust or other unfair competition laws.

For any activity which is deemed to unreasonably restrain trade, AdsML, its members and individual representatives may be subject to severe legal penalties, regardless of our otherwise beneficial objectives. It is important to realize, therefore, that an action that may seem to make "good business sense" can injure competition and therefore be prohibited under the antitrust or unfair competition laws.

To ensure that we conduct all meetings and gatherings in strict compliance with any such laws and agreements in any part of the world, the AdsML Code of Conduct is to be distributed and/or read aloud at all such gatherings.

- There shall be no discussion of rates, fares, surcharges, conditions, terms or prices of services, allocating or sharing of customers, or refusing to deal with a particular supplier or class of suppliers. Neither serious nor flippant remarks about such subjects will be permitted.
- AdsML shall not issue recommendations about any of the above subjects or distribute to its members any publication concerning such matters. No discussions that directly or indirectly fix purchase or selling prices may take place.
- There shall be no discussions of members' marketing, pricing or service plans.
- All AdsML related meetings shall be conducted in accordance with a previously prepared and distributed agenda.
- If you are uncomfortable about the direction that you believe a discussion is heading, you should say so promptly.

Members may have varying views about issues that AdsML deals with. They are encouraged to express themselves in AdsML activities. However, official AdsML communications to the public are the sole responsibility of the AdsML Consortium. To avoid creating confusion among the public, therefore, the Steering Committee must approve press releases and any other forms of official AdsML communications to the public before they are released.

1.4 Document Number and Location

This document, Document Number AdsMLEnvelope-1.1.4-SpecP1Usage-AS-5, is freely available. It is located at the AdsML website at http://www.adsml-framework.org/.

1.5 Purpose of this document

This document is intended to provide guidance to implementers of AdsMLEnvelope Processors.
This document describes the envelope processing model and addressing constructs that the AdsMLEnvelope was designed to support, and defines what is required in order for an implementation to conform to the AdsMLEnvelope standard. As such, it focuses on processing and routing the AdsMLEnvelope, and does not deal with the AdsML Item-level standards.

1.6 Audience

The intended audience for this document is actual or prospective implementers of AdsML Envelope processors.

Comments on this document should be addressed to the Technical Working Group of the AdsML Consortium (technical.wg@adsml.org).

1.7 Accompanying documents

This document serves as the reference guide to the AdsMLEnvelope messages to address specific business requirements. A companion document, *AdsMLEnvelope – Part 2 - Specification & Schema*, provides additional rules and guidance for using AdsML Envelope schema. They are meant to be read together.

In addition, elements and structures that are used in multiple AdsML schemas are documented in the *AdsML Type Library* specification. AdsMLEnvelope makes extensive use of such structures, therefore the *Type Library* specification is an essential reference.

All three documents are part of the AdsML Framework, which contains a suite of related documents. Readers of this document are assumed to be familiar with the full range of relevant AdsML documentation. In particular, readers are assumed to have read the *E-Commerce Usage Rules and Guidelines* document. A description of the entire document set can be found in the *ReadMeFirst* html file associated with this release of the Framework.

1.8 Definitions & conventions

1.8.1 Definitions of key words used in this document

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are used as described in IETF RFC 2119.(S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force (IETF), Request for Comments: 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt)

When any of these words do not appear in upper case as above, then they are being used with their usual English language sense and meaning.

1.8.2 Naming conventions – element, attribute, type, and file names

All element, attribute, and type names follow the '`CamelCase`' convention.

Element and type names begin using upper camel case and begin with capitals (`UpperCamelCase`). For example, '`AdsML`', '`MessageRef`', and '`AdsMLStatusType`'.

Attribute names begin using lower camel case and begin with lower case (*`lowerCamelCase`*). For example, '*`language`*' or '*`messageId`*'.

File names also follow the camel case convention and use upper camel case for each segment of the file name, plus dashes to separate the segments of the file name. Only the first two digits of the version number are included in the file name. The third digit of the version number (if there is one) and the Draft Number are only shown internally within the document. The full naming conventions for AdsML schema and specification file names are described in the document *AdsML Document Names and Identifiers – Guidelines and Examples*, a copy of which is included in this release of the Framework.

Schema for user-defined extensions to AdsML should use AdsML naming conventions as detailed above. For example, '`ExampleInstanceFile.xml`', '`ExampleSchemaFile-1.0.xsd`', '`ExampleSchemaFile-1.1.xsd`'.

1.8.3 Typographical conventions

Element and type names are given in Courier font as, for example, `AdOrder`.

Attribute names are given in italicized Courier font as, for example, *`messageCode`*.

When citing examples of values that could be assigned to elements or attributes, the value is given in Courier font, so "…the attribute taking the value of '`12`'.".

1.9 Change History

Draft	Date	Changes	Author
1.1.4-AS-5	30 June 2009	Edited to incorporate introductory material that had previously been in the Framework Overview	TS
1.1.3-AS-4	10 October 2007	• Updated to import AdsML Type Library 2.0 • Editorial	JC
1.1.2-AS-3	1 Oct 2006	Changed to Controlled Vocabulary 3.0. No other changes.	UW
1.1.1-AS-2	1 Oct 2006	AdsML references updated to reflect Registered Trademark status	TS
1.1.0-AS-1	1 June 2006	Refactored schema to use common components from the AdsML Type Library schema. Changes affect new namespace and a few other minor schema structures leading to that 1.1 document instances are not backwards compatible with 1.0 document instances. However, no changes in functionality and processing model. Renamed "AdsML" to "AdsMLEnvelope", as "AdsML" now covers the complete framework of AdsML standards. Overviews of the AdsML approach as well as other general documentation have been moved to other non-standard specific documents.	TS, UW
1.0.1-AS-2	1 July 2005	Revised the document name and number to conform to current AdsML Framework documentation guidelines. Clarified that this document is about the AdsML Envelope rather than Item-level AdsML messages. This required many small terminology changes.	TS
1.0.0-AS-1	17 May 2004	Approved Specification. Earlier change history removed.	TS

1.10 Acknowledgements

This document is a product of the AdsML Technical Working Group.

Primary authorship and editing of the first version was performed by:

- Tony Stewart (RivCom) - tony.stewart@rivcom.com

Primary authorship and editing of the succeeding versions were performed by:

- Tony Stewart (RivCom) - tony.stewart@rivcom.com
- Ulf Wingstedt (CNetSvenska AB) - ulf.wingstedt@cnet.se

Portions are based on material written by:

- John Iobst (NAA) - iobst@naa.org

Acknowledgements and thanks to other contributors for additional input to this document are listed in Appendix A: Acknowledgement for contributions to this document.

1.11 The AdsML Consortium

The documents comprising the AdsML standard were written by the AdsML Technical Working Group, a committee charged with creating the consortium's technical deliverables, and then approved by the entire membership.

More information about the consortium can be found on the consortium's website: www.adsml.org.

2 Introduction

The AdsML Framework of E-commerce Business Standards consists of a growing set of standards that are designed to work together to implement e-commerce communications and solve specific business problems in the advertising workflow. These standards exist at two levels:

- the "AdsMLEnvelope" standard defines an XML delivery envelope that can convey any number of business messages between two trading partners;
- a set of "Item-level" e-commerce standards define XML message formats for specific types of information or transactions, for example, insertion orders, invoices or artwork.

The AdsML Item-level standards are so called because they define message formats that can be used as "Items" in AdsMLEnvelope. Use of AdsMLEnvelope, while encouraged, is optional, and the Item-Level standards can be used both inside and outside of AdsMLEnvelope.

2.1 The AdsML Envelope

The AdsML Envelope is used for exchanging and sharing information – business messages – while executing business processes during the advertising lifecycle. To achieve this exchange requires the establishment of a defined choreography between information senders and receivers. This will allow for the proper level of communication among all of the partners and make the business work smoothly.

Within a given organization, there will be a number of software systems that are each capable of processing one or more of the Item types that AdsML normally transmits – for example, booking systems, publishing systems, accounting systems, etc. From the perspective of AdsML, these software systems are "Item-level" applications, because they create and consume the Items that are transmitted inside the AdsML Envelopes.

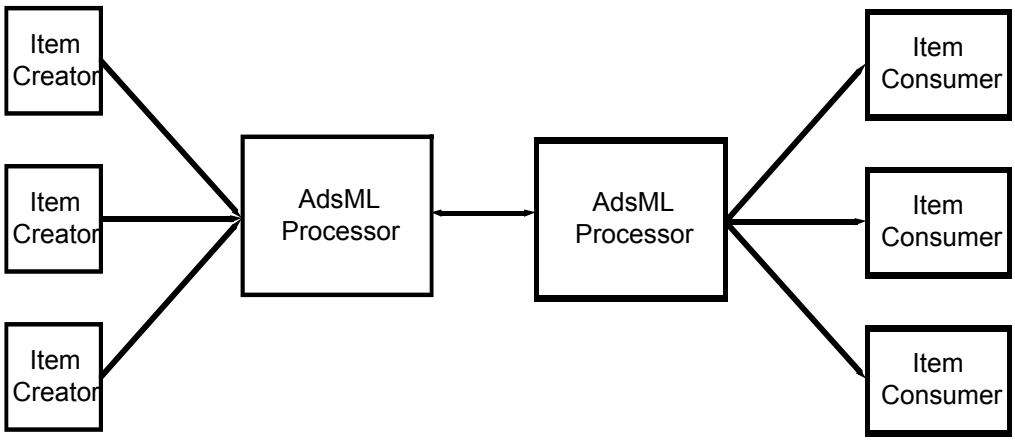
So, a given organization is likely to have a number of Item-level applications. These Item-level applications need to send information both between themselves within the organization, and also externally to Item-level applications at other organizations with which they do business.

The same organization will probably have only one AdsML processor. The AdsML processor is a software system[1] designed to process AdsML Envelopes, where "processing" includes creating, transmitting, receiving and responding to them. It will usually be positioned close to the gateway between its organization's communications systems and the outside world[2].

As shown in the illustration below, when an organization wishes to send advertising content Items to its trading partners, the Item-level applications within the organization generate the appropriate information objects, wrap them in some AdsML-specific metadata, and send them internally to the organization's AdsML processor for packaging. The AdsML processor receives these Items from the Item-level applications, packages them into one or more AdsML Envelopes (each of which

[1] The AdsML Consortium does not provide software, including AdsML Processors.
[2] It is also possible to use AdsML for communications between systems within a single organization.

is addressed to a single external organization), and then sends each AdsML Envelope to its recipient organization.

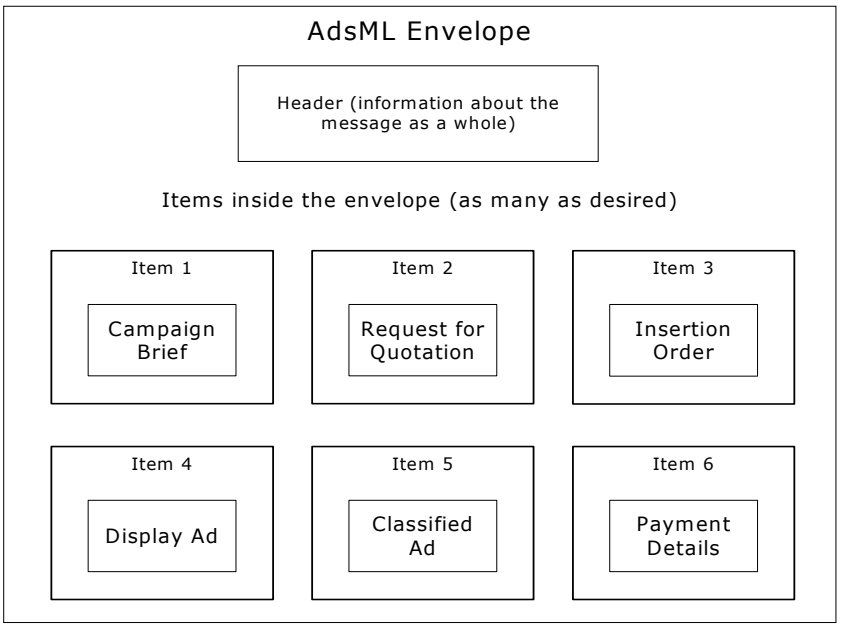


The recipient organization's AdsML Processor receives each incoming AdsML Envelope, un-packs it into its component Items, and then routes each of these Items internally to the appropriate Item-level application. Each Item that arrives at a recipient's internal Item-level application is effectively identical to the Item that was created by the sender's Item processor. At this point the AdsML Envelope no longer exists. It served its purpose by transporting the Items from the sender organization to the recipient organization.

2.1.1 Overview

Conceptually, the AdsML Envelope serves as a "wrapper", or container, that facilitates the exchange of messages (such as those defined in the Advertising Component Interactions) between trading partners in the advertising lifecycle. While it is possible that the AdsML Envelope may be used to improve communication between internal systems within a given organization, its primary purpose is to facilitate external communications between an organization and its trading partners.

Structurally, the AdsML Envelope schema defines an envelope consisting of a header and a body. The header contains information necessary to route, process, and respond to the AdsML message, and the body contains the advertising data content itself, in the form of one or more AdsML Items. The Item content can be encoded either in one of the existing advertising industry XML standards, or in a non-XML format such as an EDI standard or even a comma-delimited file. The only requirement is that the parties that are exchanging the information need to agree in advance on the format that they will use to exchange this type of information, and the format has to be able to be placed inside an XML document.



Consequently, the AdsML envelope serves multiple goals – enabling process automation, allowing continued use of existing investments in technological infrastructure, and the provision of a common interface to facilitate systems integration.

The *Envelope Processing Model, Addressing and Operational Conformance* documentation describes the types of processing that the AdsML Envelope was designed to support. It provides diagrams and descriptions of the components of an AdsML Processor, along with examples of how some of the key elements and attributes in the AdsML Envelope can be used to support common types of message processing.

2.1.2 Why this approach?

The advantage of the AdsML Envelope approach to message handling is that it helps businesses integrate and handle the complexity of exchanging information between multiple systems and multiple business partners. Rather than having to define multiple 'point to point' integrations with all of their trading partners, each partner handles this integration by implementing a single AdsML interface.

The complexity of integration – i.e. of how many formats and types of Items you are likely to receive – is controlled by the use of the AdsML Envelope interface, and by the Process Partnership Agreement (PPA) determining the information that can be sent to you.

The control is flexible at business and technical levels – the business can expand with the evolution of its PPAs. When the business requires the introduction of further systems to handle new types of information, then the single AdsML interface to the outside world remains, while giving it the freedom to extend the functionality of its AdsML processor and of the Item-level applications to which that processor is connected.

2.1.3 Conformance

The AdsML Framework includes a definition of what is required for a system built to process AdsML Envelopes to be AdsML conformant. AdsML conformance is provided at core and enhanced levels, enhanced levels successively graded by their provision of additional functionality. Providing multiple 'conformance levels' allows basic and advanced AdsML systems to be developed according to the business requirements of the individual usage scenario, enabling a system applicable to the individual circumstances to be installed while providing a basis for incremental future development.

2.1.4 Item-level message choreography

Item-level choreography defines the pattern of back-and-forth communication between the applications in each organization. This choreography is usually much more complex than the AdsML Envelope choreography, because these Items are the lifeblood of the advertisement supply chain, and their interchange defines and supports the organization's business processes. Every organization will have a number of rules indicating under what circumstances different types of business messages must be sent, acknowledged, queried, refused, altered, etc. The interplay of such rules between two organizations results in an intricate dance of messages flowing back and forth.

The AdsML Framework exists to support this Item-level message choreography, but the AdsML processor remains completely unaware of it. The AdsML processor and the AdsML Envelope format treat the Items that they carry as "black boxes", and handle them in exactly the same way regardless of whether they are advertising content, queries, responses, acknowledgements, etc. The AdsML processor acts on instructions received from the Item-level applications in a relatively simple way, adding value through its role as a form of "smart transportation", but not interacting with the underlying business processes in any other way. All of the logic and rules governing the flow of these messages must reside in the Item-level applications that create and consume them.

