AdsML® Framework for E-Commerce Business Standards for Advertising

E-Commerce Usage Rules & Guidelines

Document Authors: AdsML Technical Working Group

Document ID: AdsML3.5-EcommerceUsage-AS-1

Document File Name: AdsML3.5-EcommerceUsage-AS.pdf

Document Status: Approved Specification

Document Date: 15 April 2010

Draft Number: 1

Copyright © 2010 AdsML Consortium. All rights reserved.

Table of Contents

Copyright © 2010 AdsML Consortium. All rights reserved.

Copyright © 2010 AdsML Consortium. All rights reserved.

1 AdsML Standard Documentation

1.1 Document status and copyright

This is an Approved Specification of the AdsML E-commerce Usage Rules & Guidelines.

Copyright © 2010 AdsML Consortium. All rights reserved. Information in this document is made available for the public good, may be used by third parties and may be reproduced and distributed, in whole and in part, provided acknowledgement is made to AdsML Consortium and provided it is accepted that AdsML Consortium rejects any liability for any loss of revenue, business or goodwill or indirect, special, consequential, incidental or punitive damages or expense arising from use of the information.

Copyright Acknowledgements: The AdsML Non-Exclusive License Agreement is based on the "Non-Exclusive License Agreement" on Page iii of "OpenTravel™ Alliance Message Specifications – Publication 2001A", September 27, 2001, Copyright © 2001. OpenTravel™ Alliance, Inc. The AdsML Code of Conduct is based on the "OTA Code of Conduct" on Page ix of "OpenTravel™ Alliance Message Specifications – Publication 2001A", September 27, 2001, Copyright © 2001. OpenTravel™ Alliance, Inc.

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,

Copyright © 2010 AdsML Consortium. All rights reserved.

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.

Copyright © 2010 AdsML Consortium. All rights reserved.

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 AdsML3.5-EcommerceUsage-AS-1, is freely available. It is located at the AdsML website at http://www.adsml.org/.

1.5 Purpose of this document

This document provides technical and business-process rules and guidelines about aspects of the AdsML standards that are common to all of them. It supplements the information found in the specifications for each of the specific standards and should be considered an integral part of the textual specification of those standards.

1.6 Audience

The intended audience for this document is any prospective user of AdsML, interested parties, and the AdsML Consortium.

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 is 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. A description of the entire document set can be found in the *ReadMeFirst* html file associated with this release of the Framework.

Copyright © 2010 AdsML Consortium. All rights reserved.

1.8 Definitions & conventions

1.8.1 Definitions of key words used in the specification

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 (See Section 9 References). 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`'.

In some cases, element names mentioned in usage guidelines and narrative text in this document do not include their namespace prefix. This simplification is provided in order to make the text easier to read. Element names in code fragments are always shown with their full namespace prefix.

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	Editor
3.5 – AS	April 15 2010	First Approved version for Framework 3.5 – previous change history removed	JC
3.0 – AS 1	May 30 2008	First Approved version – previous	TS

Copyright © 2010 AdsML Consortium. All rights reserved.

		change history removed	

1.10 Acknowledgements

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

Primary authorship and editing was performed by:

- Jay Cousins (RivCom) – jay.cousins@rivcom.com
- Tony Stewart (RivCom) – tony.stewart@rivcom.com
- Ulf Wingstedt (CNET/Svenska) – ulf.wingstedt@cnet.se

Additional material was provided by:

- Joe Kirk (K Media Solutions) - kmedia@btinternet.com

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 in the *AdsML Framework Overview* and 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 "AdsML Envelope" 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 the AdsML Envelope. Use of the AdsML Envelope, while encouraged, is optional, and the Item-Level standards can be used both inside and outside of the AdsML Envelope.

The AdsML standards are part of the AdsML "Framework". In practice this means that they share both an e-commerce philosophy and a resulting set of design principles; they use common element and attribute names and structures; and they support a common message choreography (i.e. the pattern by which messages are exchanged between trading partners).

This document provides technical and business-process rules and guidelines about aspects of the AdsML standards that are common to all of them. It supplements the information found in the specifications for each of the individual standards.

Information about elements and choreographic patterns which apply to a specific AdsML item-level standard can be found in the specification for that standard.

Copyright © 2010 AdsML Consortium. All rights reserved.

Information about elements which are part of the AdsML Type Library, and are imported into each of the AdsML item-level standards, can be found in the type library specification.

2.1 Implement only what you need

The AdsML Framework aims to provide advertisers, publishers, broadcasters and their suppliers with a consistent toolkit of standards, messages and transactions that can be used to automate every aspect of the advertising supply chain, in any media, anywhere in the world. This means that even though it is still incomplete, the Framework already contains more standards and message types, and can convey more types of information, than any single organization is likely to need.

In order to implement AdsML-based e-commerce, therefore, trading partners and their vendors (or industry associations acting on their behalf) are expected to review the AdsML Framework and decide:

1. Will they use the AdsML Envelope?
2. Which AdsML standards will they implement within their particular region or business activity?
3. Which business transactions will they support?
4. Which types of information will they include in their messages?
5. Which information will be conveyed in machine-processable elements, vs. which will be sent as unstructured text that requires human handling?
6. For which machine-processable elements will they require use of a particular controlled vocabulary?

Each AdsML standard defines its mandatory and optional components, and where appropriate, each provides a Configuration Checklist to help users discuss and agree on the features and functionality that they will implement. These implementation decisions can be agreed privately between the trading partners, and/or codified in a formal "profile" which is made publicly available in order to encourage interoperability.

Based on their customers' implementation decisions, vendors can decide which types of AdsML functionality to implement in their systems. In order to market a system's AdsML capabilities, a vendor might indicate that it supports specific named Profiles, and/or the vendor might use the relevant Configuration Checklist(s) to describe the supported capabilities.

Further information about these concepts can be found in *AdsML E-Commerce Usage Rules & Guidelines* (this document), in the *Advertising Components Interactions Analysis*, and in the Specification for each standard.

NOTE: Even though you can implement just those portions that you need, all of the standards and features in the AdsML Framework are designed to work together as a cohesive whole, in that they share common technical components and a common approach to advertising e-commerce that makes them "AdsML".

3 AdsML architecture and technical approach

3.1 AdsML and XML

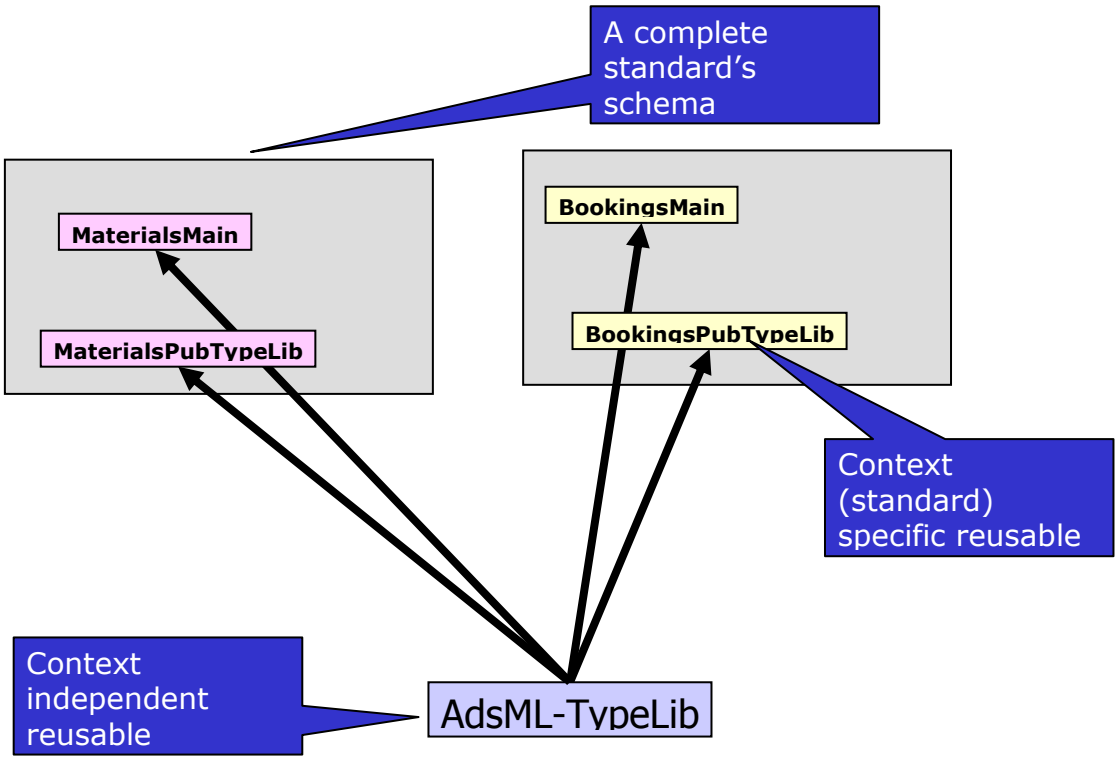
The AdsML standards are applications of the Extensible Markup Language (XML). Each AdsML standard is a vocabulary of XML elements and attributes that defines and structures a messaging envelope for exchanging advertising data. An XML

Copyright © 2010 AdsML Consortium. All rights reserved.

Schema formally describes the structure of the AdsML data model in the XML data model of elements and attributes, the XML data model creating a hierarchical 'tree' of nodes where nodes have a parent and child node relationship. When using AdsML, advertising data is serialized for exchange in XML syntax as XML documents using the AdsML vocabulary of elements and attributes. The advertising data is then exchanged between trading partners as an XML data stream that when parsed ('deserialized') can be validated against the relevant AdsML schema to ensure that the data and any associated constraints upon that data required by the AdsML data model and expressed by the AdsML Schema are correctly enforced.

3.2 Schema architecture

As shown in the diagram below, each AdsML item-level standard (for example, AdsMLBookings, AdsMLMaterials and AdsMLStructuredDescriptions) is defined in a pair of schemas: a Main schema and a Public Type Library. Each standard's Main schema contains elements that can only be used within that particular standard. The matching Public Type Library contains elements that are only meaningful in the context of the standard in question (i.e. they are intrinsically related to the information involved in, for example, a Booking or a Materials Delivery), but unlike the elements in the Main schema, they are allowed to be used in other standards. And finally, AdsML also provides a global public type library, the AdsMLTypeLibrary, which contains an extensive set of context-independent components that can be used in any or all of the AdsML standards.



In practice, the Main schema of each standard uses the XML Schema "Include" mechanism to pull in its matching Public Type Library, and uses the "Import" mechanism to incorporate the global AdsMLTypeLibrary as well as any other standard's Public Type Library that it requires.

For example, AdsMLBookings *includes* the AdsMLBookings Public Type Library but *imports* the AdsMLTypeLibrary. It also *imports* the AdsMLMaterials public type library in order to be able to support content delivery in a booking message.

Copyright © 2010 AdsML Consortium. All rights reserved.

3.2.1 Relationship to namespaces

When an AdsML schema *includes* its own Public Type Library, the elements in the Public Type Library inherit the same namespace as the main schema. In the case of the AdsML item-level standards, the Main schema (and its included Public Type Library) is assigned to the default namespace, which means that in a message conforming to that schema, elements which come from either the Main schema or its associated Type Library do not have a namespace identifier. For example, the root element in an AdsMLBookings message appears as, simply, "`<AdsMLBookings>`".

When an AdsML schema *imports* elements from the AdsMLTypeLibrary, those elements are assigned the "adsml" namespace. This means that in a given message, the names of all the elements that came from the AdsMLTypeLibrary are preceded with the string "adsml:", for example, "`<adsml:Header>`."

And finally, when an AdsML schema *imports* elements from the Public Type Library associated with another AdsML standard, it assigns those elements a namespace whose name is in the format "adsml-*xx*", where *xx* is an abbreviation of the source standard. For example, in AdsMLBookings, the names of elements that have been imported from the AdsMLMaterials Public Type Library begin with "adsml-ma:", for example, "`<adsml-ma:AdContent>`".

3.2.2 Locating documentation based on the namespace

The AdsML documentation set is organized so as to correspond to the schema architecture.

- If an element's structure is defined in the Main or PublicTypeLibrary schema for a given AdsML item-level standard, then its documentation will be found in the Specification for that standard. Each item-level Specification contains both usage rules and guidelines (part 1 of the specification), and also an element-by-element reference guide (part 2).
- If an element's schema definition is in the AdsMLTypeLibrary, then both its textual definition and any high-level usage rules and guidelines will be found in the *AdsML Type Library* specification.

Because an element's schema location can be inferred from its namespace, the namespaces in an AdsML instance message provide a valuable clue as to where to find its documentation. For example, in an AdsMLBookings message:

Element name	**Schema**	**Critical Usage Rules (if any)**	**Textual Definition**
`AdsMLBookings`, `TotalPrice`, `ProofType`	AdsMLBookingsMain.xsd *or* AdsMLBookings-PublicTypeLibrary.xsd	AdsMLBookings Specification, part 1	AdsMLBookings Specification, part 2
`adsml:Header`, `adsml:Status`, `adsml:Role`	AdsMLTypeLibrary.xsd	AdsMLTypeLibrary Specification	AdsMLTypeLibrary Specification
`adsml-ma:AdContent`, `adsml-ma:Rendering`	AdsMLMaterials-Main.xsd *or* AdsMLMaterials-	AdsMLMaterials Specification, part 1	AdsMLMaterials Specification, part 2

Copyright © 2010 AdsML Consortium. All rights reserved.

	PublicTypeLibrary.xsd		

Note that in a Materials message the context will be reversed: elements and types defined in AdsMLMaterials Main and the AdsMLMaterials Public Type library will be assigned the default namespace (with no identifier), and elements and types that are imported from AdsMLBookings will be assigned the "adsml-bo" namespace.

3.2.3 Schema filenames

The schema files from a particular standard are named as follows:

```
AdsMLBookings-2.5-Main-AS.xsd

AdsMLBookings-2.5-PublicTypeLibrary-AS.xsd
```

The format starts with the name of the standard, e.g. "AdsMLBookings", followed by the current version number and the name of the schema within the standard. The last two characters provide the status of the standard as either PS (Proposed Standard) or AS (Approved Standard) for public releases (internal working document have status code WD for Working Draft).

3.2.4 Version, identification, and language

The version, identification number, and language of each AdsML schema are recorded using the optional *version*, *id*, and *xml:lang* attributes of the schema's root <xs:schema> element.