2.2 The AdsMLEnvelope information exchange process

AdsMLEnvelope operates on a send and response model. As the name indicates, the AdsMLEnvelope standard provides 'envelope' messages for sending information. An AdsMLEnvelope message can contain 2 types of information - 'Item(s)' and 'Response(s)'. Advertising data is exchanged in 'Item(s)'. When responding to 'envelopes' that have been received, AdsMLEnvelope message response information is exchanged in 'Response(s)'.

The extent of the response process needs to be determined as part of the Trading Partner Agreement (TPA) and Process Partnership Agreement (PPA) establishment between trading partners. It is possible to control the level of response associated with each message that is exchanged. The AdsMLEnvelope message itself will have a predetermined set of responses, and the level to which response information is exchanged between sender and recipient of an AdsMLEnvelope message can be agreed and so controlled by the terms established in their PPA.

The Trading Partner Agreement (TPA) is the legal undertaking establishing the terms and conditions under which trading partners will conduct business with one another and that the trading partners agree to abide by.

The Process Partnership Agreement (PPA) specifies the information that must be available to an AdsMLEnvelope Processor on a "per partner" basis. That is, for each partner organization or sub-organization with which an organization agrees to exchange AdsMLEnvelope messages, the information required by the PPA must be known and available to the appropriate AdsMLEnvelope Processor(s).

2.3 Item and Response – operational and messaging data

As the preceding sections show, the AdsMLEnvelope message is essentially a management structure for the exchange of advertising data among the interested parties. That advertising data is carried either inside `Item` element(s) or inside `Response` element(s) and falls into two categories, **operational data** and **messaging data** respectively. Operational data is the actual advertising data being exchanged and will be carried as `ContentData` nested inside an `Item`. Messaging data is AdsML response data that is used in the message exchange and monitoring process and will be carried as `ResponseContent` nested inside a `Response`. An AdsMLEnvelope message may contain multiple `Item` elements or a single `Response` element.

2.3.1 Operational data – advertising content data

The operational data (carried inside `Item` element(s)) comprises the advertising data that business partners have to exchange with one another as part of the execution of the advertising processes and workflows required to create and publish an advertisement from initial placement to final airing.

The AdsMLEnvelope specification does not prescribe any specific format(s) for the operational data itself; rather it provides an envelope for exchanging that data. It is thus open for any kind of text-based formats for operational data that can be carried inside an XML message without harming its well-formed structure. It is assumed that most formats will be XML based, but also legacy formats such as comma-separated files can be handled.

AdsMLEnvelope messages do not directly transfer binary objects as operational data. A binary object would have to be wrapped in some kind of metadata rich format that would either include the object encoded into XML text, or include a reference to an external resource.

2.3.2 Messaging data – AdsMLEnvelope response data

The messaging data (carried inside `Response` element(s)) is AdsMLEnvelope response data – AdsMLEnvelope technical data used for functions like acknowledging the receipt of or reporting an error with an AdsMLEnvelope message or during testing. Messaging data applies to AdsMLEnvelopes messages themselves, is used to communicate responses at the envelope level, and is only concerned with AdsMLEnvelope messages. Messaging data has no relevance to `Item` level data, and so contains no references to `Item`s; it should not be confused with `Item` level response data. `Item` level response data – for example a reply acknowledging the

receipt of an insertion order or requesting a resend of artwork - is operational data and so would be carried inside `Item` elements and **not** in `Response` elements.

2.3.3 Testing

AdsMLEnvelope allows a message sender to send a test message to one or more receivers and receive appropriate test responses in return. Testing is at the envelope level and tests the AdsMLEnvelope message 'envelope' and the AdsMLEnvelope message response mechanism. Responses **MUST** be sent during testing.

To test an AdsMLEnvelope message, a message is sent and identified as an envelope test by setting the value of the *`status`* attribute of the message's `Header` element to '`EnvelopeTest`'. When responding to an envelope test, a message is sent and identified as a response to the envelope test by setting the value of the *`status`* attribute of the `Header` element to '`ResponseToEnvelopeTest`' and by also specifying a `Response` element child to the `AdsMLEnvelope` root element of the message, assigning the value of the *`status`* attribute of the `MessageRef` child element of the `ResponseHeader` element the value of '`EnvelopeTest`'.

2.3.4 Globally unique identifiers

AdsML messages (of all types) and any individual `Item`(s), `ItemContent`(s), or `Response`(s) contained inside an AdsML Envelope **MUST** have a globally unique identifier. These identifiers are defined as `QIDType` types, defined in the AdsML Type Library. The structural rules for these values are described in the AdsML *E-Commerce Usage Rules & Guidelines* document.

The globally unique identifiers are recorded as attribute values, as follows,

- AdsMLEnvelope – a *`messageId`* attribute on the `Header` element records a globally unique identifier for the AdsMLEnvelope message
- Item– an *`itemId`* attribute on the `Item` element records a globally unique identifier for an item
- Item content –
- An *`itemContentId`* attribute on the `ItemContent` element records a globally unique identifier for item content inside an item
- Error reporting content -
- An *`itemInEnvelope`* attribute on the `ErrorLocation` element identifies the `Item` in which an error is located by recording the value of the *`itemId`* of the `Item` in question
- Response –
- A *`responseId`* attribute on the `Response` element records a globally unique identifier for a response
- A *`messageId`* attribute on the `MessageRef` child element of the `ResponseHeader` element identifies the AdsMLEnvelope message to which a response refers by recording the value of the *`messageId`* of the message being responded to.

2.4 How AdsMLEnvelope relates to other standards relevant to the advertising process

As stated, the AdsMLEnvelope provides a framework in which operational data is carried as data content in `Item` elements. This operational data will be represented using a variety of content formats that are 'wrapped' by the AdsMLEnvelope Item layer for transmission between parties during the execution of advertising processes. In addition to formats such as AdsMLBookings and AdsMLMaterials developed within the AdsML effort, also earlier established industry formats for representing operational data can be used such as:

- CREST. Developed by the Classified Advertising Standards Task Force of the Newspaper Association of America (NAA, http://www.naa.org)), CREST 2.0 is an XML-based media independent format for electronically exchanging and sharing classified advertising data. CREST focuses on the three main areas of classified advertising - real estate, transportation, and employment categories, and provides a generic extension mechanism to record advertising data that falls outside these categories.
- IfraAdConnexion. Developed by Ifra (http://www.ifra.com), IfraAdConnexion is an XML-based vocabulary for the newspaper industry, the vocabulary concentrating on the ad booking and ordering processes.
- IfraTrack. Developed by Ifra, IfraTrack is an XML-based specification for the interchange of status and management information between local and global production management systems in newspaper production.
- JDF. Developed by CIP4 (http://www.cip4.org/) the Job Definition Format (JDF) is an XML-based job ticket format used to create end-to-end job tickets for a print run. JDF facilitates information exchange and facilitates integration and interoperability among workflow systems.
- SPACE/XML. Developed by IDEAlliance (http://www.idealliance.org), the XML-based Specification for Publisher and Agency Communication Exchange (SPACE/XML) is a standard for sending and acknowledging advertising space reservations, insertion and change orders, invoicing, and advertising copy data files between advertising agencies, prepress or prepress media services, printers, and publishers.

3 AdsMLEnvelope choreography

The message choreography between any two organizations that have implemented AdsMLEnvelope occurs at two different levels: between their AdsMLEnvelope Processors, and between their Item level applications.

The AdsMLEnvelope-level choreography is relatively simple, and is defined in the AdsMLEnvelope specification and the Process Partnership Agreement. Each exchange of AdsMLEnvelope messages begins when one AdsMLEnvelope processor sends an AdsMLEnvelope to another AdsMLEnvelope processor, which is usually at a different organization. Under certain defined circumstances, the second processor will send an Response message back to the first one. In some cases, the first processor will re-send an AdsMLEnvelope message for which no satisfactory response has been received.

Although the AdsMLEnvelope-level choreography can get more complicated than this, it is not particularly difficult. Its purpose is administrative and technical: it supports the communication of the Items that are contained within the Envelopes.

3.1 Envelope Response Messages

AdsMLEnvelope includes provisions both for the packaging and routing of business messages (Items) inside an AdsMLEnvelope message, and for the transmission of an Response message from the recipient's AdsMLEnvelope processor to the sender's AdsMLEnvelope processor. Common types of responses include:

- Acknowledgement that a given message was received and successfully processed
- Notification that a message contained structural or syntactic errors such that it could not be processed
- Notification that one or more of the Items within a message violated relevant provisions of the Process Partnership Agreement between the sender and recipient, for example, because the Item used a standard or format that the recipient is not able to process.

The AdsMLEnvelope standard specifies a set of facilities and rules that determine when a Response message is optional (may be sent) or mandatory (must be sent) by the recipient. For example, a response must always be sent if the message failed syntactic or structural validation, or if it contained Items that violated the rules in the relevant PPA, or if the sender of the message specified in the message header that a response is required.

3.1.1 Envelope addresses

For security reasons, the AdsMLEnvelope format does not include a "reply-to" address: the only indication in an AdsMLEnvelope of who sent the message is the ID of the sending organization. (This is intended to prevent third parties from "spoofing" the system, by intercepting an AdsMLEnvelope message and altering its reply-to address.) So in order to send Response messages to their correct addresses, the AdsMLEnvelope processor must "look up" the organization's ID in its system and retrieve the address to which messages intended for that organization should be

sent. This addressing information is part of the Process Partnership Agreement between the two organizations.

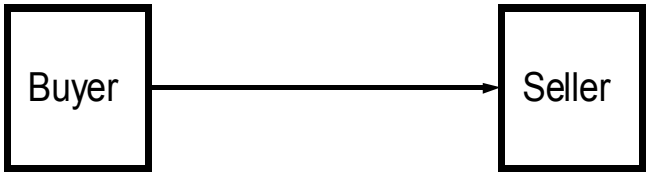
3.2 Envelope exchange paths

The AdsMLEnvelope supports two main categories of exchange paths that an Item may take. For this discussion they will be called Direct and Indirect.

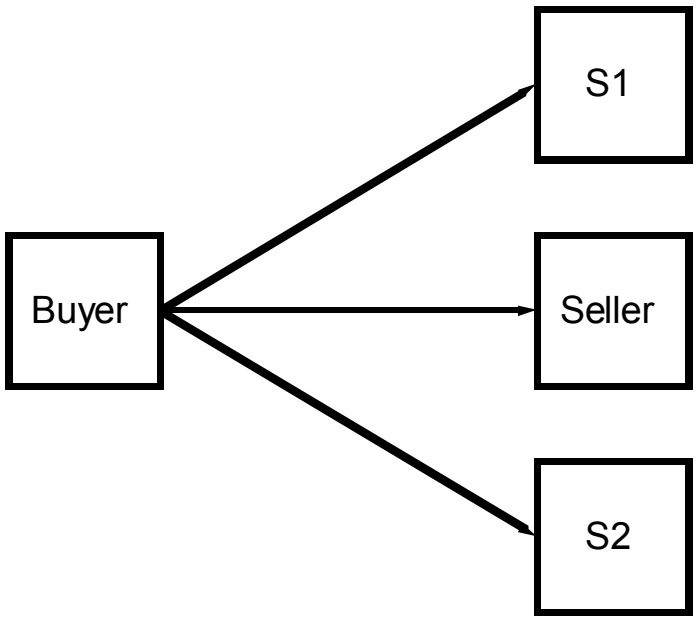
3.2.1 Direct exchanges

The direct exchange path has the AdsMLEnvelope Item moving directly to all targeted recipients. There are no intermediate parties to the exchange process, with the single possible exception of a store-and-forward action on one or both sides of the process. (Note that the message is not opened or processed by the store-and-forward intermediary. The entity performing a store-and-forward operation, if present, functions solely as a message traffic manager.)

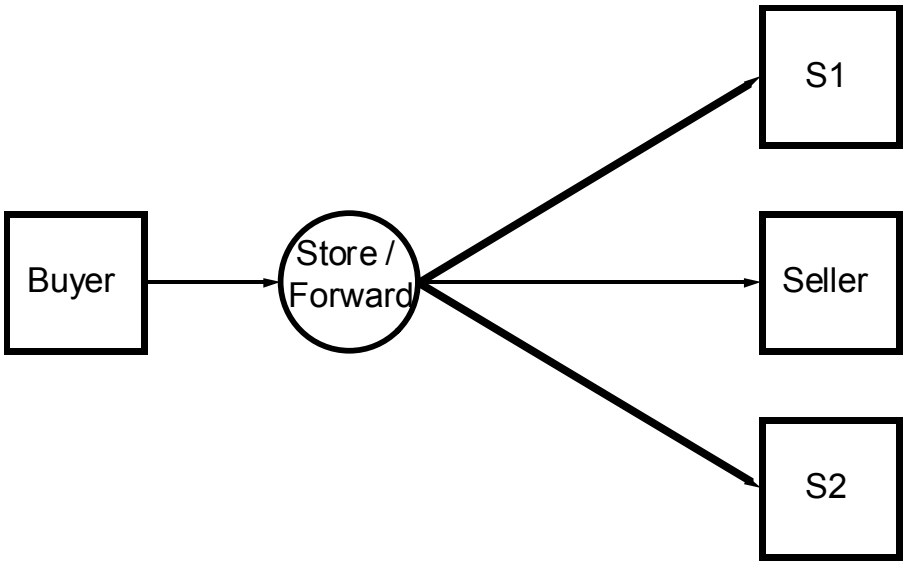
The following diagram illustrates direct transmission from buyer to seller.



The following diagram illustrates the transmission from buyer to seller with additional transmissions from the buyer to S1 and S2. S1 and S2 are secondary partners to the transaction.



The following diagram illustrates the transmission process using a store and forward intermediary, for example, when one organization acts as the gateway for another in the sending or receiving of advertising information. The intermediary is represented by a circle to indicate that it is not involved in message processing.

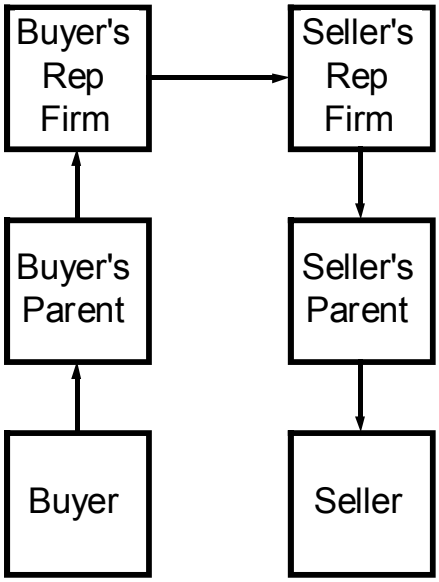


3.2.2 Indirect exchanges

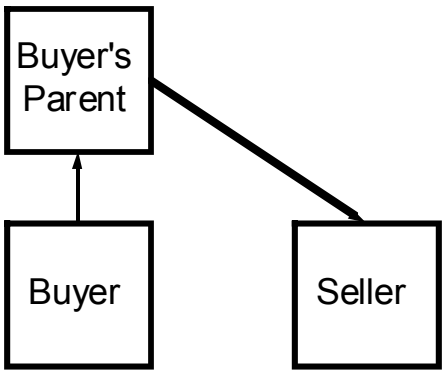
The indirect exchange path has the Item moving through one or more intermediaries. These intermediaries may perform a simple aggregation process or actually perform an action on the contents of the Item.

In the indirect exchange process, a transaction may follow a path with steps like: buyer -> buyers parent -> buyers rep firm -> sellers rep firm -> sellers parent -> seller. By example in the U.S. market: AllState Insurance -> Sears -> NSA -> NNN -> Newhouse Newspaper -> Newark Star Ledger.