Name	Occurs	Type	Description
version	?		The *version* attribute is used to record the version number of the schema using a major and minor version number, and optionally using a version letter in the format X.Y.a, where 'X' is the major version number, 'Y' is the minor version number, and, if present, 'a' indicates a draft schema version number. The version number of the AdsML main and supporting schemas will always be kept in sync, i.e. version 'x.y' of the AdsML main schema will always import version 'x.y' of the AdsML supporting schemas.
id	?		The *id* attribute is used to record a unique identifier number for the schema (so that the schema can be unambiguously identified).
xml:lang	?		The *xml:lang* attribute is used to record the natural human language used in the AdsML Schema itself. The default language of the AdsML Schema is American English. Language is recorded using ISO coding as defined by *IETF Request for Comment 3066*.

For example, the AdsMLEnvelope schema records the version number, identifier, and default language using these attributes as illustrated below,

```
<xs:schema targetNamespace="http://www.adsml.org/adsmlenvelope/1.1"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

Copyright © 2010 AdsML Consortium. All rights reserved.

```
xmlns="http://www.adsml.org/adsmlenvelope/1.1"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.1.0" id=" AdsMLEnvelope-1.1.0-Main-AS" xml:lang="en-us">
```

3.3 Data types

All data types used by in the AdsML XML Schemas are either XML Schema built-in data types, or derived from those built-in types using XML Schema type derivation by either the AdsML Schema or, in cases where users specify their own controlled vocabulary data types, in a user-defined AdsML extension schema.

AdsML specifies no restriction on the data types that may be carried inside an AdsMLEnvelope message as data inside the `ContentData` element, although such data has to conform to the constraints applicable to character data in XML documents as defined by the XML 1.0 specification[1].

3.3.1 Data typing – 'weak' vs. 'strong'

The AdsML policy is to use 'weak' rather than 'strong' data typing for operational reasons grounded in providing user flexibility and facilitating the implementation and use of AdsML. Consequently, 'weak' data types are used where 'strong' data types could have been specified in those places where any error would be non-catastrophic and so not significant enough to upset the workflow.

3.4 Mandatory vs. required, blanks vs. nulls

In general, an AdsML message **SHOULD NOT** contain any empty elements or attributes. Empty elements and attributes are potentially ambiguous because they do not indicate whether the data was accidentally or deliberately omitted. To prevent this ambiguity AdsML has defined the following rules governing when and how empty or null values may be transmitted.

Each element or attribute in an AdsML standard is defined in its schema as being either mandatory or optional. Some elements are additionally defined as being "nillable" (*nillable="true"*), which indicates that in an XML message instance they may carry the attribute *xsi:nil*. These constraints should be interpreted as follows:

- If an element or attribute is defined as mandatory, it **MUST** be present in the corresponding XML message instance.
- If an element or attribute is defined as optional, it **MAY** be present in the corresponding XML message instance, or it **MAY** be omitted.
- The presence of the attribute *xsi:nil="true"* on an element in an XML message instance indicates that the value of the element in question is missing or unknown.

This leaves open the questions of what is signified by an empty element in an AdsML message, and how to differentiate between null (missing or unknown) values and deliberately blank/zero values. The following rules and guidelines apply:

- If the value of a non-mandatory element or attribute is missing, unknown or not applicable to the current workflow, the entire element or attribute **SHOULD** be omitted from the message instance. The absence of an element or attribute from an AdsML message instance **SHOULD NOT** be interpreted

[1] For allowable text and character content in XML see the following sections of the XML specification: Section 2.2 Characters, Section 2.4 Character Data and Markup, and Section 2.7 CDATA sections.

Copyright © 2010 AdsML Consortium. All rights reserved.

by the recipient as representing significant business information unless explicitly agreed otherwise between the trading partners.

- If an element or attribute is present in an AdsML message instance, that element or attribute **MUST** either:
  - a) contain a valid business value, or
  - b) contain an agreed-upon dummy value to indicate that it does not contain valid business information (see below), or
  - c) carry only the attribute `xsi:nil` with the value of `"true"`, which indicates that it does not contain a valid business value. (This option is only available for elements that are defined as *nillable* in the AdsML schema.)
- The presence of an empty element with the attribute `xsi:nil="true"`, or the transmission of `"-0-"` as element or attribute content, indicates that the value of that element or attribute is missing or unknown and **SHOULD NOT** be considered to represent valid business information.
- The presence of an empty element or attribute in a message instance that does not contain the attribute value `xsi:nil="true"` **SHOULD** be assumed by the receiver to represent valid business information, i.e. that the value in question literally does not exist. For example, the presence of an empty `Email` address element in a `Contact` structure would indicate that the `Contact` in question is known not to possess an email address. Because of the potential for misunderstanding, transmission of such "known not to exist" values is strongly discouraged.

The AdsML Technical Working Group has attempted to define the schemas so that they require as few mandatory elements and attributes as possible. However, there may still be valid circumstances in which the value of a mandatory element is either missing or unknown. In this case the mandatory element or attribute should be explicitly identified as not containing meaningful business information by following the following sequence of guidelines:

1. If the mandatory element or attribute is defined as an AdsML `QID` type, the sender **SHOULD** attempt to populate it with a valid business value rather than a dummy value.
2. If the value of the mandatory element still cannot be provided and it is defined as *nillable*, the sender **SHOULD** transmit it as an empty element with the single attribute `xsi:nil="true"`.
3. If the value of a mandatory element or attribute that is capable of conveying text strings cannot be provided and it is not defined as *nillable*, the sender **SHOULD** populate it with the dummy value `"-0-"`.

Note that these rules do not constrain the *interpretation* of null values in a given message, in particular whether the null data was "missing" vs. "unknown" vs. "inapplicable". The semantics of message content (including the presence or absence of specific values) is determined by a combination of the relevant schema and specification, the TPA between the trading partners, the nature of the element or attribute in question and the context in which it occurs.

3.5 Message processing

3.5.1 Asynchronous messaging model

The AdsML standard supports an asynchronous messaging model. The main advantage to this approach is that it minimizes locking of system resources and

Copyright © 2010 AdsML Consortium. All rights reserved.

transactions, enabling systems to continue operations after a message has been sent. This is a prerequisite for handling very long-lived advertisement business processes.

Many earlier e-commerce messaging models were often synchronous and/or built with a central controlling node (client-server or master-slave models). Even though these models can be implemented using the AdsML standards, AdsML also supports the more flexible asynchronous approach to message exchange. As a consequence, AdsML transactional messaging can be implemented using, for instance, synchronous remote procedure calls, or asynchronous e-mail based services, or even file-based transfers using hot folders, according to the requirements of the communication parties.

In particular, the AdsML processing model has the following characteristics:

1. An AdsML system **SHOULD NOT** require an immediate response to a request. A response might arrive after a few seconds, minutes, days or even months – for instance, consider an ad order that requires manual handling. But although responses are allowed to arrive late, it is strongly **RECOMMENDED** that responses should be sent as soon as possible.
2. Messages are not guaranteed to arrive in the same sequence as they were transmitted. Due to the underlying infrastructure for message exchange, messages can take different routes, be delayed etc.
3. Both communication parties are "peers", i.e. none is said to be in control over the other (no "master-slave" model). Both parties are able to send requests simultaneously, although conflicts should be avoided.
4. A communication party **MAY** send further messages without waiting for responses for previous requests.

The AdsML standards include the means to handle an asynchronous operating environment, in the form of metadata that, if properly used, allow communication parties to detect and manage conflicts. However, communication parties **MAY** in their mutual agreement (the TPA, or Trading Partner Agreement) define a less flexible model that, for instance, states that a new request is not allowed to be sent before a response to an earlier request has been received.

AdsML does not define how conflict resolution should be handled in case of, for instance, both parties requesting conflicting updates[2]. Conflict resolution will be highly application specific and **SHOULD** be defined as a part of the TPA.

3.5.2 Message re-sending

The *sendCount* attribute is used to signal when a message is a copy of an earlier transmission. Its default value is '1'. If a system re-sends a message, it **MUST** indicate that this is a copy of a previous message by incrementing the *sendCount* attribute. With each resending of a message, the *sendCount* attribute **MUST** be incremented by 1 so that the value of this attribute always reflects the number of times that a message has been sent. Therefore, a *sendCount* value greater than 1 indicates that the message is a copy of an earlier transmission.

With the exception of the *sendCount*, *transmissionDateTime* and *transmissionSequence* attributes, the content of a re-sent message **MUST** be an exact copy of the first transmitted message (including the *transmissionID* attribute).

[2] For example, conflict resolution would be required if a new order were being processed by a seller of ad space, and the buyer of ad space submitted a change to that order before having received a Business Significant Response to it. The timing of those messages might lead to confusion about the exact current state of the order.

Copyright © 2010 AdsML Consortium. All rights reserved.

3.5.3 Duplicate messages

Once a message has been received, any later re-transmissions of that message **MUST** be discarded without further processing.

If two messages with the same *transmissionID* and the same *sendCount* are received, they are presumed to be duplicate copies of the same transmission containing the same business content. The recipient **MUST** only process one of them, except that the recipient **MAY** choose to generate an administrative response to the second message notifying the sender that it has transmitted duplicate messages.

3.6 Globally unique identifiers

3.6.1 Types of QIDs

Each of the AdsML message formats contains several identifiers based on the datatype format for qualified identifiers, the `QIDType`, which is defined in the AdsMLTypeLibrary. These identifiers are called qualified identifiers (QIDs) because their construction rules guarantee global uniqueness, and they are used as, for instance, technical values which support the AdsML messaging "machinery" (e.g. *transmissionID*, *messageID* and *inResponseToMessageID*, which are used to uniquely identify AdsML messages, create handshakes between AdsML systems, enable re-sending of messages, and guard against simultaneous updates) and one or more business-significant QIDs which uniquely identify the information being discussed in that message (e.g. `QuotationIdentifier`, `BookingIdentifier`, `PlacementIdentifier`, `MaterialsIdentifier`, etc.).

(A variant of the `QIDType` is also used as the basis for the `SchemaProfile` element and *schemaProfile* attribute, which are described later in this document.)

3.6.2 Structural rules

The structural rules for an element or attribute based on the AdsML `QIDType` are described in the AdsML Type Library specification and **MUST** be followed.

According to those rules, a sample QID value might look like this: "myorganization.com:2005-01-01:1234567890." The first part of the QID is an internet domain name that was controlled by the organization generating the ID at the time the message was created. The second part is a date on which the domain name belonged to the organization issuing the ID. (The date in this section does not have to be a current date. All QIDs issued by a given organization can begin with the same domain name and date, provided that the organization continues to control the domain name.) And the third part is a string which the generating organization guarantees to be unique within the set of all QIDs that share the specified domain name and date. This results in what should be a globally unique identifier.

It is important to note that the third part of the QID format does not have to be a meaningless or randomly-generated string. When creating a QID for a business object (such as a booking or set of materials) for which the organization already maintains an ID, the organization creating the QID can simply use their existing ID as the third part of the QID string, provided that they can guarantee not to use this same value in any other QID for the same purpose. Alternatively, the organization can generate a unique string of any kind, for instance a Windows GUID.

Copyright © 2010 AdsML Consortium. All rights reserved.

A QID **SHOULD** be treated by all parties as a unique string with no inherent internal meaning, and **SHOULD NOT** be parsed or decomposed in order to extract just a part of it for further processing. The AdsML message formats always carry additional, optional structures to convey the business significant identifiers that are actually used within each organization. (For example, the buyer's order number, the seller's order number, the production house's artwork ID, or any other identifiers that are used in the relevant business operations.) These are called "Auxiliary References" and can be found in an element adjacent to an element containing an AdsML QID. It is **RECOMMENDED** that organizations populate the auxiliary references with the business significant identifiers that they actually use in their day to day operations, so as to facilitate better communications between trading partners.

3.6.3 Scope of uniqueness

When its structural rules are followed, the GUIDType format guarantees that QIDs created by different organizations do not overlap. It is, however, up to each organization to ensure that a QID is not reused as an identifier for different objects, in different contexts. Ideally, a QID value should be unique across the entire set of QIDs generated by a given organization. Therefore, in theory, a *messageID* issued by an organization should never have the same value as a MaterialsIdentifier issued by that same organization, which in turn should never have the same value as a BookingsIdentifier issued by that organization.

In reality, it is often necessary to limit the uniqueness of a business-significant QID to a smaller domain. For example, an organization which does not internally differentiate between a placement and an insertion may wish to use the same value for both a PlacementIdentifier and the single InsertionIdentifier within that placement. Or, the algorithm for generating bookings-related QIDs may not be able to guarantee that they are unique compared to any materials-related QIDs generated by the same organization.

However, none of these concerns can or should affect the values used for the technical IDs, such as *transmissionID* and *messageID*, which enable the AdsML machinery.

So the guidelines are:

- QID values used for an AdsML technical ID **MUST** be globally unique across the entire set of technical QIDs generated by that organization
- All other types of QID values **MUST** be unique across the set of QIDs of that specific type (i.e. used as identifier for a particular object, context and purpose) generated by the organization, and **SHOULD** be unique across the set of all QIDs generated by that organization if possible.

3.7 Validation

3.7.1 Overview: schema validation vs. programmatic validation

The AdsML Framework provides a set of XML schemas and an associated extension mechanism that can be used to define both the structure and (to the extent possible using XML's schema facilities) the allowable information contents of an AdsML message. The Framework also describes a PPA[3] mechanism wherein the PPA contains the set of computer-readable information that needs to be available to an AdsML processor and that controls some aspects of its operations.

[3] The machine-processable portion of the Trading Partner Agreement

Copyright © 2010 AdsML Consortium. All rights reserved.

There is a distinct overlap between the information content of an AdsML Schema (including optional user extensions) and an AdsML PPA, since a user-extended AdsML Schema is, in essence, a formalized part of a PPA.