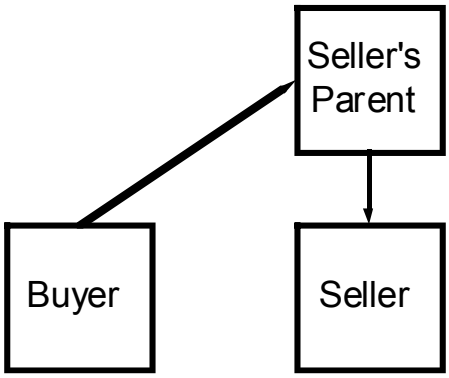
The following diagram illustrates the message path for the indirect process. Store-and-forward can appear at any part of the process and steps can be skipped.



This process exposes interesting issues of how to get from the buyer to the seller. Each of the steps in the chain is well known but may not be known to the end points. The buyer may think that the path is:



And the seller may think the path is:



The reality may be something else entirely.

AdsMLEnvelope provides a number of facilities that are intended to support each of these exchange scenarios. A description of these facilities can be found in the Processing / Addressing / Conformance documentation.

3.3 Repackaging

When an AdsMLEnvelope Item is sent through an intermediary several things may happen:

- The intermediary may perform a straight store-and-forward on the entire message containing the Item without processing any of its content
- The intermediary may explode the message into its constituent Items, and then repackage those Items into other AdsMLEnvelope messages without having processed any of the Items
- Or it may explode the message into its constituent Items, perform some kind of processing on those Items, and then create new AdsMLEnvelope messages in which those Items are sent on to their next destinations.

4 Processing Model and Routing Scenarios

AdsML provides both a definition of an XML document type (the AdsMLEnvelope) and a framework for packaging and routing the business information that is carried as Items within AdsMLEnvelope messages. The format of an AdsMLEnvelope message is designed to support very specific types of processing.

This section describes the processing model for which the AdsMLEnvelope format was designed, and the routing scenarios that it is intended to support. It contains both illustrative descriptions of the processing model and routing scenarios, and specific examples of how the AdsMLEnvelope schema should be used when realizing those scenarios.

Please note that neither the processing model nor the routing scenarios are normative parts of the AdsMLEnvelope standard. They are included for illustrative and exemplary purposes only. AdsMLEnvelope implementers are free to develop systems that deviate from this processing model, provided that those systems, and the messages they generate, conform to the requirements that are described in this document.

4.1 Terms and concepts

The description of the AdsMLEnvelope Processing Model relies upon the following terms and concepts:

Business Entity: An organization or a subset of an organization that sends and/or receives AdsML messages.

Content Creator/Consumer: A software application that creates or consumes information that needs to be sent as an Item within an AdsMLEnvelope. For example, this could be a booking system, an accounting system, a graphics package, etc. A Content Creator/Consumer will always play two roles:

> **Content Creating Application**: The role played by a Content Creator/Consumer when it creates information that needs to be sent to another system as an Item within an AdsMLEnvelope.
>
> **Content Consuming Application**: The role played by a Content Creator/Consumer when it consumes the information that was sent to it from another system as an Item within an AdsMLEnvelope.

Content: Advertising-related information created by a Content Creating Application. This could be a media pack, an insertion order, a publication plan, an advertisement, an invoice, etc.

Trading Partner Agreement (TPA): An agreement between two business entities that covers the terms of their business.

Process Partnership Agreement (PPA): Those aspects of the trading partner agreement between two business entities that are accessible to their AdsMLEnvelope processors.

Note: Although the concept of a PPA is central to AdsML, the PPA itself is not formally defined in AdsML. The absence of a formal definition is not expected to prevent users from implementing AdsML systems.

Trusted Sender: An external business entity from which a given entity is willing to receive AdsML Envelopes.

AdsMLEnvelope Processor: A software application that is capable of creating, sending, receiving and responding to AdsMLEnvelope messages. An AdsMLEnvelope Processor can play several roles:

- **Item Redirector**: The role played by an AdsMLEnvelope Processor when it reads the addressing information in an Item and, if appropriate, redirects the Item to an intermediary destination.

- **Envelope Packager**: The role played by an AdsMLEnvelope Processor when it assembles a set of Items or a Response into an AdsMLEnvelope and sends that Envelope to its next recipient.

- **Envelope Receiver**: The role played by an AdsMLEnvelope Processor when it receives an AdsMLEnvelope message that was sent to it by another AdsMLEnvelope Processor.

- **Item Content Unpacker**: The role played by an AdsMLEnvelope Processor when it converts the contents of an Item back into that Item's original format so that it can be processed by the appropriate Content Consuming Application.

- **Response Creator**: The role played by an AdsMLEnvelope Processor when it creates a Response to an AdsMLEnvelope message that it has received.

- **Response Processor**: The role played by an AdsMLEnvelope Processor when it processes a Response that was sent to it by another AdsMLEnvelope Processor.

  Note: Nothing in this document is intended to constrain the technical design or structure of the AdsMLEnvelope Processor, including, for example, whether it consists of a single monolithic application or a set of loosely coupled services each of which performs a different role. Similarly, it is possible that an Item Redirector could be a separate software application, or a role played by an Item Creator.

Item Creator: A software application that converts the content provided by a Content Creating Application into an AdsMLEnvelope Item.

Note: Nothing in this document is intended to constrain the technical design or structure of the Item Creator or an Item Content Unpacker. In particular, it is possible that the Creator and Unpacker functionality for a particular kind of information could be combined into a single "AdsMLEnvelope Adaptor" that binds to an application that both creates and consumes that kind of information. In this case the Adaptor would play two roles: Item Creator and Item Content Unpacker.

Originating Entity: The business entity that creates the content that becomes an AdsMLEnvelope Item.

Sending or Sender Entity: The business entity that sends an AdsMLEnvelope. Message. The Items contained within a given message may or may not have been originated by the Sender of that Envelope.

Destination Entity: The business entity that is the intended consumer of the content that was created by the Originating Entity.

Receiving or Recipient Entity: The business entity that receives an AdsMLEnvelope message. The Items contained within a given message may or may not be destined for that Recipient.

Redirection: An action performed by an Item Redirector when it intervenes in the routing of an Item and sends it to a business entity other than the one that the Content Generating Application specified as its Destination Entity.

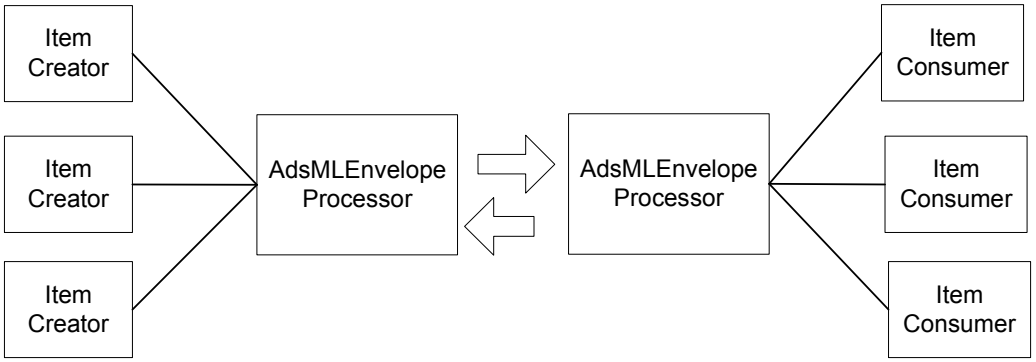
Intermediate Entity: A business entity that is not the originally stated "Destination" of an Item, to which that item is redirected. The intermediate entity might merely "store and forward" the Item without processing its content, or might provide a service that involves validating or acting upon that content.

4.2 AdsMLEnvelope architecture

This section describes an expected "typical" high-level architecture of an AdsMLEnvelope installation. It is for illustrative purposes only.

4.2.1 Overview

The following high-level illustration serves as a good starting point for this discussion:



The above diagram shows that:

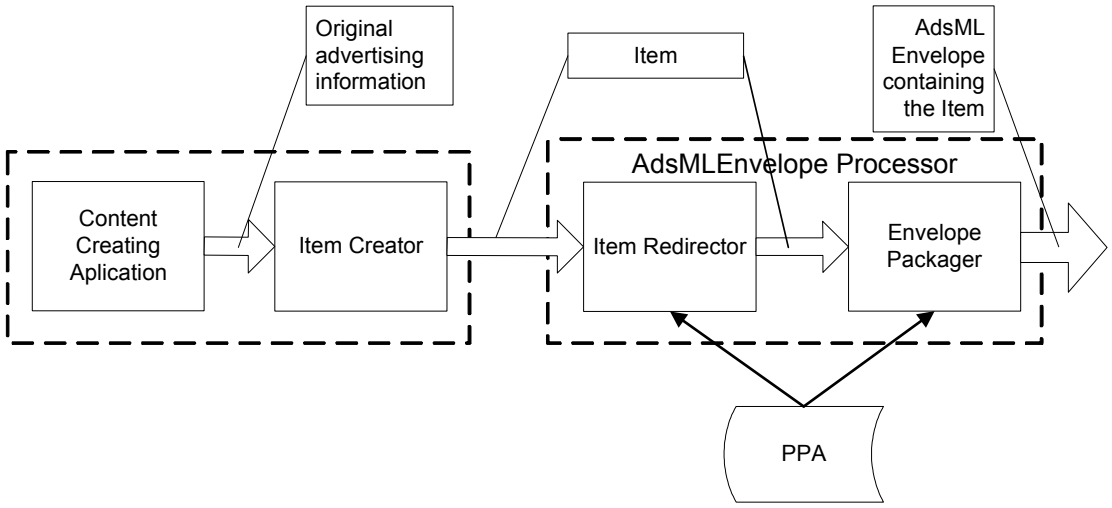
- Multiple Content Creating Applications generate content that can be packaged by an AdsMLEnvelope Processor into an Envelope, which is then sent to an AdsMLEnvelope Processor at a different business entity.
- The second AdsMLEnvelope Processor unpacks the Items from the Envelope and routes them to multiple Content Consuming Applications.

Notes:

- The actual transport of the Envelope over a physical network is not within the scope of AdsML. Rather, AdsML assumes a common infrastructure of networks, protocols and security and will use this infrastructure for secure transmission of business messages. The envelope processor talks to the transport layer, which may send or receive the Envelope by any appropriate method and following any appropriate communication model (e.g. "push" or "pull" and their many variants).
- Each AdsMLEnvelope processor is assigned a persistent ID that is unique within the context of the business entity that controls this processor. The Processor ID is recorded in the header of all AdsMLEnvelope messages that are created by that processor.

4.2.2 Content creation – single source

If we take a closer look at the process of extracting advertising content from the application that created it and packaged it in an AdsMLEnvelope (shown on the left side of the above diagram as a single flow from Item Creator to AdsMLEnvelope Processor), we see that actually it contains several parts:



The above diagram shows that:

- The Content Creating Application generates some content and provides it to the Item Creator.
- The Item Creator converts the content into a valid Item by encoding and optionally encrypting it, and fills in the Item-level metadata with values that it extracts from the source content. In order to do so, it needs to be closely associated with the Content Creating Application and have knowledge about the internal structure of the source content (as indicated by the box that surrounds them).

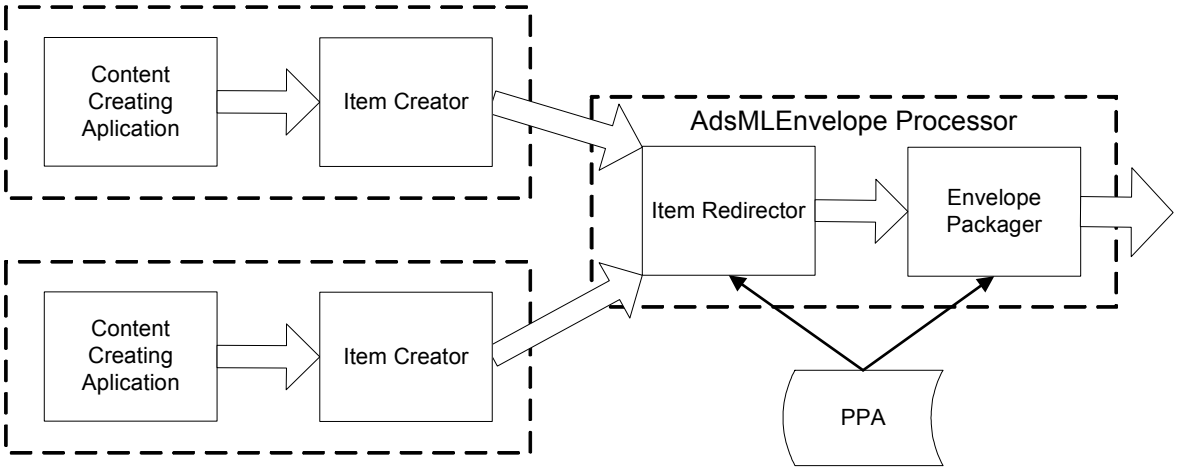
- The Item is then optionally processed by an Item Redirector, which uses rules stored in the Process Partnership Agreement (PPA) between the current Sender and the Item's intended destination to determine whether the Item should be redirected to an intermediary destination.
- The (possibly redirected) Item is processed by the Packager, which places it inside an AdsMLEnvelope along with zero or more other Items that are going to the same place. The Packager uses the addressing information from the PPA to determine the transport mechanism and physical address to which the Envelope should be sent.
- The AdsMLEnvelope is sent to its destination.

Notes:

- An Item Creator needs knowledge about a Content Creating Application's source content. Therefore, there is likely to be a separate Item Creator associated with each type of Content Creating Application.
- The Content Creating Application can be expected to have an addressing mechanism already in place, perhaps with an application-specific catalog structure. Depending on the needs of the host organization, either the AdsMLEnvelope Processor and PPA can reference the address catalog used by the Content Creating Application, or the Item Creator must translate the addresses generated by the Content Creating Application into the format used by the AdsMLEnvelope Processor.
- Item-level metadata values filled by the Item Creator include the Item's type, format, priority and intended destination, as well as the type of encoding and/or encryption that was applied to it.
- Redirection is never arbitrary: it occurs because of a business agreement between the entity that is sending the Item and its intended Destination entity.
- In order to perform its redirection and addressing functions, the AdsMLEnvelope Processor must have access to all of the PPAs that its host entity has entered into.

4.2.3 Content creation – multiple sources

Within a given organization, there may be multiple Content Creating Applications, in which case the architecture would look more like this:

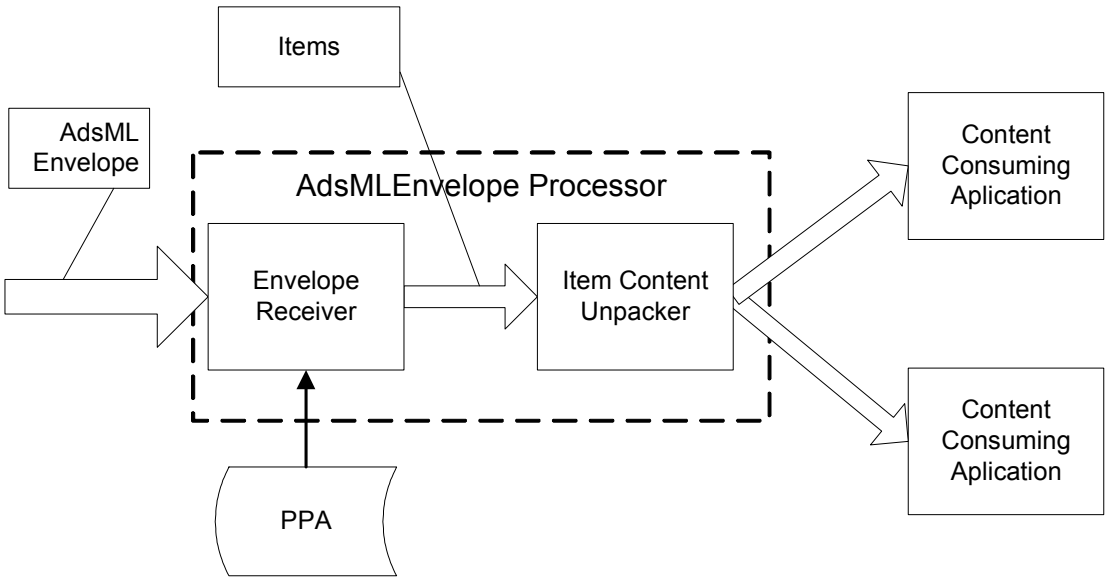


This diagram shows that:

- When there are multiple Content Creating Applications, each is associated with a different Item Creator.
- The output of these Item Creators is a set of Items, which can all be sent to a single Item Redirector for possible redirection, because redirection is a generic process that requires only the metadata contained in the Item and the rules contained in the relevant PPA as inputs to its operation.
- The Items are then sent onwards to the Packager, which assembles those Items that are addressed to the same business entity into a single AdsMLEnvelope and sends that envelope to its destination.