So, for what purposes would one use an AdsML schema, and how does this relate to a PPA? This section provides a brief, non-normative overview of how the AdsML Technical Working Group expects that they will be used. It is followed by several sections contain normative rules pertaining to validation of AdsML messages.

During production, an AdsML schema would normally be used for validation of the structure and content of incoming AdsML messages, whether they are AdsMLEnvelopes or messages conforming to one of the item-level e-commerce standards. (Note that users are not required to use the AdsML Schema for this purpose, provided that they implement validation by some mechanism that achieves the same results.) For each type of information being exchanged, each communications partner would set up a single schema that restricts the types of information it is willing to receive. This schema (including user-defined extensions) would be used to validate all incoming messages from all communications partners.

Therefore, a trading partner's schema would not necessarily contain the same schema extensions as those in the AdsML schemas that the communications partners set up for their own systems, and communications partners would not normally exchange their actual schemas or PPAs with each other. The only business requirement is that the recipient of a message must be able to validate every value and type of information sent by the sender, and therefore their respective AdsML schemas must each permit these kinds of information to be included in the messages that they exchange.

Schema validation of an incoming AdsML message is expected to be a front-line operation that is performed as soon as possible after the message has been received. It is not necessarily the only validation, however. Depending on the recipient's business requirements, the software receiving this message may apply additional validation and verification operations in order to ensure that the contents of the message conform to its business rules and to the relevant PPA. The PPA may contain rules that cannot be validated by AdsML schema processing – for example, rules that are driven by the presence or absence of complex combinations of metadata within the message. If the message recipient wishes to validate against these rules, it will need to perform additional validation that is driven directly by the PPA (or perhaps by another mechanism such as an XSLT stylesheet) rather than by the AdsML schema.

On the outbound side, when creating an Item and then addressing it, both the AdsML Item Creator and the AdsML Item Redirector[4] must have access to relevant PPA information about the agreement between the sender and the intended recipient of this information. For example, the encryption and encoding put in place by the Item Creator must be acceptable to the Item Content Unpacker that will eventually receive it; similarly, the types and XML formats of the information must be acceptable. Even though the user may have created an AdsML schema for validation of incoming messages that contains all of this information and is machine-readable, and therefore could also be used to drive outbound processing, the format of an AdsML schema is not optimized for this kind of use. Therefore, we expect that implementers of AdsML systems will design their own information format for storing the PPA content, even though much of that content is also available in the AdsML schema.

[4] Both the "AdsML Item Creator" and "AdsML Item Redirector" are components of an AdsML Processor, as described in *AdsML Envelope Processing Model, Usage Rules & Guidelines*.

Copyright © 2010 AdsML Consortium. All rights reserved.

During system setup and testing, users may use their AdsML schema to validate outbound messages that they are about to send to a communications partner, in order to ensure that their system is functioning properly and generating the correct message structure and content. However, it is expected that once a system goes into production, the need for such outbound validation will have greatly diminished and most users will remove this step from their message creation process.

To summarize: the AdsML schema is intended to be used for inbound validation; because it is machine processable it could also be used by the outbound Item Creator, but the implementer is free to provide this information to the Item Creator in other ways.

3.7.2 Validation rules

3.7.2.1 Validation and Schema Location

A trading partner **MUST NOT** send any invalid AdsML messages. However, use of XML Schema based validation of production messages in runtime is **OPTIONAL**. Systems are allowed to use any available approach to ensure that their output is valid.

For production messages, a schema location **SHOULD NOT** be given in document instances using the *xsi:schemaLocation* attribute. Systems are **REQUIRED** to be able to identify which schema a particular document instance belongs to by reading the mandatory *adsml:schemaVersion* attribute.

3.7.2.2 No empty values for elements and attributes

All elements and attributes that appear in an instance **MUST** take a value, i.e. are not allowed to be empty. The only exception to this is the case of elements which are defined with an empty content model or where the presence of the attribute *xsi:nil="true"* on an element indicates that the value of the element in question is missing or unknown.

3.7.2.3 Fixed and Default values

All fixed or default values specified for elements or attributes in the schema **MUST** be present in an XML document instance conforming to that schema; schema validation and the post-schema-validation infoset (PSVI) **SHOULD NOT** be relied upon in order to make fixed or default values available for processing.

This restriction is imposed so that a particular mode of validation (XML Schema validation and the PSVI) is not relied upon to ensure that all data content of a message is present in an instance messages. This allows for non-XML Schema validation of an instance.

This constraint is enforced in the schema by specifying attributes that carry fixed values with a 'use' of required, by not specifying default values, and by the policy that element content should not be empty in instances.

3.8 Sequence of elements

The sequence of elements in an AdsML message is not significant. An AdsML user **MUST NOT** use element sequence to imply e.g. parent/child or anchor/reference relationships, etc.

Copyright © 2010 AdsML Consortium. All rights reserved.

3.9 Customization and extensions

An AdsML user **MUST** only extend the lists of allowable values for elements and attributes in an AdsML message by means of the extensibility mechanisms that are defined in the relevant AdsML standard.

An AdsML user **MUST** only define custom Properties for use in an AdsML message by means of the mechanisms for doing so that are defined in the relevant AdsML standard.

An AdsML user **MUST NOT** extend the structure or contents of an AdsML message in any way other than those defined in the relevant AdsML standard.

3.10 Internationalization

As an XML-based technology, AdsML provides support for Internationalization in line with that provided by XML. Dates and times in AdsML are recorded in ISO 8601 Extended Format using the built-in XML Schema data types for date and time, `xs:dateTime`. Language is recorded as a string conformant to RFC 1766 (See Section 9 References) using the built-in XML Schema data type for language, `xs:language`. AdsML supports character sets that conform to ISO/IEC 10646.

3.11 Security and encryption

The AdsML Item-level standards define XML message formats that can be used to transmit many kinds of sensitive information, from invoice and payment totals to advertiser's credit card details. Users and implementers of AdsML systems should be aware that XML is inherently human-readable, and so sensitive data such as credit card details can potentially be read in transit.

It is the responsibility of organizations sending and receiving AdsML messages to ensure that the messages are suitably protected and/or encrypted while traveling between the sender and the recipient. This is normally accomplished at a "higher level" than the AdsML business message, for example, by encrypting the envelope or channel within which the message travels. The AdsML Item-level standards are designed to work successfully within such an environment, but do not themselves provide any special mechanisms or instructions for implementing the necessary security.

4 E-commerce messages

Note: The information in this section applies only to the AdsML item-level standards; it does not apply to the AdsMLEnvelope standard.

4.1 Categories of messages

An AdsML e-commerce message represents an exchange between two parties in the advertising business chain. They can be of two kinds: business-significant messages, and administrative responses.

4.1.1 Business-significant messages

Business-significant messages directly affect either the information stored in a company's systems or its normal workflow processes. Messages in this category typically fall into one of four sub-categories:

i. Request for action or information

Copyright © 2010 AdsML Consortium. All rights reserved.

ii. Provision of information or content, including additional or corrected information
iii. Confirmation that an action has been carried out
iv. Indication that a requested action cannot be carried out, either because of business constraints (e.g. "The date you want is not available") or because the message requesting the action contained insufficient information (e.g. "You failed to specify the publication date")

Many e-commerce interactions consist of exchanges of two or more business-significant messages following pre-set patterns. For example, when a buyer of advertising sends a message to a seller of advertising requesting a Booking ("AD-O"), the publisher will in due course respond with a Booking Response message ("AD-OR") that either accepts the Booking or indicates that it could not be accepted. Both of these are business-significant messages, because they have a direct impact on the data stored in each party's systems.

Other types of business-significant messages include status enquiry messages and the responses to those enquiries, and business-level exception messages indicating a failure to carry out a requested action. Status enquiry messages are business-significant because they are triggered by and interact with normal workflow processes. Business-level exception messages typically fall into the category of "You didn't give me enough information, so I am unable to process your order." They represent an exception to normal processing, but they are not error messages, and they, too, directly impact business workflows.

NOTE: AdsML does not assume that the exchange of business-significant messages will occur in any particular timeframe. As discussed in the section on asynchronous messaging, substantial time may elapse between the sending of a given message and the receipt of its business-significant response.

4.1.2 Administrative responses

Administrative responses are replies to business-significant messages that do not directly affect business information or workflow processes. An administrative response indicates that "Your message was received" and whether it contained technical errors.

Every business-significant AdsML message **SHOULD** be replied to first with an administrative response, and then, if appropriate, with a business response to that message.

In situations when no business-significant message would be appropriate, the administrative response serves to terminate the message exchange sequence. For example, when a buyer of advertising receives a Booking Response message ("AD-OR") containing the business-significant information, "Your order has been accepted", no further business-significant message is necessary. Both parties have recorded the booking in their systems. However, in most environments the buyer of advertising will send an Administrative Response back to the seller of advertising stating that the seller's AD-OR message had been received. This provides a "handshake" that closes the e-commerce loop, and assures both parties that the transaction has been recorded in both of their systems.

4.1.2.1 Processing rules

In a typical AdsML architecture, the Administrative Response message may be generated by an AdsML message processor or generic e-commerce gateway *before* the business information in the incoming message is loaded into the recipient business application (e.g. a buying system, sales system, production system, etc). The Administrative Response message **SHOULD** be sent immediately,

Copyright © 2010 AdsML Consortium. All rights reserved.

and its content **MUST** be based entirely on technical validations. Later the business application may, if appropriate, send a business-significant response which reflects the results of further processing and business-specific validations. This response will be conveyed in a primary AdsML message as defined in the *Advertisement Components Interaction Analysis*, e.g. an AD-OR Order Response or an AM-MR Ad Material response, rather than an Administrative Response.

Similarly, when an Administrative Response is received by the trading partner that sent the original business message, it may well be processed by that party's AdsML message processor or generic e-commerce gateway. In this case the contents of the Administrative Response may never reach the business application that generated the original AdsML message or the business people who use that application.

The AdsML Administrative Response messages are designed to support this architectural approach. The information contained in an Administrative Response **MUST** be based entirely on a technical analysis of the incoming message and **MUST NOT** convey business information that needs to reach the business application or business users who sent the message to which it is a response.

4.1.3 Message codes and structures

The AdsML Framework assigns a unique code for each type of business message, including business-significant "response" and "receipt" messages. Each AdsML business message is identified by a message code that specifies whether the message is, for instance, an ad order, an ad order change, a materials delivery, or responses to these messages.

The message type of a business-significant message is expressed as a code value in the *messageCode* attribute in the header of the message, and its structure is determined by use of an element which has the same name as the type of message in question, in a schema CHOICE near the root of the message structure. (For example, an Ad Order message is always assigned a message code of "AD-O" and its structure is determined by use of the `AdOrder` element in the schema.) The message type names and code values are defined in the *Advertising Component Interactions Analysis* within the AdsML Framework.

When a business-significant message is sent in response to another AdsML message, the message code of the incoming message that triggered this response is conveyed in the *`inResponseToMessageCode`* attribute in the response message's header. For example, the response to a new order and the response to a request to cancel an order are both conveyed in "AD-OR" (order response) messages, but their *`inResponseToMessageCode`* values will be "AD-O" (new order) and "AD-OX" (cancellation request), respectively. The presence of this attribute value in all response messages enables recipients more easily to route and process incoming response messages based on the type of request to which they are a response.

An AdsML administrative response message is always assigned the same *messageCode* as the message to which it is a response, but it uses a different structure. In this case a separate attribute, *messageClass,* identifies that it is an administrative response rather than a business message. The structure of administrative response messages is defined by use of the `adsml:AdministrativeResponse` element, rather than the element (such as `AdOrder` in the example above) which defined the structure of the business-significant message to which it is a response.

To summarize:

Copyright © 2010 AdsML Consortium. All rights reserved.

A business-significant message **MUST** be assigned the *messageCode* value defined for that type of message in the *Advertising Components Interactions Analysis*, it **MUST** use the structure defined by the element whose name most closely matches the name for that type of message, and its *messageClass* **MUST** be "BusinessTransaction".

An administrative response message **MUST** have the same *messageCode* value as the business-significant message to which it is a response, its structure **MUST** be based on the adsml:AdministrativeResponse element, and its *messageClass* **MUST** be either "MessageReceivedAcknowledgment" or "TechnicalError".

4.1.4 ZZ-Error responses to catastrophic errors

A special type of Administrative Response message is the "ZZ-Error" message. ZZ-Error messages are used for handling catastrophic errors where it is not possible to respond with an error message of the same Type (perhaps because the Type of the incoming message has been omitted or become corrupted).

Structurally, a ZZ-Error message is identical to any other administrative response; however, its *messageClass* is always "TechnicalError" and its *messageCode* is always "ZZ-Error". The body of the message can then be used to provide more information about the error that occurred.

4.2 Message Choreography

In order to ensure interoperability, it is essential that systems exchanging AdsML item-level messages have the same view on which messages to send, and which to expect to receive.

The *Advertising Component Interactions Analysis* defines all of the messages that are within scope for AdsML, and for each one, indicates the type of business response message that a recipient of such a message would normally be expected to send (for example, whether they should issue a Response). However, it does not stipulate the sequence in which these messages should be exchanged.

The AdsML message choreography described here takes the next step, by categorizing the types of messages defined in the *Advertising Component Interactions Analysis* and providing a set of recommended patterns for exchanging them. These choreographic guidelines apply equally both to users of AdsML Item-level message formats such as the AdsML Bookings and Materials standards, and to users of other, non-AdsML formats such as IfraAdConnexion.

Many of the AdsML e-commerce standards go further, in that for each type of message that can be exchanged, they specify a detailed set of choreography patterns. Please see each standard's documentation for further information about that standard's recommended or required choreography.

These recommendations apply only to the Item-level messages that are exchanged between the trading partners' software systems. They do not apply to the use of the AdsML Envelope. Message exchange patterns for the AdsML Envelope are defined in *Envelope Processing Model, Addressing and Operational Conformance*.

4.2.1 Request-response vs. datagram communications

The preferred messaging model for all AdsML e-commerce communications is the Request-Response model as described below.

However, since legacy applications may have limited ability to receive or generate e-commerce messages, it is also possible to use a model where only requests and

Copyright © 2010 AdsML Consortium. All rights reserved.

possibly administrative responses are transmitted, assuming an acceptance on the receiver's side. If a problem occurs when a message cannot be accepted, it has to be solved manually. This kind of model is called a datagram model.

Typically, datagram communications are implemented because only one of the two communicating parties is capable of receiving e-commerce messages and automatically entering them into a workflow. (In practice, it is usually easier for a legacy application to transmit an e-commerce message than to receive one.) In such cases, the parties may implement datagram communications in one direction: from the party that is less capable of receiving e-commerce messages, to the party that is more capable of receiving them. It is also possible for communications partners to implement bilateral datagram communications, where some classes of messages (for example, ad orders, changes and cancellations) are sent as datagrams in one direction, and others (for example, order status messages) are sent as datagrams in the other direction. Parties wishing to use the datagram model must agree on which direction(s) of datagram messaging they will support.

As a summary:

1. Implementations of AdsML e-commerce standards **SHOULD** apply the full Request-Response model
2. If agreed by communication parties, implementations **MAY** use a datagram model (no business level responses required), and if so, they must also agree on which direction(s) of datagram messaging they will support.

In both cases, the ability to send an Administrative Response is considered a fundamental capability of an AdsML-compliant system that **SHOULD** be supported.

4.2.1.1 Request-Response pattern guidelines

The following guidelines for the exchange of Item-level messages following the request-response model are strongly **RECOMMENDED**:

1. Each message exchange begins with the transmission of a business-significant message from one partner to another. The nature of the recipient's response depends on whether the incoming message contains technical errors, and if not, whether a business-significant response message has been defined for it.

2. Each incoming business-level message should cause its recipient to generate an Item-level administrative response message back to the sender of the incoming message, according to the following guidelines:

    - If the incoming message contains a technical error, the response to that message should be of the same message type as the incoming message, but with a Message Class of "TechnicalError".
        - For example, the response to an incoming "AD-O" ad order message that contains a technical error should be another "AD-O" message with a Message Class "TechnicalError".
    - Otherwise, the response should be of the same message type as the incoming message, but with a Message Class of "MessageReceivedAcknowledgement".

    (Note that an administrative response **MUST** be sent if the *administrativeResponseRequired* attribute in an incoming business-level message has a value of "true".)

Copyright © 2010 AdsML Consortium. All rights reserved.

3. With the exception of broadcast or response messages (see "Categories of message patterns", below), each incoming business-significant message should cause its recipient eventually to generate a business-significant response message back to the sender of the incoming message, according to the following guidelines:

    - If the *Advertising Component Interactions Analysis* defines one or more possible business-significant responses to the incoming message, then the response should be one (or in a few cases two) of the business-significant message Types shown in the ACIA document as a possible response. For example:
        - The only business-significant response to an incoming "AD-O" (ad order) message is an "AD-OR" (ad order response) message.
        - The only business-significant response to an incoming "AD-OSE" (ad order status enquiry) message is an "AD-OS" (ad order status) message.

    - The incoming business message is identified using the "inResponseToMessage" message attributes of the business-significant response:
        - The unique message identifier for the incoming message may be recorded as the value of the `inResponseToMessageID` attribute.
        - The unique message code (e.g. "AD-O") for the incoming message may be recorded as the value of the `inResponseToMessageCode` attribute.

      Note: a business-significant response is either a 'Response' or a 'Status' message. The "inResponseToMessage" attributes are required in a 'Response' message. In a 'Status' message the attributes are optional; a 'Status' message can be broadcast and is not always requested, therefore.

    - If no business significant response has been defined for the incoming message, then the response should be an administrative response. This is accomplished by returning a message of the same Type as the incoming message, but with a Message Class of "MessageReceivedAcknowledgement."
        - For example, since no business-significant response is defined for it, the normal response to an "AD-OR" (ad order response) message is an Item-level response of type "AD-OR" with message class of "MessageReceivedAcknowledgement".

4. An incoming administrative response message must not be responded to with another AdsML message.
    - If an administrative response message itself contains a technical error, this should be reported to its sender by non-AdsML means (i.e. phone or email).

5. If an incoming business-significant message contains an error but it is not possible to respond with an error message of the same Type (perhaps because the Type of the incoming message has been omitted or become corrupted), the recipient should return a "ZZ-ERROR" message instead.

Copyright © 2010 AdsML Consortium. All rights reserved.

- For example, the response to an incoming message that could not be processed should be an administrative response with Message Code "ZZ-ERR" and Message Class "TechnicalError".

6. A message recipient **MUST NOT** continue the normal message exchange pattern once a message containing a technical error has been received.

4.2.1.2 Datagram pattern guidelines

The following guidelines for the exchange of Item-level messages following the datagram model are strongly **RECOMMENDED**:

1. Each message exchange begins with the transmission of a business-significant message from one partner to another. The nature of the recipient's response depends on whether the incoming message contains technical errors.

2. Each incoming business-level message should cause its recipient to generate an administrative response message back to the sender of the incoming message, according to the following guidelines:

   - If the incoming message contains a technical error, the response to that message should be of the same message type as the incoming message, but with a Message Class of "TechnicalError". For example, the response to an incoming "AD-O" ad order message that contains a technical error should be another "AD-O" message with a Message Class "TechnicalError".
   - Otherwise, the response should be of the same message type as the incoming message, but with a Message Class of "MessageReceivedAcknowledgement".

   (Note that an administrative response **MUST** be sent if the *administrativeResponseRequired* attribute in an incoming business-level message has a value of "true".)

This ends the datagram message exchange.

4.2.2 Categories of message exchange patterns

Message exchange patterns typically fall into four categories, each of which is described below. The category within which a given message is used will affect the normal response to that message. Instances of these meta-patterns can be found throughout the AdsML Framework.

4.2.2.1 Transaction

In a business transaction, one party initiates an exchange of information about a transaction, and the other responds. This includes requesting a transaction, changing or canceling a transaction, or enquiring about the status of a pending transaction. Transactions include both ordering transactions (reservations, bookings, production orders) and payment transactions (an invoice followed by either a payment or a claim concerning the invoice).

At least two business-significant messages will normally be exchanged: an initial message conveying the details of the requested action, and a business-significant response confirming the action.

Copyright © 2010 AdsML Consortium. All rights reserved.

In this category the request-response mode is strongly preferred. The datagram model is only used when necessary to integrate legacy systems that are not capable of supporting the request-response model.

4.2.2.2 Delivery of material

One party delivers a set of business-significant material to the other. Usually this material was (or will be) referenced by a transaction. For example, ad material is business-significant because it will be referenced by a production order, reservation or booking.

A message delivering materials *should* trigger a business-significant "response" message, but in many systems the datagram model is used for deliveries.

4.2.2.3 Broadcast information

One party "broadcasts" information to one or more recipients that is not referenced by a transaction and for which no response is necessary. For example, routine publication of an updated Mediapack, or monthly generation of account statements.

Broadcast messages always follow the datagram model: only an administrative response is expected.

4.2.2.4 Status reporting

One party provides information to one or more recipients about the status of a pending transaction or delivery. Sometimes this information is provided as a response to a request for status information; in other cases it is provided spontaneously based on internal business rules.

Status reporting can follow either the request-response or datagram model, depending on whether the status information was requested or provided spontaneously.

4.2.3 Support for the message exchange patterns

The AdsML Framework includes three components that can be used to define and support the recommended Item-level exchange patterns:

1. The names and descriptions of the messages described in the body of the *Advertising Components Interactions Analysis* and repeated in the table in the Appendix of that document. With one exception ("ZZ-ERROR"), the defined AdsML message types consist entirely of business-significant messages.
2. The "Possible business-level responses" that are indicated for each message in the table in the Appendix of the *Advertising Components Interactions Analysis*. This column of information tells you what the normal business-significant response to a given message is expected to be. If no business-significant response is indicated for a message, then the normal response to that message is expected to be an administrative response.
3. The "message class" flag in the Item Header section of the AdsML Envelope. This flag permits an AdsML Item-level message to be identified as containing one of the three classes of information described above:
    - Business-significant ("BusinessTransaction")
    - Acknowledgment ("MessageReceivedAcknowledgement")
    - Error ("TechnicalError")

Copyright © 2010 AdsML Consortium. All rights reserved.

4.2.4 Integrating "manual" messages into the AdsML choreography

The AdsML choreography assumes that all of the defined business messages are transmitted as AdsML messages. Therefore, the standards include rules about many of the specific data values that should be copied, for example, from a "request" message to a "response" message.

However, it is understood that given the state of existing systems, in most environments AdsML will initially be used for just some of the defined messages, leaving the rest of the business interactions to be accomplished by "manual" processes such as phone, fax or email. For example, a buyer of ad space might phone in the details of a new booking, expecting the seller of ad space to type the information into their system and then send an AdsML message as a confirmation. Or, AdsML messages might be used to convey the details of a new booking, but then the buyer might telephone with requested changes.

In general, it is possible to accommodate scenarios like these in which AdsML messages must be integrated with other forms of information transmission. The most common approach is to use the datagram model, which generally supports using AdsML messaging for the initial (request or broadcast) message, and non-AdsML communications (e.g. phone, fax or email) for any response to that message.

It is also possible to use AdsML messaging in other patterns, for example, sending an AdsML message in response to a non-AdsML request. However, there are a number of constraints in the AdsML message designs which limit your options when trying to support this "backwards" approach. These include:

- An AdsML business-significant "response" message **MUST** reference the transmission ID of the message to which it is a response.
- AdsML "response", "change" and "status" messages **MUST** include a globally unique, persistent ID (using the AdsML "QID" format) for the transaction or information in question. Therefore, these message types can only be used to reply to a manual message if it is acceptable for the party sending the message to generate this unique ID, or if the incoming information already included a suitable unique ID.
- A "change" message will normally generate a "response" message in return.
- A "status" message does not trigger a business response.

As a result of the above constraints, the following options are recommended when responding to a non-AdsML request for a new transaction, or a change or cancellation to an existing transaction.

- New Transaction
    - Send a Status message confirming the transaction and providing the seller's ID for that transaction to the buyer.
- Change or cancellation
    - Send either a "status" message confirming the changes to this transaction that have been made in the seller's system (no response expected), or a seller-initiated "change" message (response required).

Copyright © 2010 AdsML Consortium. All rights reserved.

4.3 Message contents

4.3.1 Change messages

Each AdsML change message (for example, "change a reservation", "change a booking", "change a delivery order", etc.) consists of a copy of the entire transaction that is being changed, as it will exist *after the change has been confirmed*[5], along with optional information to describe the nature of the change and indicate the sections of the transaction that have been changed. This design makes it easy for the message sender to generate the message (it is essentially a copy of the current state of this transaction in their system). The recipient of the message can either perform a parse-and-compare operation in order to identify just the changed sections and update their system accordingly, or can push the new information set into their system "as is" without choosing to parse it in such detail.

We have chosen this approach in favor of the most commonly used alternative model, which would have been to transmit a much smaller change message containing just the changed information. Our primary reasons are:

- Earlier e-commerce message formats were designed at a time when message transmission was expensive, so a premium was placed on reducing the number of bytes in each message, for example by sending only the changed data. Transmission costs have dropped to the point that within the context of a textual business message such as an AdsML message there is no significant cost difference between sending a larger vs. smaller message.
- The nature of the information being transmitted has become much more complex and interrelated, not only within the messages, but more importantly, as reflected in the database structures maintained by the communications partners. A given AdsML message is likely to contain just a fraction of the information about this transaction that is stored in each party's database.
- It is safer and less expensive for the initiating party to re-send the entire transaction than to parse out just the "changed" sections. Further, the sender cannot reliably know which aspects of a changed transaction might be significant to the message recipient, or how the components of that transaction are stored in the recipient's system.
- It is safer and more reliable for the recipient to receive a copy of the entire message, and perform any necessary analysis on their side, than to receive just those sections of the message that the sender has assumed will be significant.