4.2.4 Content reception

The process of receiving, validating and unpacking incoming advertising information looks like this:



This diagram shows that:

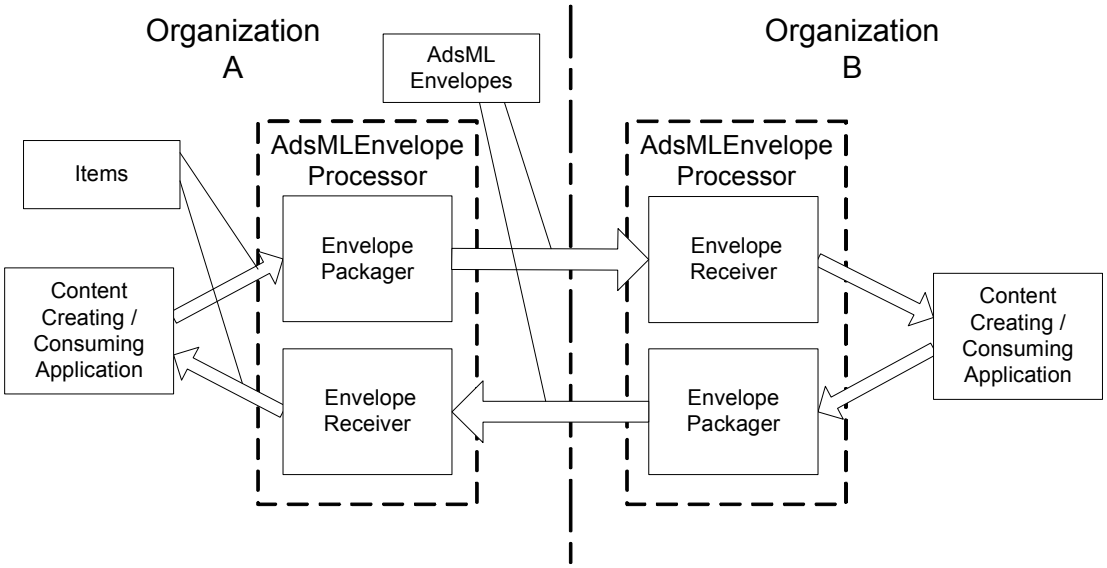
- An Envelope Receiver receives the incoming AdsMLEnvelope message.
- The Receiver validates the Envelope against the terms and conditions stored in the relevant PPA, and (assuming that all is ok), extracts its component Items and sends them to the Item Content Unpacker.
- The Item Content Unpacker converts the contents of the Items it receives into a form that is functionally equivalent to the content that was originally generated by the Content Creating Application at the start of the process. It then passes this content to the Content Consuming Application with which it is associated.
- The Content Consuming Application consumes (acts upon) the content.

Notes:

- Actions performed by the Receiver include verifying and validating that the structure and metadata of the AdsMLEnvelope conform to the standard, and generating any Responses that need to be sent to the sending business entity.
- Actions performed by the Item Content Unpacker include extracting the content from inside the Item, reversing any encoding or encryption that was performed by the Item Creator, and routing the content to the Content Consuming Application.
- This architecture assumes that all of the Item-level metadata associated with an Item disappears when the Item is unpacked, so a single Item Content Unpacker can service many different Content Consuming Applications. However, it is possible to envision a different architecture in which there would be multiple Item Content Unpackers, one for each type of Content Consuming Application.

4.2.5 Two-way information flow

The above detailed diagrams illustrate a simplified one-way information flow. In fact, as shown in the Overview diagram at the start of this section, all information flow actually occurs in both directions. An AdsMLEnvelope Receiver may generate a Response message that goes back to the originating AdsML system, and a Content Consuming Application may generate content that must be routed to the originating Content Creating Application.



This diagram shows the round-trip information flow between two Content Creating/Consuming Applications in two different organizations, except that for purposes of clarity the Item-level packing and unpacking operations are not shown. The diagram is meant to illustrate that:

- All information flow occurs in both directions
- The Content Creating Application also acts as a Content Consuming Application
- An AdsMLEnvelope Processor will perform both Packaging and Receiving roles

Note:

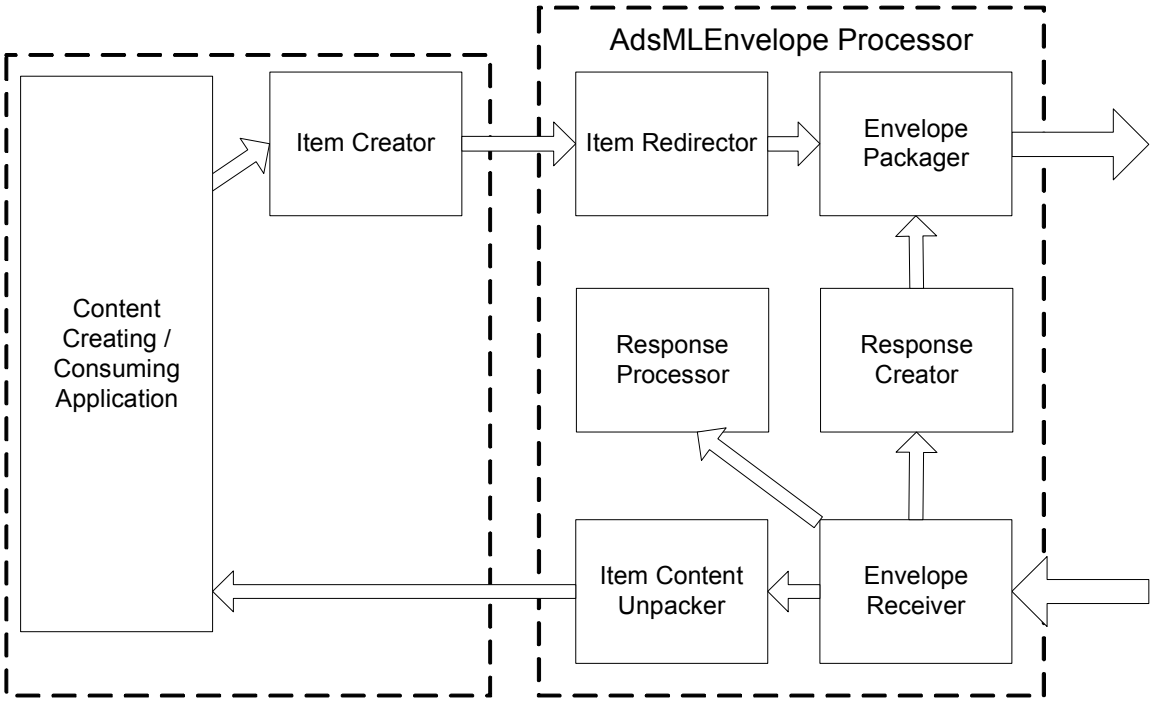
- It is not required that a Content Creating Application must also be a Content Consuming Application, or vice versa.

4.2.6 Role playing

If we "zoom in" on Organization A in the above diagram, we discover that in order to create and process both Items and Responses as shown in the previous diagrams, an AdsMLEnvelope Processor needs to play multiple roles, some of which were not shown above. The six roles played by an AdsMLEnvelope Processor are:

- Envelope Packager
- Envelope Receiver
- Item Content Unpacker
- Item Redirector
- Response Creator
- Response Processor

This diagram provides an overview of all of these roles:

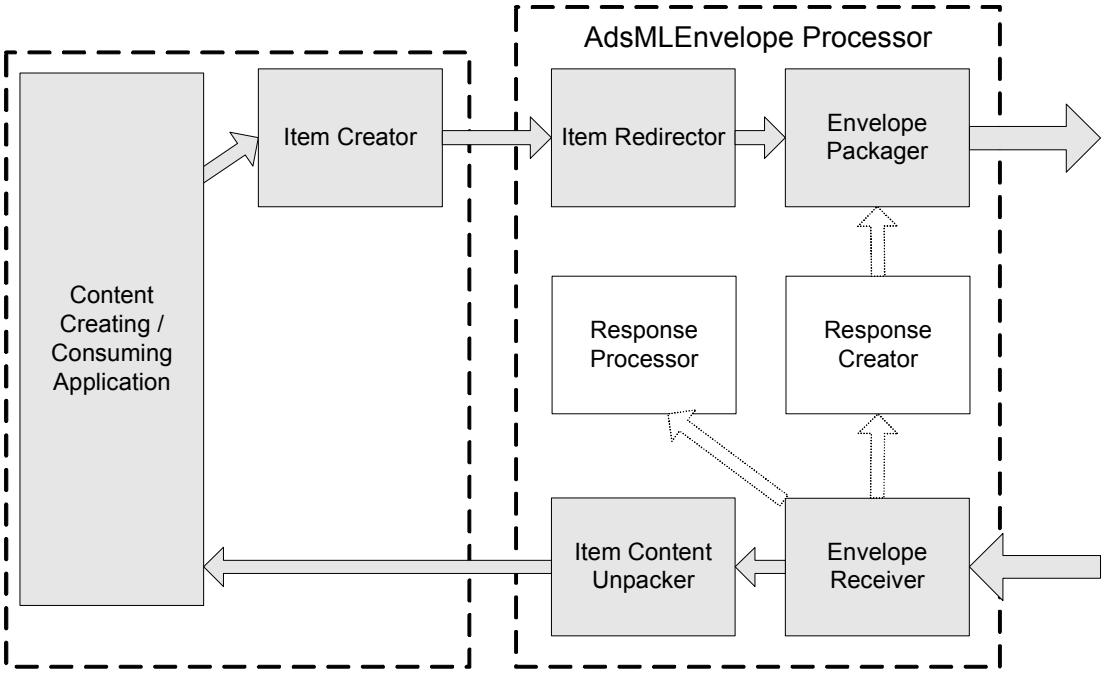


(An equivalent illustration for Organization B would simply be a mirror image of the diagram for Organization A.)

The following sections contain versions of the above diagram that illustrate these roles and relationships in more detail. In each case, the objects under discussion are shown in gray with solid outlines. All non-gray objects with dashed outlines should be ignored.

4.2.6.1 Role playing – Item processing

This diagram shows that the Content Creating/Consuming Application is directly associated with an Item Creator and also receives content from the Item Content Unpacker:

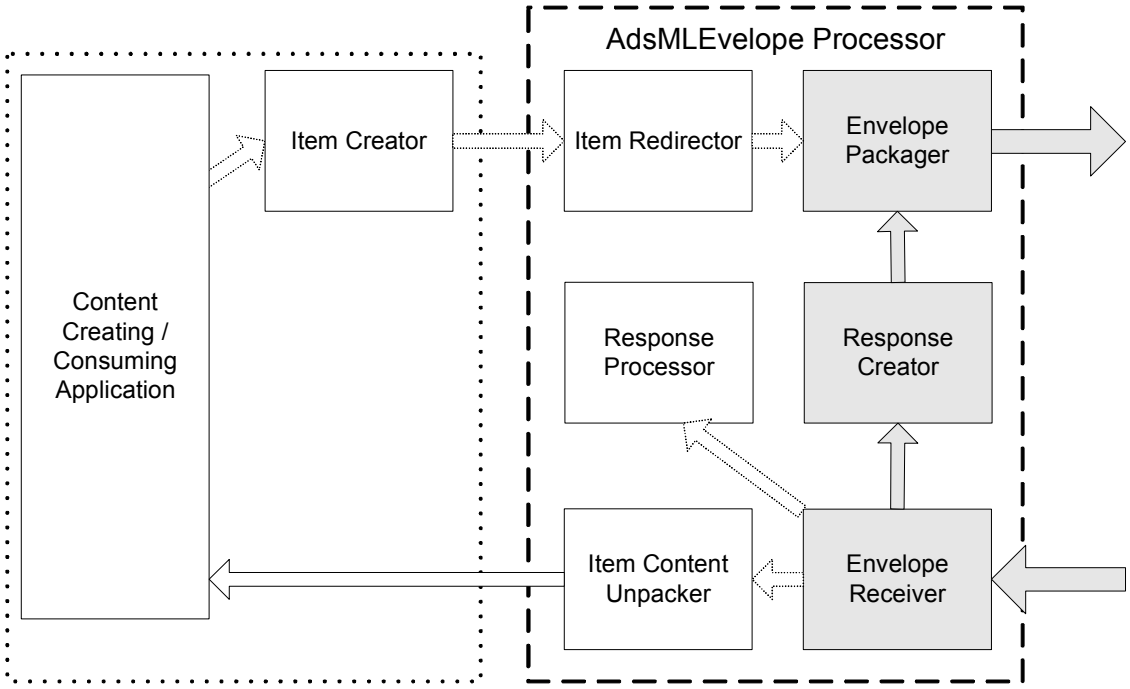


The main point of this diagram is that in order for a business entity to implement an AdsMLEnvelope system, it needs two categories of software:

- Each Content Creating Application needs to be associated with an Item Creator that is capable of dealing with outgoing information content.
- The business entity as a whole needs to have at least one AdsMLEnvelope Processor that is capable of performing the Packaging/sending role, the Receiving/validating role, and the Item Content Unpacking role. This will usually be a single piece of software.
- If the entity wishes to implement address redirection, the architecture needs to contain an Item Redirector. This may either be a fourth role played by the AdsMLEnvelope Processor, as shown above, or a standalone application.

4.2.6.2 Role playing: Response generation

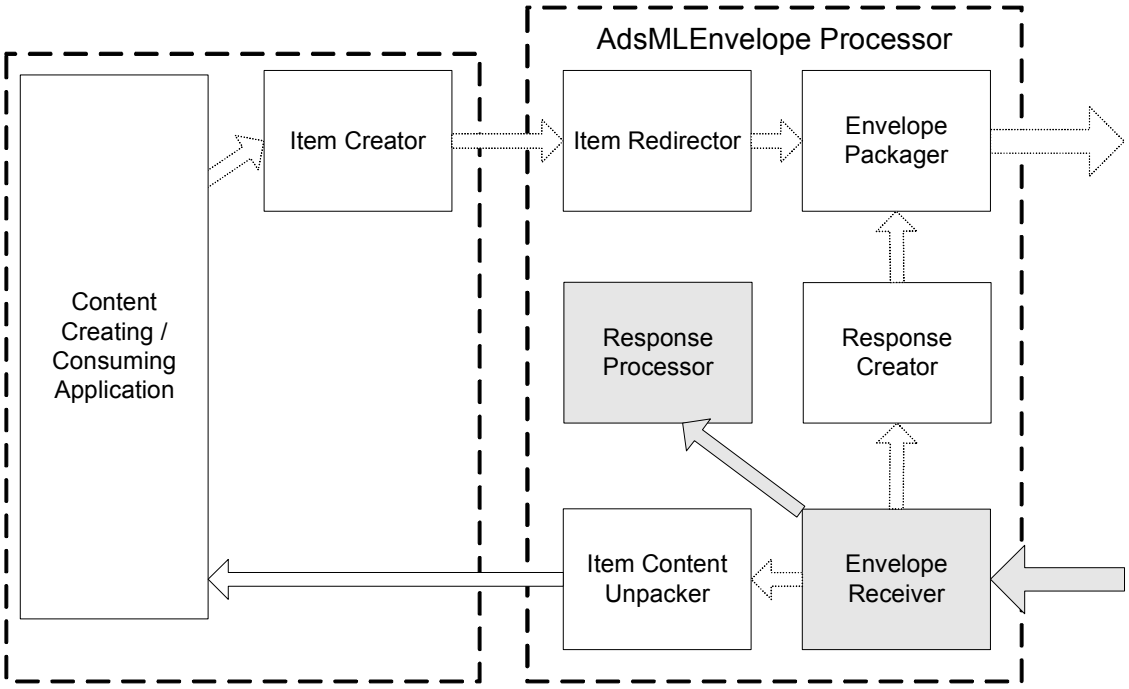
This version of the diagram emphasizes the roles that an AdsMLEnvelope Processor plays when responding to an incoming AdsMLEnvelope that contains Items.