- Receiving a full copy of the entire order also allows systems designers the option of easily displaying a complete new order for manual comparison with an existing order – rather than designing complex software to do so.

4.3.1.1 Identifying what has changed

Each AdsML change request message (for example, an Ad Reservation Change or Ad Order Change) contains an optional, repeatable `adsml:ChangeSpecification` element in its header. This structure can be used both to indicate the nature of the change(s) that have been made, and to identify the section(s) of the transaction in which they occurred.

[5] Note, in particular, that any parts of the transaction which have been flagged as deleted in the sender's system are also omitted from the change message.

Copyright © 2010 AdsML Consortium. All rights reserved.

The change specification structure is both generic and extremely flexible, in that both of its main components (a `CodeValue` and a `ChangeLocatorReference`) are optional and repeatable. In order to encourage consistency and interoperability between implementations, the following guidelines are **RECOMMENDED**:

- For each section of the transaction that has been changed, the change message header should include an `adsml:ChangeSpecification` element which indicates the nature of that change (via a code value) and identifies the section in which the change occurred (via a `ChangeLocatorReference`).
- Each instance of `adsml:ChangeSpecification` should include a code which indicates the nature of the change being described by that change specification. The code should take the form "verb.object", where the verb is either "add", "delete" or "edit" and the object is either the name of the structure in which the addition, edit or deletion took place, or a word which indicates the type of information which was added, edited or deleted. For example: "add.placement", "delete.rendering", "edit.schedule", "add.price", etc. Use of the AdsML Change Code controlled vocabulary is recommended for this purpose.
- Each instance of `adsml:ChangeSpecification` should include a `ChangeLocatorReference` which contains a QID value that matches (and therefore serves as a pointer to) the identifier of the lowest-level element in the message that contains the portions of the message in which the change occurred. For example:
    - If the change occurred in a placement in an ad order, then the `ChangeLocatorReference` should contain a copy of that placement's `PlacementIdentifier`.
    - If the change affected only a single insertion period within a placement, then the `ChangeLocatorReference` should contain a copy of the `ScheduleEntryIdentifier` of that `InsertionPeriod`, because `InsertionPeriod` is a lower-level containing element than its parent `Placement`.
    - If the change affected a placement's price, then the `ChangeLocatorReference` should identify the appropriate `PlacementIdentifier`. However, if the change affected the price of an entire booking, then the `ChangeLocatorReference` should point to the `BookingIdentifier`.
- If a change consists of a section of the transaction that has been deleted, then the change code should describe the nature of the change (e.g. "delete.placement" or "delete.schedule") and the `ChangeLocatorReference` should identify the structure in which that deletion occurred, or that was itself deleted from the transaction. For example, if an entire placement has been deleted from an ad order, then the `ChangeLocatorReference` should contain the `PlacementIdentifier` of the deleted placement, *even though that placement no longer exists in the change message*.
- The contents of any given `adsml:ChangeSpecification` element should describe a single change or a related set of changes that are all in the same section of the message. For example:
    - If two different placements in an Ad Order have been changed, then the change message should include two

Copyright © 2010 AdsML Consortium. All rights reserved.

`adsml:ChangeSpecification` elements, each of which describes one of the changed placements.

- If a single date has been changed or cancelled, then the change message should include one `adsml:ChangeSpecification` element with an appropriate code and a pointer to the `ScheduleEntryIdentifier` of the changed `InsertionPeriod`..
- If a series of schedule changes have been made in a placement, then the change message should include one `adsml:ChangeSpecification` element which includes a single code with a value of "edit.schedule" and a `ChangeLocatorReference` that points to the identifier of the placement in which these schedule changes occurred.

4.3.2 Business Response vs. Status messages

4.3.2.1 Response messages

The AdsML Framework includes many business-significant messages which have the word "Response" in their names: Ad Order Response, Ad Materials Response, etc. Response messages are used to provide the business-significant response to a particular incoming message. For each incoming message (such as Ad Material), one and only one business-significant Response message (such as Ad Material Response) **SHOULD** be sent. The Response message should be sent as soon as reasonably possible, even if further processing remains to be done before the recipient's final position will be known. The Response message **SHOULD** always be sent to the party that originated the message exchange (in this case, the sender of the Ad Material), and only to that party. A response message **MUST** identify the message to which it is a response in the *`adsml:inResponseToMessageID`* attribute. A response message **MAY** identify the message code of the message to which it is a response in the *`adsml:inResponseToMessageCode`* attribute.

4.3.2.2 Status messages

The AdsML Framework also includes many messages which provide status information about a previously initiated transaction or delivery. These have the word "Status" in their names: Ad Order Status, Ad Materials Status, etc. Status messages can be used either as the reply to an incoming Status Enquiry message, or as a broadcast message sent to one or more parties that did not explicitly request the information. Depending on the use, the metadata in the header of the message is filled in slightly differently.

When a Status is issued as a reply to a Status Enquiry, then only one instance of the Status message **SHOULD** be sent, and its header **MUST** identify the Status Enquiry to which it is a reply in the *`adsml:inResponseToMessageID`* attribute. It **MAY** also record the message code of the Status Enquiry in the *`adsml:inResponseToMessageCode`* attribute. This is analogous to the use of Response messages as described above.

When a Status message is sent spontaneously in broadcast fashion, however, it is typically used for one or both of these purposes:

- provide further information to the initiator of a transaction or delivery after having previously sent them a Response message
- provide information about a transaction or delivery to a party that was not a participant in the original message exchange.

Copyright © 2010 AdsML Consortium. All rights reserved.

When a Status message is used to provide subsequent information to the transaction initiator, it is treated as a sort of "follow-up Response" – that is, it is sent by the same party which originally sent the Response message, to the other party in that transaction. When a Status message is sent to other parties, it serves as a form of "carbon copy" message, similar to the cc list in an email distribution. In both of these uses, the Status message is treated like a datagram and **MUST NOT** include a value in its *adsml:inResponseToMessageID* attribute.

4.3.2.3 Usage examples

Status and Response messages belonging to the same group (for example, Ad Order Response and Ad Order Status) typically have almost identical structures and can convey the same business information. It is possible to send a Response message to one trading partner in reply to an incoming message from them, and simultaneously to send a Status message containing essentially the same information to other trading partners who are likely to be interested in that information (or who are obligated to receive it by virtue of a trading partner agreement).

For example, consider this scenario:

- A delivery agency transmits a set of ad materials to a publisher in the form of an AM-M (Ad Materials) message.
- Immediately the publisher discovers a problem with the way the materials were prepared. The publisher sends an AM-MR (Materials Response) message to the deliverer containing the publisher's formal response, as required by the AdsML choreography in request-response mode. The AM-MR message includes a description of the problem.
- Simultaneously, the publisher sends an AM-MS (Ad Material Status) message containing the same information to the producer of the ad materials, so as to ensure that the producer learns of the problem as quickly as possible, without having to wait for the deliverer to inform them.

In theory, the publisher might have sent a duplicate copy of the AM-MR message to the producer, but that would have violated the AdsML Framework rules, because an AM-MR must only be sent in response to an AM-M message. Since the publisher had not received the AM-M from the producer, the publisher therefore must send the producer an AM-MS message instead.

4.3.2.4 Contents of status and response messages

In general, the structure of an AdsML status or response message closely mirrors the structure of the main message in its message group, with the addition of status codes at the top of the message and next to significant structures within it. It is therefore possible to include in a status or response message all of the information that was in the earlier message to which it refers. However, including so much information in a response or status message is not always possible or desirable. Each individual AdsML e-commerce standard provides its own guidelines as to how much information should be included in a status or response message conforming to that standard.

Status information is conveyed in the optional `adsml:Status` element, which contains a `Status` code and an optional, repeatable `StatusQualifier` code that can provide more information about the status.

The `adsml:Status` structure is available at the top level of each appropriate AdsML transaction message (thus, providing status information about the transaction as a whole), and also in each significant lower-level structure inside the message (thus, providing status information about the information or sub-

Copyright © 2010 AdsML Consortium. All rights reserved.

transactions described at those lower levels.) For example, a Materials Delivery Response message (AM-MR) contains status information both at the top level, and also in the Rendering and Delivery structures. This allows for a status report which conveys both a "high level" status of the material delivery as a whole, and also "drill-down" statuses for lower level components such as each rendering and each individual delivery transaction.

Because the AdsML status mechanism is so flexible, it is important that trading partners who are setting up AdsML communications agree on how they will convey status information: which types of codes and other information will be provided, at which levels of each message, and under what circumstances.

The following guidelines are **RECOMMENDED**:

- Trading partners should populate the adsml:Status code with values from the AdsML Status Code controlled vocabulary. This vocabulary is designed to provide generic, context-independent status code values that can be used at any level in any type of AdsML message. (For example, "Received", "Accepted", "Rejected", and "BeingProcessed".) In this way, it can provide a degree of interoperability across the entire AdsML message set, whether one is describing a booking, placement, delivery, rendering, etc.
- Trading partners wishing to provide more detailed, context-specific status information should use the `StatusQualifier` code for that purpose. For example, this would be the place to convey the reason for a status code of "Rejected", or the processing stage that the item or transaction in question has reached.
- If any status information appears in a message, then the highest-level status code in that message should be populated. In other words, users should not convey status information at lower levels of a message while leaving the top-level status code blank.
- The value of the highest level status code in a transaction should convey the overall status of that transaction, i.e. the sum of its parts. In cases when the lower level components of the transaction have mixed statuses, it is up to each trading partner to decide how to summarize them into a single top-level status. However, the guiding principle is that the top-level status should reflect the "lowest" or "worst" status of any sub-transaction in the message. For example, the response to a materials delivery in which one rendering has been approved but another is still being processed should indicate that the delivery is "BeingProcessed".

See the individual AdsML e-commerce standards for more information about how the status mechanism is intended to be used in that particular standard.

4.3.2.5 Importance of status messages

AdsML does not impose a requirement that the recipient of a Status message perform any particular action as a result of receiving that message, including reading it. Trading partners wishing to use status messages for more important purposes should agree in advance on the situations in which they will send Status messages and the behavior that the receipt of a status message is expected to trigger.

4.3.2.6 Date of status and status enquiry messages

It is not possible to request status information as of a particular date. A status request is always for the "current" status.

Copyright © 2010 AdsML Consortium. All rights reserved.

There may be a delay before the recipient of a Status Enquiry is able to respond to it. In all cases, the status information reported in a Status message should reflect that message's business significant date, which is not necessarily the time at which the status request was made.

4.3.3 Multiple business objects in one message

Each of the Item-level standards allows for the transport of multiple business objects within the same AdsML message, provided that they are all of the same message type (as defined in the Advertisement Component Interactions document). For example, an AdsMLBookings instance can include one or more business messages, or "bookings", of the same type, i.e. a set of orders or a set of cancellations etc.

It is not permitted to mix different types of business objects in the same message, for example a "new order" and a "change" in the same message.

4.3.4 "Informational" structures

Many of the Item-level standards contain structures whose names end with the word "Information" and begin with the name of a type of information that originated in a different AdsML specification, for example, `BookingInformation` and `ProofOfPublicationInformation` in AdsMLFinancials. These informational structures are designed to convey information from an earlier point in the advertising workflow which is useful, but not critical, to the current transaction. For example, the ability to convey `BookingInformation` in an Invoice or Materials Delivery message may help the recipient of the message match the incoming Invoice or Ad Materials with its source booking. This can be very helpful, but the Invoice or Materials would still have been delivered even if the `BookingInformation` had been omitted from the message.

Each informational structure consists of an assemblage of optional elements that are reused from their source schema. The general philosophy when designing AdsML's informational structures has been to include every element which might possibly be useful on an informational basis at a later stage of the workflow. Therefore, the informational structures are likely to contain many more elements than are required for any given context.

Further, by their very nature, the contents of an informational structure come from upstream in the workflow, often from another system than the one which generated the current message. This information may be incomplete or out of date. As a general rule, informational structures **SHOULD NOT** be relied upon to contain actionable information in the current context. Instead, they **SHOULD** be used in such a way that the current transaction will not break if there are gaps or errors in the informational content, or if the entire informational structure is omitted by the message sender.

4.4 Multilingual content

To support internationalisation, AdsML messages permit human-readable textual content to be repeated and provided in as many different alternative languages as necessary to support multilingual business environments.

Multilingual support is technically provided by the specification of internationalization attributes (*`i18nAttributes`* group) on many of the elements containing human-readable text[6]. The group contains optional *`xml:lang`*, *`dir`* and

[6] Elements containing human-readable text that is commonly used to identify the object in question have not been internationalized.

Copyright © 2010 AdsML Consortium. All rights reserved.

source attributes and appears in many element contexts where human readable content is contained. For example: a description, disclaimer text or a property label. The *xml:lang* and *dir* attributes identify the human language used and the direction in which it should be read, e.g. '`ltr`' (i.e. 'left to right'). While the 'lang' and 'dir' attributes are typical i18n attributes, 'source' serves a subtly different purpose. In the case where multiple human language variants of a text are available, the *source* attribute is used to identify the text in the original or source language. For example, if a description was originally produced in Finnish, and Swedish and English translations are also provided for it, then the Finnish description could be marked as 'source':

```
<adsml:RateCode>
 <adsml:CodeValue>P3</adsml:CodeValue>
 <adsml:Description xml:lang="fi" adsml:source="true">3.
kansi</adsml:Description>
 <adsml:Description xml:lang="sv">3 pärm</adsml:Description>
 <adsml:Description xml:lang="en">Cover 3</adsml:Description>
</adsml:RateCode>
```

The i18n attributes are optional. The attributes are specified on structures in the AdsML Type Library and in individual standards in the AdsML Framework. Due to the re-use of AdsML Type Library structures across all schema files there may be contexts where i18n support has been provided even though it is not appropriate or necessary to use that i18n support. Examples of this include:

- AdsMLMediaPack standard: the human language used within a `Publication`'s `DescriptiveInformation` would be specified at the `DescriptiveInformation` level, rather than on individual elements contained inside it, even though the child `adsml:Description` element also contains i18n support.
- AdsMLStructuredDescriptions standard: the human language used in an Object Definition can be recorded using the `ObjectDefinition/@xml:lang` attribute. To specify the language of an `adsml:Description` within that `ObjectDefinition` would be superfluous.

Use of i18n attributes should be guided by an implementer's business case and by the specification and usage rules documentation for the AdsML standard used.

4.4.1 Rules for recording and handling multilingual content

- In general, the different human language versions of an internationalized element **SHOULD** be semantically identical. I.e. if a code value description is provided in more than one language, then the fundamental meaning of that description **SHOULD** not change from one language version of it to the next.
    - Note that in some message contexts, such as notes or contact information, internationalized elements are repeatable because of their nature and not merely because of their i18n attributes. In these cases repeated data items may have different semantics. For example, the fact that two sibling `Contacts` are included in a message, one in English and one in Swedish, does not necessarily indicate that they are the same person.
- Multilingual versions of simple textual content: If textual content is provided in more than one language, then the element containing that text will be repeated. In this case at least the `xml:lang` attribute **MUST** be specified, so as

Copyright © 2010 AdsML Consortium. All rights reserved.

to identify the language used and allow different language versions of the text to be easily differentiated from one another.

- Multilingual versions of requirements: The `RequirementSpecType`'s content model allows the specification of requirements using repeatable Code or Text elements. If there are textual instructions in multiple languages, then all of the texts in a given language **SHOULD** comprise a set which should be ANDed together and treated as a whole.
- Multilingual versions of a set of notes: The `Notes` content model uses multiple `NoteLine` elements to record a set of human-readable notes. If note line(s) are provided in multiple languages, then all of the notes in a given language **SHOULD** comprise a set which should be treated as a whole.
- Conveyance of multilingual document renderings: Different language versions of the document **MAY** be provided using the repeatable `DocumentRendering` element. If multiple document renderings are provided, they **MUST** be semantically equivalent to each other.

Note: it is up to the user to define how to filter and process multilingual content in an AdsML message.

4.5 Relationship of AdsML messages to business and technical operations

The types of messages (and associated message codes) used in the AdsML Framework are designed to support the AdsML message-level choreography. For example, the AdsML message choreography allows a message to be characterized as a "transaction request" or "transaction response", a "change" or a "cancellation." The meaning of these messages is well defined within the AdsML Framework.

It is important to realize, however, that the AdsML messages operate at a different level than either the business operations that create them, or the database operations that they are likely to trigger. For example, consider a scenario in which an advertisement buyer wishes to cancel one of the insertion dates on a previously placed order. In terms of business operations, the buyer is likely to think of this as a "cancellation", and his software system may use that word. However, the AdsML Ad Bookings standard specifies that a change to just part of an order must be conveyed in a "change" message, not a "cancellation" message, and so the buyer's system will be expected to generate a "change" message. And when this message arrives in the receiving system, it may well trigger a "delete" operation in the publisher's database. Therefore, in this example a "cancellation" operation generates a "change" message that triggers a database "delete". (Conversely, when an entire booking is cancelled, AdsML does use a "cancellation" message as one might commonly expect.)

There is nothing wrong with this apparent mismatch of terms. Each of the affected domains – business operations, XML messaging, and database management – uses its own terminology for reasons that make sense within the context of that domain.

5 Administrative responses and error handling

Note: The information in this section applies only to the AdsML item-level standards.

Copyright © 2010 AdsML Consortium. All rights reserved.

5.1 When to send an administrative response

An administrative response message **MUST** be sent when:

1. Agreed in the TPA, and/or
2. Specified in the message header (*administrativeResponseRequired* attribute), and/or
3. During system testing (see also section on "System Testing below), and/or
4. Reporting a technical error in the incoming business-significant message.

5.1.1 Responses to responses

Administrative response messages **MUST NOT** be given in response to administrative responses. If an administrative response message itself contains an error, this should be reported to its sender by non-AdsML means (i.e. phone or email).

However, as one would expect, a technical error or acknowledgment should be given to a business level response. For instance, after receiving an ad order response (AD-OR), the receiver **SHOULD** issue an administrative acknowledgment as a message receipt. If the ad order response caused a validation error, an administrative error message **MUST** be issued.

5.2 Message types

The message type for administrative response messages **MUST** be the same as the type of the message the response is about. Administrative messages **MUST** however have the *messageClass* attribute set to either 'MessageReceivedAcknowledgment' or '*TechnicalError*' in order to distinguish from business messages with the same type.

However, in case of "catastrophic" errors where it is not possible to extract the message type of the received message, the 'ZZ-ERROR' message type **MUST** be used as defined in the *Advertising Components Interaction Analysis*.

5.3 Technical errors

Technical error messages are triggered by flaws in an incoming business-significant message that make that message unsuitable for business-level processing. These flaws are technical rather than business-level errors. They usually occur either because the message became corrupted during transit, or more commonly, during the period when two parties are integrating their systems and there are still some bugs in one or both of their systems.

Errors of a technical nature **MUST** always be reported if possible. If the sender of the message containing the error is known, an error message must be returned using the message groups as defined in the AdsML Framework. In this case the AdministrativeResponse element must be used, referencing the error message class.

In case of "catastrophic" errors where it is not possible to extract the message type of the received message, the 'ZZ' message code **SHOULD** be used.

In most cases, the receipt of a technical error message will trigger manual intervention by the recipient's IT staff to determine the cause of the error and put the underlying business transaction back on track.

Copyright © 2010 AdsML Consortium. All rights reserved.

The recipient of a technical error message **MUST** assume that the business-significant message which contained the error in question was not delivered to the application for which it was intended.

A "negative" response to a business message is not considered to be an error. For instance, a denied booking is usually due to business reasons such as an invalid insertion date, bad credit etc. Such responses are handled in the normal message flow where the reason for transaction denial or delivery refusal can be expressed at a business level.

5.4 Workflow implications

Administrative responses are typically sent and received by an AdsML Processor, a piece of software which, in many cases, sits outside of the business application (e.g. sales system or production system) that generates and consumes the business-significant information. This means that the sending or receipt of an administrative response message is typically done without the knowledge of the systems or people who are handling the business-significant information.

Administrative responses can have an indirect impact on workflow processes, because the absence of an anticipated response can trigger manual processes to follow up on the situation and ensure that the booking has been properly stored in both parties' systems. However, under normal circumstances the receipt of an administrative response message will not trigger any workflow processes.

6 Establishing communications

6.1 Message transmission mechanism

The AdsML Framework neither defines nor limits the mechanism by which AdsML messages are transmitted between trading partners. Use of the AdsML Envelope and its associated addressing and processing model, while encouraged, is optional, and the Item-Level standards can be used both inside and outside of the AdsML Envelope. Further, regardless of whether the AdsML Envelope is used, the XML messages defined by AdsML are designed to be carried inside of whatever transmission envelope or mechanism is required by the communications infrastructure that the trading partners have agreed to use. Options in this area include, but are not limited to:

- Placing the AdsML messages inside of SOAP envelopes and conveying them via web services calls
- Converting the AdsML messages into text documents and conveying them as email attachments
- Converting the AdsML messages into text documents and transmitting them via FTP
- Using standard Internet protocols such as HTTPS to send AdsML messages directly between dedicated communications ports on the sending and receiving systems
- Sending AdsML messages via a communications infrastructure that has been defined as part of a regional or industry-wide initiative, such as the ebiz-for-media project in North America
- Sending AdsML messages via a commercial third-party communications provider that has been engaged by the trading parties, as is the case in the Ad\Venture project in the Netherlands

Copyright © 2010 AdsML Consortium. All rights reserved.

In all cases, selecting the appropriate communications infrastructure is a critical early step in the process of defining and configuring the e-commerce project of which AdsML is a part. However, this choice should have little or no effect on the AdsML choreography and capabilities that the trading partners elect to implement.

6.1.1 Conveyance of binary materials

6.1.1.1 In-line content

The AdsML standards support the optional ability to convey binary ad materials, such as PDF files, in-line as encoded content in the character stream of the AdsML message. This capability is primarily intended for the conveyance of textual content such as lineage ads, or for very small binary files such as thumbnails, but it can theoretically be used for any size of in-line file. Processing in-line binary content requires an extra step during the packing and unpacking of the messages. Therefore, trading partners wishing to use in-line content transmission must explicitly agree on this when setting up their systems, and must ensure that their AdsML message handling software can support in-line packaging in the AdsML workflow.

6.1.1.2 MIME packaging

The AdsML standards support the optional use of MIME multipart/related packaging [MIME 1998] to convey binary ad materials, such as PDF files, in the same MIME package as the AdsML message that is logically "delivering" those materials. Processing such MIME packages requires an extra step during the packing and unpacking of the messages. Therefore, trading partners wishing to use MIME multipart/related packaging must explicitly agree on this when setting up their systems, and must ensure that their AdsML message handling software can support MIME packaging in the AdsML workflow.

6.1.1.3 Retrieval from a specified location

The AdsML standards support the optional ability to convey binary ad materials, such as PDF files, by specifying an external location (e.g. a URL) from which the recipient of the AdsML message is expected to retrieve the content. Retrieving and processing such content requires extra processing. Therefore, trading partners wishing to transmit materials by external reference must explicitly agree on this when setting up their systems, and must ensure that their AdsML message handling software can support the necessary processing.

6.2 Testing

Note: The information in this section applies only to the AdsML item-level standards.

All AdsML item-level standards support the sending of test messages. Test messages are used to set up and test digital communications, but must not be acted upon by either party. Two modes of testing are supported: testing of the transmission infrastructure, and testing of business communications between the parties' software systems.

In order to allow transmission of test messages, the *transmissionStatus* attribute on the root element of any AdsML item-level message can be set to 'TransmissionTest' or 'BusinessMessageTest'.

Copyright © 2010 AdsML Consortium. All rights reserved.

6.2.1 Testing the transmission channel

When receiving a 'TransmissionTest' message, an administrative response **MUST** be given. A business level response **MUST NOT** be given, and the message **SHOULD** be discarded and not further processed. The idea is to test that a communications link is working, but not to feed any information into the recipient party's business systems.

Note that test transmissions **MUST** include only test messages, i.e. no real messages can be included in a test transmission.

6.2.2 Testing communications between business systems

Messages with status 'BusinessMessageTest' are intended to test business messages such as reservations, orders and content delivery. Such messages **MUST** get appropriate responses where the *transmissionStatus* attribute **MUST** be 'BusinessMessageTest'. This scenario goes further than a transmission test, because it feeds the contents of the incoming message into the recipient's business system. That recipient system is then required to send a business-level test response back to the originating party.

This scenario only works when the business systems used by both parties are able to send and receive test information. The recipient of a test business message **MUST** issue an appropriate business-level response message, where the *transmissionStatus* attribute of the response **MUST** be 'BusinessMessageTest'.

7 Achieving interoperability

The AdsML standards are designed to be used in many different workflows and environments. Therefore, they contain many optional elements and relatively few required ones. Similarly, the AdsML choreography defines many request-response message pairs that can be grouped together to implement business processes, but does not prescribe exactly which of these message pairs should be used in any given e-commerce environment. And finally, AdsML supports a process by which trading partners can control the values that may be placed in many of the data elements, but with few exceptions, does not mandate which specific controlled vocabulary values should be available for use.

As noted in this document's introduction, in order to implement AdsML-based e-commerce, trading partners and their vendors (or industry associations acting on their behalf) are expected to review the AdsML Framework and decide:

- Which AdsML standards will they implement within their particular region or business activity?
- Which business transactions will they support?
- Which types of information will they include in their messages?
- Which information will be conveyed in machine-processable elements, vs. which will be sent as unstructured text that requires human handling?
- For which machine-processable elements will they require use of a particular controlled vocabulary?

This much flexibility, while critically important, can also lead to implementation problems and a lack of interoperability. The AdsML Framework provides three important tools to help users manage these requirements: Configuration checklists, Controlled Vocabularies, and Profiles. These can then be used as building blocks when assembling a formal Trading Partner Agreement.

Copyright © 2010 AdsML Consortium. All rights reserved.

7.1 Configuration checklists

7.1.1 Checklists vs. conformance levels

In order to facilitate implementation and interoperability, pre-defined packages of features and functionality are a valuable tool. A systems vendor or trading partner can use these predefined packages to declare the overall style by which an AdsML standard (such as AdsMLBookings) is supported in their particular system or location. This is a better starting point when, for instance, expressing system requirements for a new bookings system or negotiating a Trading Partner Agreement with a new trading partner, than having to go through the complete set of possible features in the standard, one by one.

When such feature packages are organized hierarchically, as for instance in the Conformance section of the *AdsML Envelope Processing Model* documentation, they are often called *conformance levels*:

- Level 0: Base level, basic functionality
- Level 1: Normal level, most support is expected here
- Level 2: Extended "super-user" functionality and flexibility.

A levelled approach like this assumes that each higher level adds to the capabilities of lower levels. This concept, used also in other IT areas, is comprehensible and easy to communicate to users.

In the AdsML item-level standards, however, it is difficult to define a clear hierarchy. We have chosen instead to define for each standard a set of packages that are more orthogonally independent feature sets, from which implementers can pick and choose according to their business requirements. Such an approach is less helpful for achieving interoperability than a levelled hierarchy, since different systems are more likely to support different combinations of these features, and more issues will need to be resolved during the TPA set-up phase. But the TPA set-up process will nevertheless be smoother with a checklist of modules as a starting point, compared to a situation where no such modules have been defined at all.

A non-hierarchical approach also has the advantage of not "forcing" a developer to implement features that are irrelevant to the business problem at hand, only because some of the features on a particular level were required.

An approach using packages of independent modules cannot be called conformance levels since the name itself suggests a hierarchy. Also, at the time of this writing the AdsML Technical Working Group is not prepared to issue recommendations or requirements for how a particular party can be proven to "conform" to an AdsML standard. We have chosen instead to refer to these packages of functionality as "Configuration checklists".

7.1.2 Use of configuration checklists

The Configuration Checklists provide a common approach and terminology for defining the most common choices that users must make when implementing each AdsML item-level standard. Each checklist consists of a set of "packages" of features. Some of them provide options which directly affect the technical capabilities of the sending and receiving systems (for example, the ability to send binary content in-line in a message). Others reflect important choreography choices that need to be agreed between trading partners when they establish AdsML communications.

Each package consists of either:

Copyright © 2010 AdsML Consortium. All rights reserved.

- a set of hierarchical levels from which one must be selected (represented by a numbered list), or
- a set of non-exclusive options from which any combination can be selected (represented by a bullet list), or
- a list of mutually-exclusive choices from which one must be selected (represented by a textual description).

AdsML does not define a storage format for the configuration choices that have been made. It is up to users and vendors to record their configuration selections in the most appropriate format to the task at hand. Depending on circumstances, an appropriate format could range from the back of a napkin, to a word processor document or spreadsheet, to an XML file that is read in automatically by an AdsML-enabled system.

It is **RECOMMENDED** that developers and trading partners use the AdsML Configuration Checklists when selecting functionality to build into their systems, and as a starting point when communicating with each other and with their vendors about their systems' capabilities.

It is **RECOMMENDED** that vendors use the AdsML Configuration Checklists when defining and describing the AdsML-related functionality that is supported by their systems.

It is **RECOMMENDED** that vendors provide a mechanism by which each customer's AdsML-enabled system can be configured in accordance with options in the relevant AdsML Configuration Checklist. For example, a customer who is configuring a system to exchange AdsML Bookings messages would be able to use the AdsMLBookings Configuration Checklist (or at least, those options from the AdsMLBookings checklist which their vendor has chosen to support) as a starting point for specifying how they would like to use their system when communicating with their AdsML trading partners.

See the documentation for each AdsML item-level standard for further information about the configuration checklist options that are provided in that standard.

7.2 Controlled vocabularies

Despite its relatively technical name, a "controlled vocabulary" is simply a mechanism to separate the lists of allowed values that can be included in a business message from the structural definition of that message. This provides significant management benefits in the AdsML environment, where each organization that implements AdsML will have to maintain lists of allowed values ("controlled vocabularies") that will change at their own pace, independently of the pace at which the AdsML Schema is updated over time. By using a controlled vocabulary approach, the organization will be able to manage its master lists of controlled values without having to worry about how to integrate them with past or future releases of the AdsML Schema.

7.2.1 Validating controlled values

Control of allowed values in business messages can either be a part of the trading partners' application logic, or a part of a validation process using XML Schema based validators. The former relaxed approach does not require any special formal handling as long as the trading partners agree on the list of allowed values which are then recorded inside AdsML business messages as simple strings.

It is often almost always possible to include the name of the controlled vocabulary (i.e. a code list) from which a value is taken. This feature enables trading partners to dynamically name and use values from different controlled vocabularies in their business messages. For instance, consider a market where

Copyright © 2010 AdsML Consortium. All rights reserved.

two different but similar code lists for ad sizes are in use, one for magazines and one for newspapers. In a booking, both code lists use the value 'FullPage' but by also specifying the code list name, trading partners will be able to process the ad size specification correctly according to the originating controlled vocabulary's definition for that value.

But for a more formal approach, AdsML have chosen also to support validation of controlled vocabularies using some of the built-in XML Schema capabilities, because doing so makes it possible for implementers to use an off-the-shelf XML Schema validator to validate the controlled values in each AdsML message that they receive, rather than having to write custom code for the same purpose.

In this regard, the technical framework effectively allows each element in an AdsML message to be defined as belonging to one of several categories:

1. Not a code: it is not possible for regional groups to define a controlled list of values for this element that can be enforced by schema validation
2. A "fixed" AdsML code field: AdsML has defined a fixed set of values that cannot be changed by regional groups, and that will be schema validated
3. A "modifiable" AdsML code field: AdsML has defined a recommended set of values that can be schema validated, however the list can be removed or replaced by regional groups
4. Potentially a code field: AdsML has not defined a set of values, but regional groups can define their own lists of values which will then be enforced by either application logic or schema validation

For those elements in the latter two categories for which schema validation is required, AdsML provides mechanisms by which any group that is implementing an AdsML standard can define its controlled vocabulary extensions and replacements, which are packaged in schema extension files that are maintained by the regional group and versioned separately from the AdsML standard. For a detailed description of how to implement these schema extensions, see "Implementing controlled vocabularies with schema-based validation" later in this document.

7.2.2 The AdsML Controlled Vocabularies

The AdsML Framework includes a set of recommended controlled vocabularies. They are detailed in the schema whose namespace is "`adsml-cv`", and HTML documentation for them is included in the Framework release.

Users of the AdsML Controlled Vocabularies should note that each controlled vocabulary has a status associated with it, either "Proposed" or "Approved". A controlled vocabulary with a "Proposed" status should be treated as a work in progress. It could be a new controlled vocabulary which has not had sufficient review, has been recently heavily augmented, or is under reconsideration. It should be used with appropriate caution.

A controlled vocabulary with the "Approved" status should be treated as stable, in that the values in it are unlikely to change, although it is not necessarily comprehensive. It is strongly **RECOMMENDED** that if an AdsML CV contains a value with "Approved" status that closely matches the information you need to transmit you **SHOULD** use that value in your messages in order to promote interoperability across implementations of the AdsML Framework.

A controlled vocabulary can change status between releases if it is either (re)approved, which changes it to "Approved", or is modified, which changes it to "Proposed".

Copyright © 2010 AdsML Consortium. All rights reserved.

The AdsML CVs are constantly evolving. Most changes are additive, making them backwards compatible, but this is not always the case. For example, in Framework 3 the `LinkedPlacement` structure was significantly changed, such that where in Framework 2 there had been a single set of `PlacementLink` codes, in Framework 3 these were divided into three different code elements, each with different semantics. The matching CV was split and reworked to match the new semantics. As a result of these types of changes, each release of the AdsML CV set is only guaranteed to work in the Framework in which it was created, and may not be backwards compatible with prior releases of the Framework.

Each AdsML controlled vocabulary is assigned a unique Reference ID consisting of the name of that vocabulary plus the draft number of the CV package that contains it. (For example, "AdsMLAdTypeCV:1".) If you want to reference a set of AdsML CV values as they existed at a particular point in time from an external document, such as a TPA, use the Reference ID of the CV containing those values.

7.2.3 Examples

Here are examples showing the five most common ways to represent a controlled vocabulary value in an AdsML messages, in this case, a color code. The information content of these examples is identical – a color code of "ProcessColor" – but the metadata indicating how that value can be validated is different in each case. Because the same code structures are used throughout the AdsML Framework, the approaches shown here are available for any code in an AdsML message. Trading partners are free to select the approach that is most suited to the code at hand in their situation. Frequently, different approaches will be used for different types of codes, depending on how important it is to validate them and where the list of valid values can best be maintained.

7.2.3.1 Code value only