- When the Envelope arrives, the Envelope Receiver verifies and validates the integrity of the Envelope and the Items within it. If errors are discovered, or if the governing Transmission/Response Mode requires sending a response to all incoming messages, then the Envelope Receiver routes the necessary information to the Response Creator.
- The Response Creator generates the appropriate Response content based on information provided to it by the Envelope Receiver.
- The Response content is routed to the Envelope Packager for transmission back to the organization that sent the incoming Envelope.

4.2.6.3 Role playing: Response processing

This version of the diagram emphasizes the roles that an AdsMLEnvelope Processor plays when responding to an incoming AdsMLEnvelope that contains a Response.



- When the Envelope arrives, the Envelope Receiver identifies that it contains Response content rather than Items.
- The Response is routed to an Response Processor for processing.
- The Response processor takes whatever action is necessary based on the nature of the Response.

4.3 Source and nature of addressing information

4.3.1 Where does it come from?

An AdsMLEnvelope Processor can only deliver advertising information (packaged as Items within Envelopes) to the business entities identified by the Content Creating Applications. Stated another way, "I can't deliver it if you don't tell me where to sent it."

The Content Creating Applications are the ultimate source of information related to the document exchange. These applications know what companies they do business with and all other parties that might be related to any transaction. The Content Creating Applications are assumed to have correct information about business partners at the transaction level (that is, the identity of a business entity to which some advertising content should be sent) but not at the delivery level (e.g. the email address of that business entity).

The PPA that is in place between two parties is the only source of physical addressing information (i.e. transport mechanism and address) for the delivery of messages to those parties. The information contained within (or accessible to) the PPA is always the most current. Therefore, the Content Creating Application only provides the ID of the business entity to which some content should be delivered. This then is

translated during AdsMLEnvelope processing into the physical address to which that item is sent.

Every Content Creating Application must place delivery information in a recognizable place within the content that it provides, so that the Item Creator that will convert the content into an Item can find the information, or provide a custom mechanism by which it communicates the information to the Item Creator.

4.3.2 Who does it go to?

Every Item in an Envelope has a primary destination. This is the business partner that originated the exchange or is the target of an originating exchange. In EDI terms, this is the primary trading partner.

In addition to the primary trading partner, there can be any number of secondary partners that receive copies of the advertising information. In general, the secondary partners have no response requirement at the application level. (AdsMLEnvelope does not provide special handling for messages sent to secondary partners.)

For example, the following are potential trading partners as defined in the SPACE/XML specification, any of which could act as either a primary or secondary partner in a given message exchange:

```
AG Agent Agency
AO Account of Advertiser
AP Account of Origin Party
AQ Account of Destination Party
B3 Previous Name of Firm
BF Billed From
BN Beneficial Owner
BO Broker Sales Office
BT Bill to Party
BY Buying Party Purchaser Agency of Record
CF Subsidiary Division
CQ Corporate Office
CS Consolidator
DH Doing Business As
EM Party to Receive Electronic Memo of Invoice
PH Printer
PI Publisher
PM Party to Receive paper Memo of Invoice
PW Pick Up Address
SF Ship From
SH Shipper
SO Sold To If Different From Bill To
SR Samples to be Return To For proofs or tearsheets
SU Supplier, Manufacturer, Separator, Service Bureau
TS Party to Receive Certified Test Results
```

4.4 Item-level addressing metadata

4.4.1 Addressing metadata

One of AdsML's primary purposes is to facilitate the routing of advertising information – AdsML Items – between business entities. In order to accomplish this, the following Address-related metadata is associated with each Item in an AdsMLEnvelope:

- **ITEM ID** – the unique ID of this Item.
- **DESTINATION –** the business entity that is intended to consume this Item
- **TO –** the next business entity to which this Item will go. This is often the same as the Destination entity, but can be changed by the Item Redirector in order to redirect this Item to an intermediary entity.
- **LAST PROCESSED BY –** the last business entity that created, reviewed or modified the contents of this Item.
- **ACTIVITY HISTORY** – records the history of an Item as a sequence of Activities, each Activity consisting of three values: **ACTION, PERFORMED BY,** and **TIMESTAMP.**

4.4.2 Maintenance of the Item History

The Item's Activity History is a multi-valued stack that lists the Activities that have affected this Item and the business entities that performed them. It is incremented by each Item Creator that operates on the Item, or in the case of a Forwarding operation, by the AdsMLEnvelope Processor that forwards the item. The first entry in the stack always reflects the Item's creation.

Each Activity in the history stack contains three values:

- **ACTION**: Allowable values are "Create", "Review", "Modify" or "Forward"
  - **Create**: Item was created and a new globally unique ID was assigned to it. (This is the only Action that creates a new Item ID. It is always the first (oldest) entry in the Activity History stack.)
  - **Review**: Item was passed to a content creating/consuming application that reviewed or validated the contents of the item without changing it, after which the item contents were repackaged in the same Item in which they arrived.
  - **Modify**: Item was passed to a content creating/consuming application that modified the Item, after which it was repackaged in the same Item in which it arrived.
  - **Forward**: Item moved through a business entity without being unpacked and passed to a content creating or consuming application.
- **PERFORMED BY**: The business entity that performed the action
- **TIMESTAMP**: The date/time on which the Item was received by an Item Creator and repackaged, or, if in the case of a "Forward" action, the date/time on which the Item was forwarded.

Each business entity that creates, reviews, modifies or forwards an Item must increment that Item's Activity History.

Comments are invited on the following issues:

A) The history stack assumes success, but what should happen in case of errors? For example: suppose the business entity tries to forward an Item but fails. Should this action be logged in the Item History? Should there be a status associated with each action?

B) In order to record "Review" and "Modify" actions the Item Creator will need a closer connection to the Item Content Unpacker than is shown in the architecture, relatively complex internal functionality, and probably some form of persistent storage. This raises the question of whether the ability to Review or Modify the contents of an Item and then have them continue their journey in the same Item in which they arrived is sufficiently valuable so as to justify the more complex architecture that will be required. Should the Review and Modify actions be removed in order to make implementation simpler? Or as a compromise, should logging of "Review" and "Modify" be made optional and associated with one of the higher Conformance Levels?

4.4.2.1 When must the history stack be incremented?

A business entity that Forwards an Item to another destination without having reviewed or modified its contents is required to treat the forwarding operation as a continuation of the journey of the existing Item and increment its history stack appropriately.

A business entity that Reviews or Modifies the contents of an Item may choose whether to treat the next step in the Item's journey as a continuation of its prior journey (in which case the Item retains the same Item ID number and its history stack is incremented appropriately), or as a new journey by a new Item (in which case the Item Packager assigns a new Item ID and restarts its history stack by indicating a "create" operation).

This flexibility is provided in order to accommodate situations in which it would be difficult or inappropriate to recreate the original Item's ID and history information when transmitting it to its next destination. However, whenever a business entity reviews or modifies the contents of an Item without substantially transforming those contents into a new object, the preferred approach is for the entity to repackage and transmit the contents in the same Item in which they originally arrived, rather than creating a new Item.

4.4.3 Other metadata

The following other types of Item-level metadata are also used to control processing of that Item. See the Specification Part 2 for a more detailed description of these elements.

- `Item Type`
- `Priority`
- `Format`
- `Encoding`
- `Encryption`

4.4.4 ItemHeader vs. ContentHeader metadata

In most cases, each Item will contain a single unit of content that was created by the Content Creating Application – for example, a booking, an insertion order, or some advertising content. This content is stored in the `ContentData` element inside the Item. However, AdsMLEnvelope permits a single Item to contain multiple `ContentData` elements, provided that each of these separate content elements within the Item consists of an alternative representation of the same business information using a different format.

For example, a single Item could contain two versions of the same insertion order, one using the IfraAdConnexion format and the other using the SPACE/XML format, or two versions of the same advertising content, again using two different formats. This duplication is permitted **ONLY** when the only difference between the business information contained in the two content elements is the format that was used for them. The recipient of an AdsML Item that contains multiple alternative content elements is expected to process one and only one of the alternative representations, and to discard the contents of all of the other `ContentData` elements without having opened or examined them.

The ability to store multiple representations of the same Item within the Item element is optional functionality that requires prior agreement between both the

Sender and Recipient of that Item. In order to enable it, the structure of every Item allows for the possibility that it will contain multiple `ContentData` elements, each of which has an associated `ContentHeader` element. The AdsML metadata contained within an Item is allocated either to the Item's Header (in which case that metadata applies to all of the contents of the Item) or to a `ContentHeader` within the Item (in which case that metadata applies only to the specific `ContentData` with which it is associated). For example, the Item `Type` element is associated with the Item as a whole, while a separate Item `Format` element is associated with each unit of content contained inside that Item.

For purposes of clarity, the rest of this document ignores the distinction between `ItemHeader` and `ContentHeader` metadata, and is written as if each Item contains a single unit of content and all of the metadata attached to that Item applies to all of the content within it.

4.5 Envelope-level addressing metadata

4.5.1 Transport mechanism and physical address

In order to avoid exposure of a business entity's AdsMLEnvelope processor's physical address, an AdsMLEnvelope does not contain a copy of the physical address to which it should be sent. It is up to the Envelope Packager to determine the transport mechanism (e.g. http, ftp, email, digital delivery service, etc.) that should be used when conveying an AdsMLEnvelope to a given recipient, and the specific address to which the Envelope should be sent. This information is contained in the PPA between the sender and the recipient. Acting on this information, the Envelope Packager creates an appropriate physical package, and submits both it and the intended address to an appropriate delivery system.

4.5.2 Contact information

The header of an AdsMLEnvelope contains a `Contact` element, which in turn can contain one or more `Address` elements. These elements play no role in the addressing or routing of the AdsMLEnvelope or the Items within it. Instead, they are intended to provide human-readable contact information to the destination organization in case an error occurs and the recipients wish to contact someone at the sending organization in order to resolve the error.

4.6 Redirection capabilities and implications

AdsMLEnvelope provides capabilities to enable the redirection of an Item from the Destination specified by its Content Creating Application, to an intermediary destination specified in the PPA between the sender and the originally intended recipient of the Item. These redirection capabilities are optional and are associated with Level 2 conformance. This section discusses some of the associated issues.

The metadata provided in the AdsMLEnvelope layer (in the form of Item-level and Header-level metadata within an AdsMLEnvelope) is meant to support a relatively simple yet powerful set of capabilities, as described below. However, it must be remembered that under normal circumstances this information is only available to the AdsMLEnvelope Processor (and to human operators of processors in the form of reports and logs), not to the content creating and consuming applications

themselves. Further, AdsMLEnvelope acts in service to the instructions (in the form of a specified Destination) provided by the content creating application.

4.6.1 Redirection basics

- Addressing information is maintained, and addressing operations such as redirection are performed, at the Item level. In terms of addressing, the AdsMLEnvelope serves as a relatively simple container for a set of Items, each of which carries its own addressing information
- Address redirection can only affect the next "hop" that an Item will take. In other words, the Item Redirector can only affect the `To` element, and can only specify a single `To` location.
- In terms of redirection, an Item Redirector has only two choices: send the Item to its specified `Destination`, or redirect it to an alternate `To` location.
- It is an error to redirect an Item back to its sender.
    - (If you do, the original sender may report a "duplicate Item ID" error and refuse to process the Item.)
- Only a Content Creating Application can specify the `Destination` value. The AdsMLEnvelope facilities act in service to this specified Destination. As a consequence, an Item Creator will always set the `To` element equal to the `Destination` element.
- A Content Creating Application can only specify a single Destination. No mechanism is provided for a Content Creating Application to say that an Item must go "here, then there, then to this third place."
- Therefore, complex address chains in which an Item must pass through more than one intermediary on its way to its Destination depend on each Item Redirector in the chain having access to the appropriate information to send the Item on its way. Local knowledge at each step determines the next step.

4.6.2 Redirection responsibilities and requirements

There are only two permitted reasons for an Item Redirector to intervene in the routing of an Item:

- Agreement between the current business entity and the specified Destination entity that Items of, for example, this Type and Format should be redirected to a different entity
    - Along with Type and Format, redirection agreements can be based on any values contained in the Item's AdsML-defined metadata, including Item-level user-defined Properties. However, the redirection rules must not be based on information that is only found inside the Item's `ContentData` element.
- Agreement between the current business entity and a third party acting on its behalf that all outgoing Items should route through that third party, which will provide a "store and forward" service on behalf of the sending entity.

A given Item Redirector can only implement <u>one</u> of these two rules on a given Item, because no mechanism is provided for the Item Redirector to indicate that an Item should go first to one location and then to another.

- Therefore, in a situation where an entity contracts with an agent acting on its behalf to perform services of a "store and forward" nature, it is the responsibility of the entity to ensure that its agent has a PPA in place with each of the Destinations to which items might be sent via that agent, and can

perform any additional redirections that those Destination entities might request.

The above discussion is about responsibilities and requirements that are implemented by the Item Redirector. In addition, any Content Consuming Application through which the Item passes can perform address interventions:

- Each time an Item is unpacked and its contents passed to a Content Consuming Application, that application determines the next Destination of that content, and whether its next journey will be considered a continuation of the previous one, or a new journey starting at that business entity.

- Therefore, once an Item has been redirected, the AdsMLEnvelope facilities cannot "guarantee" that it will reach the Destination that was specified by the entity that originally created it, because any entity to which it is redirected might intervene and change its stated Destination by converting it into a new Item. This apparent weakness is considered by the AdsML Technical Working Group to be, in fact, a strength, because it supports in a relatively simple and straightforward way the complex twists and turns that occur in the real world.

When an Item is redirected through a third party, depending on the nature of the business relationships between those three parties, the intermediary may not have a formal business relationship with the original sender; this occurs, for example, when the party to which the Item is redirected only has a business relationship with the originally specified Destination, not with the Sender. In this case, it is the originally Destined recipient's responsibility to ensure that:

- The Intermediary it specifies has the necessary PPA information in place to accept and act upon the Item
- The business entity ID values that are used by all the participants in this scenario to identify each other will be understood by all of the systems through which the information will pass.

4.7 Addressing scenarios

This section contains three usage scenarios that illustrate common uses of the AdsMLEnvelope addressing and redirection functionality. While the scenarios themselves are merely illustrative, the examples of how the element values should be recorded in each situation reflect the requirements of the AdsMLEnvelope standard. In other words, in a real world situation analogous to any of these scenarios, an system is required to fill in the relevant addressing metadata with values that follow the patterns shown here.

4.7.1 Scenario 1: Originator to Destination

The following scenario illustrates the simplest case, in which an entity creates some content that is sent directly to a second entity where the content is consumed. Therefore, in this scenario the Originating Entity is also the Sender, and the Receiving Entity is also the Destination of the information contained in the message.

Note that this example describes only those actions performed by each software component that are relevant to the addressing scenarios. Each component also has other responsibilities that are described elsewhere in this document.

4.7.1.1 Within the Originating Entity ("A")

a) A Content Creating Application creates the content that will go into an Item, and passes the content to the Item Creator that is associated with that Content Creating Application (or with that type of content).
b) The Item Creator associated with that Content Creating Application encodes and packages the content as an Item. The Item Creator fills in the Item's type, format, priority and encoding, and places the following values in the Item's address-related metadata elements:
    a. ITEM ID: "1"
    b. DESTINATION: "B"
    c. TO: "B"
    d. LAST PROCESSED BY: "A"
    e. ACTIVITY HISTORY: Action = "Create"; Performed by "A"; Date/time
c) A's Item Redirector examines this Item, compares its destination to the address-related business rules stored in the local PPA, and determines that the item does not need to be redirected to a different entity. Therefore, none of the Item's addressing information is changed.
d) A's Envelope Packager assembles all the Items that are addressed TO "B" and puts them in a single Envelope addressed to that business entity. Based on information contained in the PPA, it routes the envelope by an appropriate transport mechanism (e.g. http or ftp) to the address provided by "B".

4.7.1.2 Within the Destination Entity ("B")

a) B's Envelope Receiver receives the Envelope from "A", verifies that "A" is a trusted sender, that the envelope's structure conforms to the AdsMLEnvelope Standard, and that the Envelope and Item-level metadata conform to the PPA that is in place between "A" and "B". Assuming that a response is required, the Envelope Receiver generates the response content and makes it available to B's Envelope Packager. The Receiver then extracts the Items from the envelope and makes them available to the Item Content Unpacker.
b) B's Envelope Packager receives the response content, places it in an AdsMLEnvelope, and sends that Envelope to A.
c) B's Item Content Unpacker extracts the content from the Item, restores it to its original form, and sends it to the appropriate Content Consuming Application.
d) The Content Consuming Application receives the content and acts upon it.

4.7.2 Scenario 2a: Originator to Destination via an Intermediary that does not change the content

The following scenario illustrates a slightly more complex case, in which a business entity creates some content that is routed to its destination via an intermediary. The intermediary then performs a "store and forward" service that does not involve processing the Items that it handles, other than to aggregate them into a different envelope and forward them to their destination. Familiarity with Scenario 1 is assumed.

Note that this scenario could be extended through multiple intermediaries.

4.7.2.1 Within the Originating Entity ("A")

a) A Content Creating Application creates the content and passes it to the appropriate Item Creator.
b) The Item Creator packages the content as an Item. The Item Creator fills in the Item's type ("XX"), format ("YY"), priority and encoding, and places the following values in the Item's address-related metadata elements:
    a. ITEM ID: "1"
    b. DESTINATION: "B"
    c. TO: "B"
    d. LAST PROCESSED BY: "A"
    e. ACTIVITY HISTORY: Action = "Create"; Performed by "A"; Date/time
c) A's Item Redirector examines this Item and compares its destination to the address-related business rules stored in the local PPA. The Addressor finds a rule that Items intended for entity B of type "XX" and format "YY" should be redirected to entity C instead. Therefore, the addressor changes the TO element, leaving the rest of the address information unchanged.
    a. ITEM ID: "1"
    b. DESTINATION: "B"
    c. TO: "C"
    d. LAST PROCESSED BY: "A"
    e. ACTIVITY HISTORY: Action = "Create"; Performed by "A"; Date/time
d) A's Envelope Packager assembles all the Items that are addressed TO "C" and puts them in a single Envelope addressed to that entity. Based on information contained in the PPA, it routes the envelope by an appropriate transport mechanism (e.g. http or ftp) to the address provided by "C".

4.7.2.2 Within the Intermediate Entity ("C")

a) C's Envelope Receiver receives the Envelope from "A", verifies that "A" is a trusted sender, that the envelope's structure conforms to the AdsMLEnvelope Standard, and that the Envelope and Item-level metadata conform to the PPA that is in place between "A" and "C". The Receiver extracts the Item from the envelope and routes it to C's Item Redirector. Note that in this scenario, because C is performing a "store and forward" service, the items are not Unpacked and are routed directly to an Item Redirector with no intermediate processing. For simplicity, we assume that no response message is required from C to A.
b) C's Item Redirector examines the Item and compares its destination to the address-related business rules stored in the PPA between C and the Item's stated Destination, B. In this case no further redirection is required. However, C must still change the Item-level addressing metadata to reflect the fact that it is passing through C's control, and it must increment the history information:
    a. ITEM ID: "1"
    b. DESTINATION: "B"
    c. TO: "B"
    d. LAST PROCESSED BY: "A"
    e. ACTIVITY HISTORY:
        i. Action="Forward"; Performed by "C"; Date/Time
        ii. Action = "Create"; Performed by "A"; Date/time

c) C's Envelope Packager assembles all the Items that are addressed TO "B", puts them in a single Envelope addressed to "B" and routes it to the address provided by "B".

4.7.2.3 Within the Destination Entity ("B")

a) B's Envelope Receiver receives the Envelope from "C", verifies that "C" is a trusted sender, that the envelope's structure conforms to the AdsMLEnvelope Standard, and that the Envelope and Item-level metadata conform to the PPA between that is in place between "C" and "B". Assuming that a response is required, the Envelope Receiver generates the response content and makes it available to C's Envelope Packager. The Receiver then extracts the Items from the envelope and makes them available to the Item Content Unpacker.
b) B's Envelope Packager receives the response content, places it in an AdsMLEnvelope, and sends that Envelope to C.
c) B's Item Content Unpacker extracts the content from the Item, restores it to its original form, and sends it to the appropriate Content Consuming Application.
d) The Content Consuming Application receives the content and acts upon it.

4.7.3 Scenario 2b: Originator to Destination via an Intermediary that touches the content

The following scenario illustrates what happens if the intermediary, C, reviews or modifies the content. Note that two options are provided: at its discretion, C may elect either to create a new Item or to continue the journey (and increment the history) of the Item that contained the content it received.

4.7.3.1 Within the Originating Entity ("A")

This section is the same as in scenario 2a.

4.7.3.2 Within the Intermediate Entity ("C")

a) C's Envelope Receiver receives the Envelope from "A", verifies that "A" is a trusted sender, that the envelope's structure conforms to the AdsMLEnvelope Standard, and that the Envelope and Item-level metadata conform to the PPA that is in place between "A" and "C". The Receiver extracts the Item from the envelope and routes it to C's Item Content Unpacker. For simplicity, we assume that no response message is required from C to A.
b) C's Item Content Unpacker extracts the content from the Item, restores it to its original form, and sends it to the appropriate Content Consuming Application.
c) C's Content Consuming Application receives the content and acts upon it. Since C is an intermediary acting on behalf of B, let us assume that C's application validates that the content conforms to B's technical requirements, and if necessary changes the content so that it conforms to B's requirements. Having done so, C's Content Consuming Application passes the validated content to the appropriate Item Creator.
d) C's Item Creator re-encodes the content and packages it as an Item. C can now choose whether to treat this as a new Item or as a continuation of the original Item. The options are:
    a. C TREATS THIS AS A NEW ITEM AND STARTS A NEW HISTORY STACK:

i. ITEM ID: "2"
ii. DESTINATION: "B"
iii. TO: "B"
iv. LAST PROCESSED BY: "C"
v. ACTIVITY HISTORY: Action = "Create"; Performed by "C"; Date/time

b. C TREATS THIS AS AN EXISTING ITEM AND INCREMENTS THE HISTORY STACK:
iii. ITEM ID: "1"
iv. DESTINATION: "B"
v. TO: "B"
vi. LAST PROCESSED BY: "C"
vii. ACTIVITY HISTORY:
1. Action="Review"; Performed by "C"; Date/Time
2. Action = "Create"; Performed by "A"; Date/time

e) Assuming that no further redirection is required, C's AdsML Envelope Packager assembles all the Items that are addressed TO "B", puts them in a single Envelope addressed to "B" and routes it to the address provided by "B".

4.7.3.3 Within the Destination Entity ("B")

This section is the same as in scenario 2a.

5 Process Partnership Agreement

The Process Partnership Agreement, or PPA, contains all of the information that must be available to an AdsMLEnvelope Processor in order to communicate properly with each of its communication "partners", where each "partner" is a business entity with which it can exchange AdsMLEnvelope messages. This section describes the relationship between the PPA, TPA and an AdsMLEnvelope Processor, see also the *AdsML E-Commerce Usage Rules and Guidelines* for a general discussion.

5.1 Access to rules and "AdsMLEnvelope metadata"

The "choices" made by an Item Redirector, Envelope Packager or Envelope Receiver and Item Content Unpacker when creating or receiving and validating an AdsMLEnvelope or when redirecting an Item **MUST** be based on the combination of the rule set contained in the relevant PPA, plus the AdsMLEnvelope metadata that is available to the AdsMLEnvelope Processor (where by "metadata" we mean the AdsML-defined elements and attributes of the Item(s) or Envelope in question, including any user-extended Properties within the Item(s) or Envelope.) This metadata is the only information in the message or its contents that can be assumed to be visible to the AdsMLEnvelope Processor. Therefore, the processing choices made during these operations **MUST NOT** be based on information that is only found inside the `ContentData` element of an AdsML Item.

Note that the Item Packager is assumed to be closely bound to the content format used in that Item, and is exempt from this rule.

5.2 Relationship to a Trading Partner Agreement

Two business entities will only have one Trading Partner Agreeement (TPA) between them. The TPA represents, in its entirety, the business relationship between the two entities (where an entity may be an entire organization or a part of an organization).

Part of this information is recorded in the PPA, which is that part of the TPA that describes how to use AdsML facilities to do ebusiness between the two partners. The PPA will be used by the AdsMLEnvelope Processor parts (Item Redirector, Envelope Packager and Envelope Receiver) in combination with the AdsMLEnvelope metadata (both Envelope and Item level) to do appropriate processing.

5.3 Intermediary and on-behalf-of business partners

As the word "Partnership" implies, it is assumed that in most cases any two business entities that are willing to exchange AdsMLEnvelope messages with each other will have some form of business relationship, and will "partner" in the exchange of messages. However, this does not mean that they must have a traditional business relationship in which one of them pays the other for its services.

A further discussion of this issue can be found in Appendix B.

5.4 PPA Format and Contents

AdsML does not specify the physical format of a PPA or the full set of its possible contents. Two partners who are establishing AdsML communications with each other are free to implement rule-based processing based on any of the AdsML-defined metadata, provided that each of their AdsML software systems are capable of accessing and acting upon the processing rules that they have agreed between them.

Note that a user-extended AdsML schema is considered a part of the PPA, because it defines, in machine-readable form, the message content that an AdsML implementation expects to receive.

The list below is illustrative. The optional functionality and types of validation that two communications partners have agreed to implement will define the need for any given type of information in their respective PPAs. Depending on the ambitions of an implementer and the conformance Level that has been implemented in the systems in question, the list could be longer or shorter than that shown here.

For a given communications partner with which an AdsMLEnvelope Processor communicates, a minimal PPA can include:

- Self identification
  - The business entity ID (combination of ID string and Type attribute) that will be used to identify "myself", for example in the Sender element of the Envelope header.
- Partner identification
  - The business entity ID string and ID Type that will be used to identify this particular partner, for example in the `To` and `Destination` elements of an Item header.
- Addressing information:
  - Transport mechanism (e.g. HTTP, FTP, email, or a delivery service)
  - Address to which to send the message
  - (Note that this information is not recorded inside the Envelope itself)
- Transmission/Response functionality ("send and forget" or "store and resend until acknowledged" model)
  - "Send and forget" is the default
  - "Store and resend" must be agreed by both parties in order for that functionality to take effect
  - Note that this is a bilateral option that must be the same in both directions
- The version(s) of the AdsMLEnvelope standard that may be used when communicating with each other.

Depending on the operations that an AdsMLEnvelope system is required to perform, a PPA might include additional information such as:

- Lists of allowed Controlled Vocabulary values
  - For each type of Controlled Vocabulary in an AdsMLEnvelope whose contents can be restricted or extended by an AdsMLEnvelope User, the PPA (including, if present, the user-extended schema) must specify the permitted values that can included in an Item or Envelope sent-to or received-from this trading partner..Some of the controlled vocabularies that might normally appear in a PPA include:

        - Item types
        - Formats/Standards
        - Encoding types
        - Encryption types
- Authentication credentials
    - Whether a digital signature is required for the AdsMLEnvelope
    - The circumstances (such as specific Item Types) under which a digital signature is required for each Item within an Envelope
- Redirection
    - The values or combinations of values of the AdsMLEnvelope metadata in an Item that is Destined for this partner that should trigger a redirection of that Item to a third party.
- Whether multiple XML representations of the same content are accepted
    - Note that multiple representations must be agreed by both parties in order to take effect
- Priority handling
    - Whether higher priority items should receive special handling
    - If so, any priority value thresholds that should trigger specific types of handling (e.g. "priorities of 7 and higher should bypass the holding queue and be handled immediately")

6 Response choreography

This section discusses the AdsMLEnvelope message choreography (the send and response patterns) that must occur once an AdsMLEnvelope Processor sends a message to another AdsMLEnvelope Processor. Some of this functionality is defined in one of two "Transmission/Response modes", shown immediately below. The remainder of the response choreography is independent of the selected Transmission/Response mode, but is triggered instead by the Status of the Envelope and its *responseRequired* attribute.

6.1 Transmission/response modes

AdsMLEnvelope supports two modes of transmission/response functionality: *"Send and forget"* and *"Store and resend until acknowledged"*. Only these two modes are available. Any given pair of communications partners **MUST** agree to use one of these modes when communicating with each other and **MUST NOT** reduce or ignore any of the required functionality of that mode.

"Send and forget" is the mandatory functionality that all AdsMLEnvelope processors must support. It defines the default choreography that two business entities must follow when transmitting and responding to AdsMLEnvelope messages, in the absence of a specific agreement to the contrary.

"Store and resend until acknowledged" is optional functionality that is associated with AdsMLEnvelope Level 1 conformance. It defines a more robust choreography that two business entities may follow when communicating with each other, provided that both of their processors support AdsMLEnvelope Level 1, and that they have both agreed to use the "Store and resend until acknowledged" mode when exchanging messages with each other.

6.1.1 "Send and forget"

"Send and forget" is the minimum default functionality that all AdsMLEnvelope processors must support. The following list describes the "Send and forget" functionality.

- A system is not required to retain a copy of the AdsMLEnvelope messages that it has sent or any audit trail information about them
- A system never automatically resends an AdsMLEnvelope and never increments the *sendCount* attribute.
- A system **MUST** generate a Response to an Envelope that it has received when any one (or more) of the following conditions is true:
    a. The Envelope's Status is "Production" and validation or verification of the Envelope's structure and contents identifies one or more Catastrophic or Item-level errors.
        - Note that this condition applies regardless of whether the Envelope contains Item or Response content.
    b. The Envelope's Status is "EnvelopeTest"
    c. The Envelope contains Item content and it's *responseRequired* attribute is 'true'

- If none of the above conditions is true, a system **SHOULD NOT** generate a response to an Envelope containing Item content, and **MUST NOT** generate a response to an Envelope containing Response content.
- Any such Envelope Response message (whether "message OK", an Envelope-level Catastrophic error, or an Item-level verification error) **MUST** contain within itself a copy of the entire Envelope to which it is a response, including all of the Item content that was contained within that Envelope
- A system is not required to identify duplicate Envelopes or Items that it has received, though it may do so if it wishes.
- The recipient of an AdsMLEnvelope is permitted to ignore its send count and to pass all valid and verified Items in the Envelope to the appropriate Content Consuming Application even if the Envelope's send count is greater than zero, or if an apparently duplicate Envelope has previously been received. It is therefore up to the Content Consuming Application to detect any duplicate Item content that may have been sent to it.

Note that this model discourages the use of responses except when reporting an error or during system setup. This is because in this model, even routine "message OK" responses contain copies of the original message content, and therefore double the amount of bandwidth consumed by AdsMLEnvelope communications. However, the model does not actually prevent the sending of "message OK" responses if a user wishes to do so.

6.1.2 "Store and resend until acknowledged"

"Store and resend until acknowledged" describes optional functionality that requires an agreement between both sender and recipient in order to take effect. The "Store and resend until acknowledged" model is preferred over the "Send and forget" model and should be followed if possible. However, an AdsMLEnvelope implementation **MUST** only follow the "Store and resend until acknowledged" model when communicating with a trading partner that has explicitly agreed to follow this model. Therefore, a given AdsMLEnvelope implementation that is capable of supporting "Store and resend until acknowledged" will only be able to use this functionality when communicating with a subset of its trading partners, and will have to follow the "Send and forget" model with the rest of its trading partners.

The concept of "resending" applies to the entire AdsMLEnvelope, so that in every possible way a re-sent Envelope is a duplicate of its original, except for the incremented Send Count and the date/time of the retransmission. It is assumed (and required) that the Items within the Envelope, including their metadata, do not change when re-sent. For example, the system does not increment the Action History of the Items in a re-sent Envelope.

The following list describes the "Store and resend until acknowledged" functionality.

- When a system sends an AdsMLEnvelope containing Item content whose Envelope status is "production", it **MUST** retain a copy of that Envelope at least until it has received a Response indicating that the Envelope was successfully received, validated and verified by its intended recipient. If no response is received within an appropriate amount of time, the system **MUST** re-send another copy of the Envelope.

    - This requirement does not apply to Envelopes that contain Response content. An Envelope that contains a Response to another Envelope is assumed to have been received correctly and **MUST NOT** be re-sent.
- This resending cycle continues until either a Response to the Envelope has been received, or until an unacceptable amount of time (or number of resends) has passed without receiving a response. At that point the sender's system signals an error in order to trigger manual follow-up and resolution of the communications problem.
    - The amount of time or number of resends that must pass before an error is raised is at the sender's discretion, but the sender's system **MUST** implement such a limit and, when the limit has been reached, **MUST** raise an error to its operators that provides sufficient information so that they can attempt to rectify the error and then re-send the contents of the Envelope.
- Each time a system re-sends an Envelope, it increments the Envelope's `sendCount` attribute.
- A system that receives an AdsMLEnvelope whose envelope status is "Production" **MUST** generate a response to that Envelope, except that if the incoming Envelope does not contain any Item content (i.e. it contains only a Response) then the system **MUST** only generate a response to the Envelope if it contains an error that requires a response.
- In this mode the `responseRequired` attribute is ignored.
- The Response to an Envelope or Item **SHOULD** contain a reference to the ID of the Envelope or Item to which it is a response rather than a copy of that entire Envelope or Item.
- A system **MUST** retain sufficient information about the Envelopes that it has received for a reasonable period of time so that it can verify, when receiving an AdsMLEnvelope, that it has not previously received another Envelope with the same globally unique Envelope ID, and if it has, whether that Envelope came from the same sender. The following rules **SHOULD** then be applied, depending on whether the second Envelope is from the same Sender as the first one, and whether it has the same or a different Send Count:

	Same send count	**Different send count**
Same sender	Error: Report error to sender, discard Envelope	Not an error: Discard envelope, do not report an error
Different sender	Error: Report error to Sender, route Envelope to manual processing	Error: Report error to Sender, route Envelope to manual processing

Unlike the "Send and forget" model, "Store and resend until acknowledged" requires that a response be provided for every incoming Envelope that contains Item content. This is because the model specifies that in the absence of such a response the sender will resend their Envelope. In order to avoid unnecessary message re-transmissions, the sending of "message OK" responses is a core feature of this model.

6.2 Production and Test modes

The AdsMLEnvelope Transmission/Response modes described above define the circumstances under which an AdsMLEnvelope Processor may or may not respond to an Envelope whose status is "Production". This section describes the relationship between the three possible Envelope Status values ("`Production`", "`EnvelopeTest`" and

"`ResponseToEnvelopeTest`"), the permissible contents of the AdsMLEnvelope, and the response choreography that must be followed.

If the Envelope's status is "`Production`" then:

- The Envelope **MUST** contain at least 1 Item or exactly 1 Response; it **MUST NOT** be empty, it **MUST NOT** contain a mixture of Items and Responses, and it **MUST NOT** contain more than one Response
- The rules in the applicable Transmission/Response mode determine whether a Response must be generated
- The receiving AdsMLEnvelope Processor is expected to act upon all valid and verified Items contained within the Envelope.
- If the Envelope contains only a Response, then its responseRequired attribute **SHOULD** be set to '`false`'.

If the Envelope's status is "`EnvelopeTest`" then:

- The Envelope **MUST** contain at least one dummy Item or exactly one dummy Response; it cannot be empty.
- The receiving AdsMLEnvelope Processor **MUST** discard any Items contained within the Envelope without acting upon them or routing them to Content Consuming Applications.
- The receiving AdsMLEnvelope Processor **MUST** generate a response message whose Envelope status is "`ResponseToEnvelopeTest`"
- The response **MUST** contain the same results that a validation/verification of an incoming "Production" Envelope would have generated: either an indication that the message was "OK", or any appropriate validation or verification errors

If the Envelope's status is "`ResponseToEnvelopeTest`" then:

- The Envelope **MUST** contain exactly one Response and no Item content
- The receiving AdsMLEnvelope Processor **MUST NOT** generate a response to it.

6.2.1 Test Items

The AdsMLEnvelope Processor provides facilities, as described above, to support the setup and testing of Envelope transmissions between two systems. AdsML does not provide facilities for using the AdsMLEnvelope Processor to support the testing of the contents of Items that are transported within AdsMLEnvelope messages.

It is assumed that when users are implementing communications between a Content Creating Application and a Content Consuming Application, they will perform any required testing based on information that is contained in the content created and consumed by those applications. This content-level information is "invisible" to the AdsMLEnvelope Processor. As a result, AdsMLEnvelope Processors will not be able to distinguish between "test" Items and "production" Items, and will treat them all in the same fashion.

Individual Item-level standards, however, do define metadata within their message formats to support the identification and special handling of test Items. This metadata is meant to be acted upon by the Content Creating and Content Consuming Applications which generate and consume those messages, not by the AdsMLEnvelope Processor which operates at the Envelope level.

6.3 Responses

6.3.1 Response addressee

The Response to an AdsMLEnvelope, including notification of verification errors relating to any Items within that Envelope, is sent to the business entity that was specified in the incoming Envelope's header as its Sender.

6.3.2 Response priority

The priority of a response **MUST** be no lower than the priority of the Envelope to which it is a response.

6.3.3 Upstream error notification

If an Item inside an Envelope generates a verification error, notification of that error **SHOULD** be sent to (and only to) the Sender of the Envelope, which may not be the same as the entity that last processed the Item in question (and presumably caused the error). It is not the responsibility of the entity that performed the validation to notify any additional "upstream" business entities that may have handled the Item.

For purposes of error notification, a business entity that sends an Envelope is assumed either to have "touched" all of the Items in the Envelope (in which case it assumes responsibility for repackaging them correctly before re-transmitting them to their next destination), or if performing a "store and forward" operation, to have a sufficient business relationship with the entity that last processed each Item so that it can inform that entity of any errors concerning that Item.

6.3.4 Response to Responses

An AdsMLEnvelope Processor **MUST NOT** respond to a Response message unless it causes verification or validation errors.

6.3.5 The responseRequired Attribute

The *responseRequired* attribute only affects processing when the status of the Envelope is "Production", it contains Item content rather than a Response, and the Transmission/Response mode is "Send and forget". In this situation, setting *responseRequired* to "true" **MUST** override the recipient's default behavior and cause the recipient to send a response even if the Envelope did not generate any validation or verification errors. In all other situations, this attribute is ignored.

7 Error handling

An Envelope Receiver is required to validate incoming messages and identify certain kinds of structural and content errors. Depending on the nature of the errors it encounters, the Envelope Receiver must be able to respond in two ways:

1. If the error is of a type for which AdsMLEnvelope defines response handling (see below), and if the sender of the message can be identified, the AdsMLEnvelope processor **MUST** send an envelope to the Sender containing a Response that informs the Sender of the error(s)
2. In all cases the AdsMLEnvelope processor **SHOULD** report the errors, including those for which AdsMLEnvelope has not defined specific response handling, internally to a person or system that is the designated recipient of non-standard errors, so that the person or system can attempt to resolve the error.

Four types of error situations, and their associated handling, are summarized in the following table and defined more fully below. Further information about the error reporting mechanisms is provided in the AdsMLEnvelope Specification Part 2 - Schema.

Error Category	Is validation mandatory?	Triggering Conditions	Response if error
Non-acceptance	Yes - **MUST**	• Unknown sender • Sender's credentials fail authentication • Not a trusted sender • Not well-formed XML	• Discard envelope without processing • Optionally notify sender
Catastrophic error	Yes – **MUST**	• Not valid according to the definitions embodied in the un-extended AdsML Schema • Not valid according to PPA-defined Envelope-level requirements	• Discard envelope without processing • Notify sender
Catastrophic error	No – depends on PPA and conformance level	• Not valid according to user-extended AdsML Schema at the Header level	• Discard Envelope without processing • Notify sender
Item error	No - **SHOULD**	• Not valid according to PPA-defined Item-level requirements • Not valid according to user-extended AdsML Schema at the Item level	• Discard the Item or route it for manual error handling • Notify sender • Attempt to process all other Items

7.1 Non-acceptance

7.1.1 Error definition

An AdsMLEnvelope is considered to be un-acceptable (in the literal sense of the word) when either of the following is true:

- The recipient system cannot open the Envelope because some or all of it is not well-formed XML
- An evaluation of the Sender's business entity ID(s) and ID Type(s) (and digital signature, if required) in the Envelope's header shows that the Envelope comes from an unknown business entity or from an entity that is known to the recipient but not a "trusted sender" of AdsMLEnvelope messages.
- An evaluation of the Sender's ID provided by the underlying communications system shows that the Envelope comes from an unknown or not trusted business entity.

7.1.2 Error handling

If the Sender is unknown to the recipient, the AdsMLEnvelope Processor **MUST** discard the Envelope without processing its contents or replying to its Sender.

If the Sender is known to the recipient but not a trusted sender, the recipient **MUST** either discard the Envelope as above, or treat it as a Catastrophic Error (as defined below). In both cases, its contents are discarded without further processing.

7.2 Catastrophic errors

7.2.1 Error definitions

A Catastrophic Error is a condition in which the entire Envelope is considered to be in error, and therefore, none of its Item or Response content can safely be processed. A Catastrophic Error occurs when any of the following are true:

- The Envelope does not have a globally unique ID
    - Note that only AdsMLEnvelope Processors that retain appropriate information about previously-received Envelope IDs can perform this validation.
- The structure or contents of the Envelope do not conform to the version(s) of the un-extended AdsMLEnvelope Schema that the recipient is able to process.
    - For example:
        - The recipient supports AdsMLEnvelope 1.0 but the message is in a 2.0 format.
        - The message is missing mandatory information
    - Note: by "un-extended" we mean the AdsMLEnvelope Schema without any user-defined extensions or restrictions.
- The Envelope contains non-agreed controlled vocabulary values in its Header.
    - For example, the sender and recipient have agreed to use certain user-defined Properties in their Envelopes, but the Envelope either does not contain a required Property value or contains a value that has not been agreed.

    - Note that this error only applies to header-level metadata, and does not apply to controlled vocabulary values in any of the Items within the envelope.
- The Envelope contains Response content that generates a verification or validation error of any kind
    - For example, the Response is structurally incorrect, is missing mandatory content, or contains non-agreed controlled vocabulary values as defined in the governing schema (whether or not that schema has been extended by the trading partners).

7.2.2 Error handling

When a Catastrophic Error occurs, the recipient Processor **MUST** report a "catastrophic" error and discard all of the contents of the Envelope without further processing. Depending on the Transmission/Response mode that is in effect, the error report either references or includes the contents of the entire Envelope that triggered the error.

Note that the recipient is not required to validate the Envelope using the supplied AdsMLEnvelope Schema and extension mechanisms. However, the recipient is required to validate the incoming Envelope by a mechanism that achieves the same results.

7.3 Item errors

7.3.1 Error definitions

An Item within an Envelope (but not necessarily anything else in the Envelope) is considered to be in error when the Envelope's metadata about that Item indicates that any of the following are true:

- The Item has the same globally unique ID as another Item that has previously been received in a different AdsMLEnvelope message.
    - Note that only AdsMLEnvelope Processors that retain persistent information can perform this validation.
- The Item contains non-agreed AdsMLEnvelope controlled vocabulary values (for example, it is of a Type, Format, Encoding or Encryption that the recipient has not agreed to accept from this Sender)
    - Note that this applies both to AdsML-defined controlled vocabulary values, and to optional user-defined controlled vocabulary extensions
- The Item's metadata lacks one or more values that have been specified as required values in the PPA between the Envelope's Sender and Recipient.
    - For example, if a digital signature is required for each Item, its absence is considered an Item-level error.

7.3.2 Error handling

When an Item is in error the AdsMLEnvelope Processor Reports the error to the Sender and tries to resume processing of any other Items in the Envelope. All non-error Items should be processed if possible.

An error report in response to one or more Item-level errors in an Envelope lists the Items that were successfully routed to their respective Content Consuming

Applications. The absence of an Item from this list tells the Sender that its recipient did not process the Item in question.

As with catastrophic errors, the Envelope Response to an Item-level error either references or includes the contents of the entire Envelope that triggered the error. There is no mechanism in this situation to reference or include the contents of just the offending Item.

8 Priority handling

8.1 Overview

Each AdsMLEnvelope can be assigned an overall "priority" value. In addition, each Item or Response element inside an Envelope can be assigned its own priority.

The intention of the priority mechanism is to allow the sender of an Envelope, Response or Item to signal its relative priority compared to other Envelopes, Items or Responses that may also require processing.

8.2 Operational requirements

8.2.1 Processing

Priority handling is associated with AdsMLEnvelope Level 2 conformance. An AdsMLEnvelope Level 2 system **MUST** be capable of applying special handling to higher-priority Envelopes, Items and Responses in the following situations:

- When prioritizing the selection, packaging and transmittal of outgoing Items, Responses and Envelopes
- When prioritizing the validation, unpacking and internal distribution of incoming Items, Responses and Envelopes

However, AdsMLEnvelope does not define the processing that should be associated with specific priority values. It is up to individual business entities and their communications partners to determine what, if any, special handling should be applied to higher priority items.

8.2.2 Values

Besides the restraints imposed by their formal schema definition, the following constraints apply to priority values:

- The priority assigned to an Envelope **MUST** be no lower than the highest priority assigned to any of the Items or Responses in an Envelope
- The priority of a Response to an Envelope **MUST** be the same as the priority of the Envelope to which it is a Response
- The priority of a Response to an Item **MUST** be the same as the priority of the Item to which it is a Response

9 Responsibilities of AdsMLEnvelope Processors

This section lists the primary responsibilities of an AdsMLEnvelope Processor in each of the roles that it can play. This is meant to summarize and supplement the more detailed descriptions of these responsibilities that appear throughout this document.

9.1 Responsibilities of an Item Creator

9.1.1 Encoding

The overall objective for the Item Creator is to prepare the content to be transferred so that it can be correctly packaged in the AdsMLEnvelope, routed by the AdsMLEnvelope Processor and unpacked by the destination system.

Being an XML document, the AdsMLEnvelope requires all content to be transported within it to be well-formed according to the W3C's Recommendation *Extensible Markup Language (XML) 1.0 (Second Edition)* (http://w3.org). This includes ensuring that no characters within the content violate the character set specified in the XML standard, as well as removing or commenting out DOCTYPE and other declarations that may only occur at the start of stand-alone XML documents.

The information model expressed in the AdsMLEnvelope Schema enables the following content types:

1. XML document - any well-formed XML document can be transported without any special precautions.
2. XML string data - any string including only characters that are valid according to the XML standard can be transported, provided that the string is marked as a CDATA section and does not include characters such as <&> that may destroy the well-formedness of the XML structure.
3. Non-XML string data and binary data - all strings including characters that cannot be properly represented in XML documents, e.g. binary data, have to be encoded using, for instance, hexbinary or base64binary encodings.

The AdsMLEnvelope standard does not require implementers to choose a particular programming technique when implementing the packaging as long as the above requirements on the resulting content are met.

In case of binary or other non-XML data where some encoding method has been used, the method must be recorded in the Item level metadata to enable the recipient to decode the data before routing it to the Item handler.

9.1.2 Responsibilities list

- Create a syntactically valid Item
- Place the content in an Item Content element, encoding it as necessary so that it can safely be carried inside an AdsMLEnvelope (i.e. remove any character sequences that would cause an XML parsing error)
- Use an encoding mechanism that will be acceptable to the recipient

- Fill in all Item-level metadata, including addressing information, based on either information provided by the Content Creating Application or metadata inside the generated content
- Ensure that the values in the addressing metadata follow the patterns described in the "Addressing Scenarios" section of this document.
- Encrypt the information, if required, using an encryption mechanism that will be acceptable to the recipient
- Update the Item's history stack
    - Note: in case of a "store and forward" operation of an entire Envelope, where none of its Item contents changes, it may be more appropriate for the Envelope Packager to increment the history stacks of the Items in the Envelope
- Optionally, merge alternative representations of the same content that are intended for the same recipient into a single Item
    - Note: depending on the source and nature of the contents, this function may need to be performed by a different component.

9.2 Responsibilities of an Item Redirector

- Optionally redirect the Item to a different recipient, based on information contained in the PPA between the current business entity and the stated Destination of the item.

9.3 Responsibilities of an Envelope Packager

- Create a syntactically correct AdsMLEnvelope containing either a single Response or one or more Items
    - Identify Items that are intended for the same recipient and place them in an AdsMLEnvelope
    - Fill in the header-level metadata of that Envelope
- Transmit the Envelope to the intended recipient's AdsMLEnvelope Processor via the transport mechanism and to the address specified in the relevant PPA
- Optionally, respect the priority levels of the available Items and Responses when determining which ones to package and deliver, and in what sequence
- Optionally, store and resend Envelopes that contain Items until an acknowledgment has been received
    - Do not store and resend Envelopes that contain a Response.

9.4 Responsibilities of an Envelope Receiver

- Ensure that the Envelope comes from a "trusted sender"
- Validate that the Envelope's structure and non-user-defined metadata conform to the AdsMLEnvelope schema whose version is stated in the message header, and that this version is acceptable to the recipient
- Validate that any user-defined metadata in the Envelope conforms to PPA agreed between the sender and recipient.

- If required by the combination of the results of the validation process, the Transmission/Response Mode that pertains to this Envelope, and the relevant metadata in the Envelope's header, provide sufficient information to the Envelope Response Creator that it can create an appropriate Response to this Envelope.
- If the Envelope's status is "test" or a Catastrophic Error has occurred, ensure that none of the Items in the Envelope are routed for further processing.
- If any of the Items in the Envelope trigger Item-level errors, ensure that those Items are either discarded or routed to error handling.
- Depending on the role played by the current business entity and the nature of the Items contained within the envelope, handle the Envelope according to internal business rules, performing some combination of the following actions:
    - Extract those individual Items that do not have Item-level errors and have not reached their final destination ("store and forward"), increment their history information as required and send them to the AdsML Item Redirector.
    - Extract those Items that do not have Item-level errors and send them to the Item Unpacker
- Optionally, respect the priority levels of the Envelopes and the Items within them when determining which ones to process and route internally, and in what sequence

9.5 Responsibilities of an Item Content Unpacker

- Remove the contents of the Item and restore it to a condition that is functionally equivalent to the contents that were originally placed in the Item.
- If an Item contains alternative representations of the same content, select the alternative version that will be processed and route the Item to the appropriate Content Consuming Application for that type of Item
- Route the restored content to the appropriate Content Consuming Application.
    - The decision of where to route a given unpacked Item is expected to be based on a combination of the information found in the Item header, including:
        - Format (e.g. the name of standard) with its associated Version
        - Message type (e.g. Booking, materials delivery, etc.)
        - Message class (Standard business information, administrative acknowledgement or error)
        - Any user-defined properties
    - NOTE: In some cases, it will be necessary to store the unpacked content as a file in the recipient's file system. In that case, you can use a user-defined property in the Item header to send the target filename to the content unpacker.

9.6 Responsibilities of a Response Creator

- Create a syntactically valid Envelope Response that contains all required metadata and indicates the nature of the Response
- If a Catastrophic error was encountered, indicate the nature of the error
- If one or more Item-level errors were encountered:
    - Indicate at least the nature of the first error, and optionally indicate the nature of any additional errors
    - Identify all of the Items (if any) that were processed without error
- Include either a copy of the original Envelope or its ID
- Provide the Response to the Envelope Packager for transmission to the sender of the original Envelope

9.7 Responsibilities of a Response Processor

- If an error has occurred, take appropriate action such as logging the error and/or notifying an operator
- If no error has occurred and the Transmission/Response Mode is "Store and resend until acknowledged", indicate to the system that the Envelope to which this is a Response has been received, so no further resending is necessary.

10 Conformance

10.1 Approach

The Conformance requirements in this section are designed to facilitate the process by which two or more trading partners implement systems that exchange AdsMLEnvelope messages, by providing sufficient information so that the trading partners can reasonably determine that the behavior of the systems they have built, and the structure and contents of the Envelopes being exchanged between those systems, conform to the AdsMLEnvelope specification.

Conformance is defined in terms of mandatory "Core" functionality and information content, supplemented by two additional "Levels" of optional functionality and information content. An AdsMLEnvelope processor can therefore be labeled as supporting either "core", "level 1" or "level 2" functionality.

Using this approach, in order for two trading partners to agree on the functionality that they will support when communicating with each other, their first step should be to determine the Levels of support that are provided by their systems. Only optional functionality that is supported by both of their systems can be agreed upon.

The Technical Working Group feels that this approach provides an appropriate degree of information, which will enable users to implement systems that successfully exchange AdsMLEnvelope messages. It is not a current goal of the Technical Working Group to go further and define Conformance to a level of detail that would enable objective third party testing and certification of AdsMLEnvelope Conformance.

10.2 Conformance levels

This section summarizes the functionality that is associated with each Level of AdsML conformance.

10.2.1 Core

An AdsMLEnvelope Processor must provide all of the functionality described as mandatory in this document that is relevant to the role(s) played by that processor (e.g. Item Redirector, Envelope Packager, etc.), with the exception of the following:

- No validation of user-defined vocabulary values
    - (Note that the lack of this validation does not prevent the inclusion of such values in an AdsMLEnvelope message: it merely means that the recipient will not be able to validate that the user-defined values in a given message conform to the relevant PPA)
- No "Store and resend until acknowledged" capabilities
- No use of alternative Item content within a given Item
- No address redirection
- No special handling based on the Priority attribute

10.2.2 Level 1

An AdsMLEnvelope "Level 1" processor must provide all of the "core" functionality plus:

- Validation of user-defined vocabulary values
- "Store and resend until acknowledged" capabilities

10.2.3 Level 2

An AdsMLEnvelope "Level 2" processor must provide all of the "Level 1" functionality plus:

- Support for use of alternative Item content within a given Item
- Address redirection
- Special handling based on the Priority attribute
    - This means that higher priority items and envelopes **MUST** be handled "first". However, the nature of the handling is left up to the implementer.

10.3 Conformance requirements

This section consists primarily of conformance requirements that were defined in AdsML Requirements version 1.0. The requirements in this section apply in addition to the conformance requirements that are described throughout the remainder of this document.

10.3.1 Allowable Item content

1. The contents of an Item **SHALL** be a single XML document instance or non-XML character string that conforms to one of the following three options:

    a. An XML document instance: a valid or well-formed XML document that conforms to the W3C's Recommendation *Extensible Markup Language (XML) 1.0 (Second Edition)* (http://www.w3.org/TR/REC-xml/), in which any declarations (such as DOCTYPE) that are not permitted in the body of an XML document have been removed or commented out.
    b. XML string data: any string including only characters that are permitted according to the XML standard can be transported and does not include characters such as <&> that may destroy the well-formedness of the AdsMLEnvelope's XML structure.
    c. Non-XML string data and binary data: all strings including characters that cannot be properly represented in XML documents, e.g. binary data, have to be encoded using, for instance, hexbinary or base64binary encodings.

This requirement is meant to ensure that no characters within the Item content violate the character set specified in the XML standard, as well as removing or commenting out DOCTYPE and other declarations that may only occur at the start of stand-alone XML documents.

2. An Item's Content data content may contain a reference to the location of one or more external information objects, for example, artwork files that are retrievable from a specified URL or FTP address.

10.3.2 Alternative representations of the same advertising information

1. A user (such as an aggregator) may choose to represent a single instance of advertising information (such as a particular insertion order) using multiple XML Standards or vocabularies (such as IfraAdConnexion and SPACE/XML) and convey them in an AdsMLEnvelope, provided that:
    a. The trading partner agreement between the sender and recipient **MUST** allow for the transmission of alternative representations of the same advertising information
    b. All of the alternative representation of the information **MUST** be contained in a single Item
    c. All such Items **MUST** be identified as being alternative representations of the same advertising information
    d. The standard or vocabulary used for at least one of those Items **MUST** conform to the relevant trading party agreement between the sender and the recipient.

2. The recipient of an Envelope containing alternative representations of the same advertising information **SHALL** be free to use whichever of those alternative Items it chooses to use (or none of them, if none conforms to the relevant trading party agreement), and to discard the rest of them, without having opened or read any of the alternative representations.

10.3.3 Communication with internal systems

The only required interaction between an AdsMLEnvelope Processor and other internal systems (such as a booking system, page layout system, etc.) is that the AdsMLEnvelope Processor must be able to receive advertising information from those systems and send advertising information to those systems. The AdsMLEnvelope Processor is not required to exchange any other kinds of information with internal systems, for example, confirmation details about when and how a given Item was transmitted to a trading partner.

10.3.4 Creation of AdsMLEnvelope messages

An AdsMLEnvelope processor **SHALL** generate AdsMLEnvelope and AdsMLEnvelope Response messages that are valid according to the version of the AdsMLEnvelop standard that is referenced in each message.

10.3.5 Message logging

An AdsMLEnvelope Processor is not required to store persistent information about messages that it has sent or received, except to the extent necessary to support the immediate generation of an Envelope Response message (a mandatory capability), or to the extent necessary to support resending previously sent AdsMLEnvelope messages (an optional capability).

10.3.6 Message resending

1. When an AdsMLEnvelope Processor resends a previously sent message, it **SHALL** increment the "number of times this message has previously been transmitted" counter.

2. An AdsMLEnvelope Processor that has implemented "Store and resend until acknowledged" functionality **MUST** also have implemented the "Send and forget" functionality, and **MUST** use "Send and forget" as its default communications mode when exchanging AdsMLEnvelope messages with a system that has not affirmatively agreed to use the "Store and resend until acknowledged" functionality.

10.3.7 Processing model

AdsML is not intended to constrain an implementation to follow any particular processing model, provided that the AdsML implementation fulfills the operational requirements described in the relevant AdsML documentation.

10.3.8 Response requirement

An AdsMLEnvelope Processor **SHALL** act on the response-required value in an AdsML message, when present, by sending an acknowledgement Response message when requested to do so, even when this behavior contradicts the relevant trading partner agreement.

10.3.9 Redirection

1. An AdsMLEnvelope Processor **MUST NOT** redirect an Item back to its Sender.

2. Redirection can be based on any values contained in the Item's metadata as defined in the AdsMLEnvelope Specification, including Item-level user-defined Properties. However, redirection **MUST NOT** be based on information that is only found inside the Item's `ContentData` element.

10.3.10 System testing

1. An AdsMLEnvelope Processor **SHALL** support the transmission of test Envelope and test Response messages.

2. When an AdsMLEnvelope Processor receives a test Envelope, it **SHALL** generate a Response to that Envelope even if no response would have been required had it not been a test message.

10.3.11 Transmission/Response modes

1. An AdsML processor **MUST** support "Send and forget" functionality.

2. The "Store and resend until acknowledged" model is preferred over the "Send and forget" model and **SHOULD** be followed if possible.

3. An AdsML Processor **MUST ONLY** follow the "Store and resend until acknowledged" model when communicating with a trading partner that has explicitly agreed to follow this model.

10.3.12 Validation and feedback

1. An AdsMLEnvelope Processor **SHALL** ensure that an incoming AdsMLEnvelope message is valid according to the AdsMLEnvelope standard. This may be done either by performing XML Schema validation using the supplied XML Schema

and controlled vocabulary mechanisms, or by any other means that achieves the same results.

2. When an invalid Envelope is received, the AdsMLEnvelope Processor **SHALL** transmit a valid Response message to the sender informing them of the error(s).

10.3.13 Verification and feedback

An AdsMLEnvelope Processor **SHOULD** verify that the contents an incoming AdsMLEnvelope message conforms to the relevant agreements between its sender and recipient.

11 Appendix A: Acknowledgement for contributions to this document

Acknowledgement and thanks for contributions to this document are also due to:

- Members of the AdsML Technical Working Group, and including contributions from,
- Jay Cousins (RivCom Ltd) – jay.cousins@rivcom.com
- Adrian Davis (Quickcut) - adavis@quickcut.com
- Marcel Dumont (Rosetta) – marcel@rosetta.nl
- Joe Kirk (K Media Solutions) - kmedia@btinternet.com
- Christian Rohrbach (iware - PubliGroupe)
- Reviewers of the Last Call Working Draft,
- Hans Faye-Schjøll (Knowlex AS) schjoell@knowlex.no
- Israel Viente (Vio Worldwide Ltd.) israel_viente@il.vio.com
- Jon Simcox (Oppolis Software Ltd.) jon.simcox@oppolis.com

12 Appendix B: Intermediary and on-behalf-of business partners

As the word "Partnership" implies, it is assumed that in most cases any two business entities that are willing to exchange AdsMLEnvelope messages with each other will have some form of business relationship, and will "partner" in the exchange of messages. However, this does not mean that they must have a traditional business relationship in which one of them pays the other for its services.

Often, two business entities will need to exchange AdsMLEnvelope messages with each other because one of them is acting as the agent for a third party. For example, suppose A wishes to send ad content to B, but B hires C to act as its agent and receive all such materials. In this case A and B probably have a traditional business relationship, and B and C certainly have one, but there is no requirement that A and C have a direct business relationship.

On the other hand, in order to implement AdsMLEnvelope functionality between A and C, there is a requirement that A's and C's systems behave as if they had a relationship. A's AdsMLEnvelope system must "know" that it should redirect ad content from B to C, as well as what types and formats of information C will accept. C's AdsML system must "know" that A is a trusted sender and be willing to accept ad content from A.

In this scenario, because B is the only one of the three players that has a relationship with both A and C, it is up to B to ensure that A and C have the information they need. B must ensure that A knows what types and formats it can send, and the circumstances under which they should be redirected to C. It must also ensure that C is willing to accept such messages from A, and that C knows what to do with them when they arrive.

At the end of this process, the result (in terms of AdsML's requirements) should be much as if A and C had a direct business relationship with each other: each of them should have a PPA on file that enables them to communicate with each other in appropriate ways.

In the rest of this document we ignore these complexities and use the term "partner" to describe any two entities that wish to exchange AdsML documents.