```
<Colors>
  <ColorType>
    <adsml:CodeValue>ProcessColor</adsml:CodeValue>
  </ColorType>
</Colors>
```

In this first example, the sender provides a color code value "`ProcessColor`" but does not indicate how to validate it. Either the trading partners have previously agreed on a set of valid color code values, or the recipient's application will not be able to validate the value automatically.

7.2.3.2 Explicit reference to an external code list

```
<Colors>
  <ColorType>
    <adsml:CodeList>OurColorCodes</adsml:CodeList>
    <adsml:CodeValue>ProcessColor</adsml:CodeValue>
  </ColorType>
</Colors>
```

In this example, the message explicitly indicates that "`ProcessColor`" can be validated against a code list named "`OurColorCodes`". Assuming that the recipient's application has access to a current copy of OurColorCodes, the application can programmatically determine whether "`ProcessColor`" is a valid entry.

Copyright © 2010 AdsML Consortium. All rights reserved.

7.2.3.3 Explicit reference to an AdsML code list

```
<Colors>
  <ColorType>
    <adsml:CodeList>AdsMLColorTypeCv</adsml:CodeList>
    <adsml:CodeValue>ProcessColor</adsml:CodeValue>
  </ColorType>
</Colors>
```

This example is identical to the prior one, except that the naming convention of the CodeList indicates that CodeValue comes from an AdsML recommended controlled vocabulary.

7.2.3.4 Schema-validatable reference to a user-provided controlled vocabulary

```
<Colors>
  <ColorType>
    <adsml:CodeValue
       xsi:type="usercodes:OurColorCodes">ProcessColor</adsml:CodeValue>
  </ColorType>
</Colors>
```

This example is similar to the previous one, in that the message asserts that "`ProcessColor`" can be validated against a controlled vocabulary named "`OurColorCodes`". But in this case, use of the *`xsi:type`* attribute asserts that the OurColorCodes value set has been recorded in a schema with the namespace "`usercodes`". Therefore, if an XML schema processor is used to validate the message, it will ensure that "`ProcessColor`" is a valid entry.
Note: The use of *`xsi:type`* in the AdsML Framework to drive schema validation is described in "Implementing controlled vocabularies with schema-based validation" later in this document.

7.2.3.5 Schema-validatable reference to an AdsML controlled vocabulary

```
<Colors>
  <ColorType>
    <adsml:CodeValue
       xsi:type="adsml-
cv:AdsMLColorTypeCv">ProcessColor</adsml:CodeValue>
  </ColorType>
</Colors>
```

This example is operationally identical to the prior one, except that in this case the xsi:type value "`adsml`-cv:AdsMLColorTypeCv" indicates that "`ProcessColor`" should be validated against an AdsML-provided controlled vocabulary named "`AdsMLColorTypeCV`". All of the AdsML controlled vocabularies can be found in the schema whose namespace is "`adsml`-cv".

7.3 User-defined properties

In order to encourage interoperability, users are not allowed to add their own elements to the AdsML schemas or messages. However, sometimes a regional group will need to exchange machine-processable values for which AdsML does not provide the necessary structures.

The solution is to create one or more user-defined properties using the `adsml:Properties` mechanism. This mechanism allows trading partners to define

Copyright © 2010 AdsML Consortium. All rights reserved.

the additional information they need to transmit, and convey it in such a way that it can be efficiently identified and processed.

AdsML provides optional, repeatable `adsml:Properties` elements in every message format, often at more than one location. For example, in AdsMLBookings there are `adsml:Properties` structures available in the business-message header, in each placement group, and in each placement.

Within `adsml:Properties` there are two ways to represent a user defined value: as a `Property`, or as a `LabeledProperty`. In terms of their information content, these structures are effectively identical. Each of them allows you to convey a name:value pair, where the name is the name of the property in question, and the value is the value of that property that is being conveyed in this particular message. The difference is that `Property` requires you to define the property name in an AdsML extension schema and reference it using an *`xsi:type`* attribute, while `LabeledProperty` allows you to convey the information more informally.

7.3.1 Usage rules

Regardless of the mechanism that is used, trading partners wishing to exchange user-defined properties **SHOULD** agree in advance on the syntax, labels, meanings and usage of the properties they wish to exchange.

Any user-defined properties contained in a received message instance that have not been agreed in advance are not bound to be understood and therefore **SHOULD** be ignored by the receiving application.

7.3.2 Syntax examples

Here are examples of two values being transmitted first as a pair of user-defined `LabeledProperty(s)`, and then as a pair of user defined `Property(s)`.

These examples are adapted from an actual AdsML implementation in which the messages were sent between two internal systems within the same business, and therefore contained technical information (a Change Flag and a Last Updated date/time) that would not normally be found in a business-to-business AdsML message.

In non-AdsML syntax, these values could have been represented as: ChangeFlag=”true” and LastUpdated=”2001-12-17T09:30:47”. The only way to convey this additional information while conforming to the AdsML standard, however, is to use one of the two properties mechanisms as shown below.

7.3.2.1 Using adsml:LabeledProperty

The `LabeledProperty` element is similar to the code structures that are used throughout the Framework and were described in the previous section. There are two internal elements: a mandatory `Value` containing the property value, and an optional `Label` which identifies its type.

```
<adsml:Properties>
  <adsml:LabeledProperty>
    <adsml:Value>true</adsml:Value>
    <adsml:Label>ChangeFlag</adsml:Label>
  </adsml:LabeledProperty>
  <adsml:LabeledProperty>
    <adsml:Value>2001-12-17T09:30:47</adsml:Value>
    <adsml:Label>LastUpdated</adsml:Label>
  </adsml:LabeledProperty>
</adsml:Properties>
```

Copyright © 2010 AdsML Consortium. All rights reserved.

It is strongly **RECOMMENDED** that when using `LabeledProperty`, the `Label` should be explicitly provided as shown above. This allows software to differentiate between multiple properties in the same message, and makes it easier for the recipient to process them.

It is **RECOMMENDED** that the Label provided for a user defined property should include the domain name of the organization that defined the property, a date on which that organization controlled the domain name, and the name of the property itself, for example: "MyOrg.com:2006-10-01:ChangeFlag". This is the same format that is used for AdsML "QID" elements. Use of a fully qualified property label in this way allows a receiver getting, e.g., LastUpdated properties from two different senders to recognise the difference automatically. In this case, the full message fragment would be:

```
<adsml:Properties>
  <adsml:LabeledProperty>
    <adsml:Value>true</adsml:Value>
    <adsml:Label>myorg.com:2006-10-01:ChangeFlag</adsml:Label>
  </adsml:LabeledProperty>
  <adsml:LabeledProperty>
    <adsml:Value>2001-12-17T09:30:47</adsml:Value>
    <adsml:Label> myorg.com:2006-10-01:LastUpdated</adsml:Label>
  </adsml:LabeledProperty>
</adsml:Properties>
```

If you are only transmitting one property, however, and wish to do so in as compact a manner as possible, you and your trading partners may agree to convey just its value. As can be seen in the example below, this is a risky approach unless you are certain that the message recipient will know how to process the un-labeled value:

```
<adsml:Properties>
  <adsml:LabeledProperty>
    <adsml:Value>true</adsml:Value>
  </adsml:LabeledProperty>
</adsml:Properties>
```

7.3.2.2 Using adsml:Property

The `Property` element uses an *`xsi:type`* attribute to label each user-defined property. Here are the same two user-defined properties being conveyed using `Property` elements:

```
<adsml:Properties>
  <adsml:Property xsi:type="lat:ChangeFlag">true</adsml:Property>
  <adsml:Property xsi:type="lat:LastUpdated">2001-12-17T09:30:47
</adsml:Property>
</adsml:Properties>
```

This is a more compact syntax than `LabeledProperty`, but in order to use it, you must first create a user extension schema in which you declare each property that you want to use. (For more information about creating and referencing a user extension schema, see "Implementing controlled vocabularies with schema-based validation" later in this document.)

The advantage of doing this is that it allows schema-validation of the property elements. The potential disadvantage is that your messages will no longer be

Copyright © 2010 AdsML Consortium. All rights reserved.

valid according to the un-modified AdsML schema. This is an unavoidable side-effect of using the extension schema mechanism.

Here is the syntax to declare these two properties in a user extension schema:

```
<xs:simpleType name="ChangeFlag">
  <xs:restriction base="adsml:PropertyRootType"/>
</xs:simpleType>
<xs:simpleType name="LastUpdated">
  <xs:restriction base="adsml:PropertyRootType"/>
</xs:simpleType>
```

This example shows a minimal declaration, in that it allows use of the properties but does not restrict the values that can be conveyed in them. It is possible to define much more restrictive XML schema validations for each property, if you wish.

7.4 Profiles

A "profile" is a subset of a standard that is defined in order to reduce complexity, facilitate implementation, and increase interoperability. For example, the XML standard is a profile of a more complex standard called SGML. XML carved away most of the optional features of SGML and specified a narrow subset of it. The XML standard is much easier and less expensive to implement than SGML was, and it allows its users to solve many business problems. However, there are some activities for which SGML is still used, because those activities require features that are not available in XML.

It has always been the intention of the AdsML working group that regional or industry associations will create profiles of the AdsML Framework, in which they specify which parts of the Framework should be implemented by their members in order to solve advertising-related business problems that are common to members of that group. Once a suitable profile has been defined, vendors can implement software that supports just the features and functionality in that profile, and participating organizations can use such software to exchange messages which are known to address their industry's business requirements in an interoperable fashion.

Profiles are created by using the same AdsML capabilities – primarily configuration checklists and controlled vocabularies – that are available to any organization wishing to implement AdsML-based e-commerce. The only significant differences between profiles and less formal AdsML configuration activities are that profile definitions are normally made publicly available so that many organizations can benefit from them, and AdsML messages which conform to a formal profile are then tagged with the name of that profile. This allows software systems to identify when an incoming message conforms to a given profile, and to validate, route and process that message accordingly.

An AdsML profile **MUST** be a strict subset of its parent standard. Therefore, the changes in a profile **MUST** only affect optional aspects of the standard being profiled. Anything that is declared mandatory in the parent standard **MUST** also be mandatory in the profile, and any XML message that conforms to a profile **MUST** also conform to the parent standard's XML schema.

An organization can create as many AdsML profiles as it needs.

7.4.1 Types of information

It is anticipated that the following types of information will commonly be defined in an AdsML profile:

Copyright © 2010 AdsML Consortium. All rights reserved.

1. Standard(s) and types of information
    a. The AdsML standard to which this profile applies. It is **RECOMMENDED** that each profile apply to just one AdsML standard.
    b. The types of information that will be conveyed using this profile.
    c. Whether or not the AdsML Envelope will be used, and if so, which of the Transmission/Response modes will be supported.
2. Messages and choreography
    a. The list of AdsML message types that are available for use in this profile. This will often be a subset of the full suite of messages defined by the AdsML standard to which the profile applies.
    b. Subsets of the *Advertising Components Interactions Analysis* that must be supported by users of this profile, i.e. business situations in which a specific AdsML message must be sent.
    c. Conversely, subsets of the *Advertising Components Interactions Analysis* that are not supported by users of this profile, i.e. situations in which specific AdsML messages which are otherwise supported in the profile must not be sent.
    d. Use of datagram vs. request-response mode, and if datagram mode, the direction in which the messages will be sent.
3. Element cardinality
    a. Elements that AdsML has declared as optional but whose use is mandatory in the profile, including substitution groups (for example, `placement.newspapermagazine` in AdsMLBookings).
    b. Conversely, optional elements or substitution groups that must not be used in messages conforming to the profile.
    c. Optional elements or substitution groups that remain optional in the profile. (By default, all optional elements and substitution groups fall into this category unless they are explicitly excluded or made mandatory.)
    d. Maximum occurrences of potentially repeatable elements which are less than the maximum occurrence allowed by the AdsML schema.
4. Mandatory use of particular controlled vocabularies.

This list is intended to be indicative rather than normative.

7.4.1.1 Example

For example, an extremely simple profile of AdsML Bookings intended for use in a pilot project might specify:

Profile name and ID	AdsMLBookings US magazine pilot myorg.org:2006-01-01:AdsMLBookingsUSMagPilot:1
Standard being profiled	AdsMLBookings v. 1.0
AdsML Envelope?	Yes "Store and resend until acknowledged" mode
Information overview	Magazine display ad orders (no quotations or reservations; no inserts)

Copyright © 2010 AdsML Consortium. All rights reserved.

	New orders only (no changes or cancellations)
Messages	AD-O, AD-OR
Choreography	Request-response mode The message exchange consists of a new order (AD-O) from buyer to seller, followed promptly by an administrative response indicating its receipt. In due course the seller sends a a business response (AD-OR) which indicates either acceptance or rejection of that order. The AD-OR also triggers a prompt administrative response. No other messages are included.
Element inclusions	Use `.NewspaperMagazine` structures rather than `.Generic` (A list of mandatory elements or types of information would go here.)
Element exclusions and limitations	One placement per booking; No placement groups; Max 10 dates per placement, all within a single calendar month; No pricing information. (A list of other AdsML elements or types of information that must not be included in the messages would go here. Or, the profile could list the optional elements and types of information that are allowed, if that is easier to document.)
Controlled vocabularies	(A list of elements and their required controlled vocabularies goes here. These could be from the AdsML CV list, or other vocabularies that were defined or selected.)

7.4.2 Profile identification

AdsML does not specify a format for storing or publishing an AdsML profile. It is up to the organization that creates a profile to codify it in a suitable format.

However, each organization that defines an AdsML profile **MUST** assign a unique identifier to that profile. All AdsML messages conforming to a given profile **MUST** include that profile's unique identifier in the message body, according to the following rules:

In the AdsML item-level standards such as AdsMLBookings and AdsMLMaterials, the profile identifier is conveyed in the *`schemaProfile`* attribute of the root element. In the AdsML Ad Ticket format, the profile identifier is contained in the `SchemaProfile` element. In both cases, the structural rules for an element or attribute are based on the AdsML `VersionedQIDType` described in the AdsML Type Library specification and **MUST** be followed.

According to those rules, a sample profile identifier value might look like this: "myorg.org:2006-01-01:AdsMLBookingsUSMagPilot:1". The profile identifier consists of three mandatory sections separated by colons, followed by a fourth, optional section, which is also preceded by a colon when it is used. (In the example above, all four sections are provided.)

Copyright © 2010 AdsML Consortium. All rights reserved.

1. The first section of the profile identifier is an internet domain name that was controlled by the organization that defined the profile at the time the profile was published.
2. The second section is a date on which the domain name belonged to the organization that defined the profile. The date **SHOULD** be the publication date of the profile in question, and **MUST** be a date on which the publishing organization controlled the specified domain name.
3. The third section is the generating organization's name for the profile. This can be any value, however it **SHOULD** be a name that indicates the scope and/or purpose of the profile and it **MUST** be a unique string within the set of all profile names that share the same domain name and date.
4. The optional fourth section is a version identifier for the profile. This may be in any format that the organization chooses.

The colon (":") character is reserved for use as a separator between the four sections of the profile identifier. The colon character **MUST NOT** appear in any of the data sections themselves. Following this pattern will make it easier for developers to write a parsing routine that can deal with any profile identifier.

The default value for a profile identifier in any AdsML message is blank (omitted), which indicates that the message conforms to the AdsML specification as a whole and no formal profile has been applied.

7.5 Trading partner agreement

A Trading Partner Agreement (TPA) is an agreement between two (or more) parties that governs how they will do e-commerce together. A TPA will encompass many different types of information, from legal and contractual decisions, to workflow patterns, to technical details regarding system configuration. In practice a TPA usually turns out to be more than one document, some of which are considerably more formal than others, which are discussed and agreed separately by various parties within the organizations.

AdsML does not define a process for arriving at a trading partner agreement, or a format in which to record the results. These will necessarily vary from one organization to the next. But in general, parties wishing to engage in e-commerce communications should consider at least the following types of issues and record their mutual decisions in a format that is acceptable and accessible to all parties:

Legal and contractual

- Who are the parties?
- When did/will the TPA come into force, and when will it expire?
- How will the AdsML e-commerce messages relate to any contracts or agreements that are in force between the parties?
- Under what circumstances does an AdsML message constitute an acceptable "order", "delivery", "invoice", etc? What specific information must such messages contain in order to replace the current, non-AdsML methods for transmitting these things?
- How does "acceptance" of an AdsML order or delivery relate to the parties' contractual obligations or to any Terms and Conditions that they may each wish to assert?

Transmission and security

- How will the AdsML messages be transmitted between the partners (HTTPS, FTP, SOAP service calls, etc.)?

Copyright © 2010 AdsML Consortium. All rights reserved.

- To what address will the messages be sent? (Note that this can vary according both to transmission type and message type.)
- What if an address is unavailable? What about backup or alternate addresses or transmission methods?
- What type of IDs will the parties use to identify themselves and their partners (e.g. tax identification number, DUNS code, or similar unique value)?
- What tools will the parties employ to ensure that the messages always go through unchanged and cannot be corrupted, intercepted or spoofed by other parties? (Encryption, encoding, digital signatures, etc.)

AdsML Envelope Processing (if used)

- Which type of Transmission/Response model should be used ("send and forget" or "store and resend until acknowledged")?
- What version(s) of the AdsML Envelope standard will be used?
- What addressing redirection, if any, should be performed by an AdsML Envelope processor when dealing with a partner?
- Will priority handling be supported?

Workflow

- For which types of business transactions will the parties implement AdsML-based e-commerce?
- What will be the primary workflows?
- How will the parties handle changes, cancellations, repeats, pickups, errors, and other common workflow variations?
- What will be the relationship between the AdsML messages and the current non-AdsML procedures?
- At what point will the non-AdsML workflow be shut down? If both AdsML and non-AdsML workflows will run in parallel, what happens when there are discrepancies between the two?
- How will hybrid workflows be supported, for example, where an AdsML-booked order is modified by a telephone message

AdsML Message Processing and Configuration

- Is there a defined profile of AdsML that will be used? (If so, this will supply many of the other answers in this section.)
- Which AdsML standards will the parties use, and which versions of them?
- Which messages and types of information will the parties exchange in order to support the agreed workflows?
- What standard or format will be used to represent each such type of message (e.g. IfraAdConnexion, AdsMLBookings, AdsMLMaterials, CREST, SPACE/XML, a non-XML EDI standard, etc.), and what are the supported version numbers?
- Will the parties implement datagram or request-response communications?
- Is an administrative response required for every message?
- Which types of optional information will be considered mandatory in, or conversely excluded from, the messages?
- For which elements will the parties require use of controlled vocabularies, and for each such element, which vocabulary will be used and how will it be identified in the messages? Note that this can encompass a great deal of

Copyright © 2010 AdsML Consortium. All rights reserved.

critical information, including party identifiers, classifications, publication lists, size, color, positioning, etc.

- For each message type, which parts of the Configuration Checklist in the relevant Usage Guidelines will apply?

Error Handling

- After what amount of time will the lack of response to a message be considered an error requiring corrective action?
- Are there other defined circumstances that should be treated as errors?
- If an error is detected, what are the procedures to resolve it, including people to contact on the other side? This needs to be answered twice:
    - Transmission and envelope error procedures
    - Business information error procedures

Although this list can seem daunting, trading partners in early AdsML projects have consistently reported that the TPA development process has served to improve communications between them. Ironically, the move to automated e-commerce can result in more direct contact between the parties rather than less (at least during the setup and testing stages), which provides an opportunity for the parties to improve business relationships as well as workflows and procedures.

7.5.1 Process Partnership Agreement

The Process Partnership Agreement, or PPA, is the subset of the information in a Trading Partner Agreement that must be available to an AdsML Processor in order for it to communicate properly with each of its communication "partners".

For example, the PPA can identify the types of information and formats that can be sent to a given communications partner, any addressing redirection that should be performed by an AdsMLEnvelope processor when dealing with that partner, and the transport mechanism and physical address to which AdsML messages intended for that entity should be sent.

A single logical rule set governs AdsML processing between any two partners. The rule set is contained in each partner's PPA. While most of these rules will be symmetrical, symmetry is not a requirement: a different set of rules can apply to each partner depending on whether it is Sending information to the other partner or Receiving information from it, provided that at a business level the two parties have agreed to operate in this fashion.

7.6 Party identification

In order to implement AdsML-based e-commerce, it is necessary for trading partners to agree on the types of identifiers that they will use to identify the parties that are referenced in an AdsML message. In this regard, parties fall into two broad categories:

- Parties for which it is important to the trading partners that they be able to uniquely and unambiguously identify that party. These are usually the parties with which the trading partners have a financial relationship, and as a result, for which they are likely to maintain an internal database record with its own unique ID and other information. In the Bookings workflow, typical examples are the Buyer and Seller of advertising, and often the Advertiser itself.
- Parties for which it is useful for trading partners to know their name, but it is not important that they have an unambiguous identifier. These are usually

Copyright © 2010 AdsML Consortium. All rights reserved.

third parties that are more loosely related to the transaction, but whose names may be helpful when communicating about that transaction.

Trading partners usually know quite clearly which parties to a given type of transaction need to be unambiguously identified. When this is the case, it is important to agree in advance on just how those parties will be identified. There are several possible approaches:

1. The trading partners agree to use identifiers provided by a third-party, such as government tax ids (VAT number, EIN number, etc.), or D-U-N-S or S&P numbers.
2. The trading partners share one or both of their lists of identifiers with each other, and agree to use those identifiers in their messages
3. The trading partners decide to enlist an industry association or other suitable group to maintain a master list of trading partners, each with its own unique identifier, which can then be referenced in the messages
4. The trading partners fail to agree on a shared set of identifiers, forcing each of them to add extra processing to incoming messages in order to "map" the party identifiers to their own internal system identifiers.

Each of these approaches has varying degrees of strengths and weaknesses, and unfortunately, all of them are in use somewhere today. To the extent possible, it is **RECOMMENDED** that trading partners agree to use identifiers provided by a suitable external authority (option 1, above), and if necessary, supplement this approach by also using one of the other mechanisms as well. Each `Party` in an AdsML message can be associated with multiple `Identifiers`, and each `Identifier` is labeled with a string that indicates its type or source. This allows trading partners to provide as many different types of identifiers as necessary to ensure that the message recipient will be able to resolve the identification references correctly.

For more information about the `Party` and `Contact` structures, see the *AdsML Type Library* specification.

8 Implementing controlled vocabularies with schema-based validation

8.1 Introduction & rules for use

AdsML provides a facility that allows specific controlled vocabularies (CVs) to be used and validated if desired. This has been achieved by specifying a default type known as a 'root type' for the element context. For instance, the `adsml:CodeValue` element is defined as an `adsml:CodeRootType` where a CV value can be provided and also be validated as a part of the overall validation of the message.

The `adsml:CodeRootType` is in fact based on a simple string data type allowing any string value to be used in case no value control is required, or done in custom application logic.

In cases where an XML Schema based validation of the CV values is preferred, the CVs first need to be defined using XML Schema in a schema file. AdsML provides a set of core built-in CVs such as a set of status codes, ad positioning codes etc. These CVs are defined in the AdsML Controlled Vocabularies schema and specification.

Copyright © 2010 AdsML Consortium. All rights reserved.

But also trading partners, or industry associations, may use the same approach as taken by AdsML to create specific libraries of CVs that may be used instead of the CVs recommended by AdsML.

Where one of the 'root types' is specified as the type of an element, this indicates that the root type may be replaced (or 'substituted') in an AdsML instance document by a CV type defined in an XML Schema and derived from the root type.

In order to validate against an XML Schema defined CV type, the CV type must be specified in XML business messages by using the *xsi:type* attribute of the element in question. For instance, the following sample shows how the value of the adsml:CodeValue element is defined as being a member of the adsml-cv:AdsMLStatusCodeCV:

```
<adsml:CodeValue xsi:type="adsml-cv:AdsMLStatusCodeCV">
Completed
</adsml:CodeValue>
```

AdsML specifies controlled vocabulary root types for use in element contexts where precise values are, or are likely to be, required - that is, a user may want to specify and use a controlled vocabulary of values.

Functionally, therefore, where AdsML specifies a controlled vocabulary 'root type' this mechanism allows an element value to take data at 3 levels:

1. Default data type - any value of a data type allowed by the root type (e.g. string, Boolean, integer, et cetera) can be specified for use. The root type is the default data type of the element and the value is not confined in any way beyond the data type allowed by the root type
2. AdsML defined specific value – an AdsML controlled vocabulary value (e.g. particular Item types, a priority rating in a range, et cetera). The AdsML controlled vocabulary in question is specified by using the *xsi:type* attribute of the element in question in the instance document
3. User defined specific value – a user-defined controlled vocabulary value (e.g. a list of Item types to cater for specific types of item that are particular to the individual trading circumstances of specific trading partners, a priority rating in the range 1-4, et cetera). The user-defined controlled vocabulary in question is specified by using the *xsi:type* in the same way as for the AdsML controlled vocabularies above.

Consequently, the AdsML controlled vocabulary mechanism allows the user flexibility and control in how they record the data values that they use in the AdsML message and enables trading partners to tailor their needs to their individual operating contexts while maintaining interoperability. This allows the user to exercise control over the extent to which they define precise values for use or not and the extent to which they decide to validate values using XML Schema validation. For example, if a user wants to record an encryption method as a string they can do so and if they want to specify that only two specific encryption methods are used then they are able to specify that and enforce the constraint by using XML Schema validation.

The following sections describe in more detail the approach to creating XML Schema based CVs.

8.1.1 Root types

As stated, the 'root types' are the default types of the element contexts in AdsML where a controlled vocabulary can be specified for use. The root type specifies the data type of that context and the use of this default mechanism enables variable

Copyright © 2010 AdsML Consortium. All rights reserved.

degrees of control to be exercised over the value that appears in this context in an AdsML instance document. If no specific value is required, then the root type is used by default and the only constraint to be imposed on values is validation against the base type of the root type. Alternatively, if a specific value is required, then a controlled vocabulary can be used - the default root type is overridden and substituted in an instance document by the use of either an AdsML or a user defined vocabulary derived from that root type.

Depending on the usage context, controlled vocabulary root types may be either simple or complex types. Simple root types are the most common and can be used in element contexts where the element has no attributes; complex root types can only be used in element contexts where the element has attribute(s)[7].

A simple root type is no more than a built-in XML data type under a different name, perhaps with one of the facets restricted – for example, the maximum length of characters allowed. Simple root types are always derived by restriction.

8.1.2 AdsML defined Controlled Vocabularies

AdsML provides a set of basic controlled vocabularies that can be substituted and used for root type contexts. The AdsML Controlled Vocabularies are defined in the AdsML Controlled Vocabulary schema and are the optional but recommended official AdsML controlled vocabularies available for use in AdsML.

Note that in some usage context where there is a requirement to specifically control values that an AdsML CV will directly be specified as the declaration type of an attribute or element context in the schema. An example of this is in the AdsML Envelope where the `ItemType` element is declared as the `adsml:MessageClassCV` controlled vocabulary. Such CVs are defined not in the AdsML Controlled Vocabulary schema, but rather in schemas local to each standard, or in the AdsML Type Library.

8.1.3 User defined Controlled Vocabularies

If users wish to define their own controlled vocabularies for use in root type contexts in order to meet the requirements of their particular usage context, then they **SHOULD** do so by creating a user-extension schema in which their controlled vocabularies are defined. (See the chapter on *User defined controlled vocabularies* for how to create a user-defined controlled vocabulary extension to AdsML.)

8.1.4 Guidelines for controlled vocabulary use in AdsML

In an AdsML message instance, controlled vocabulary values **MAY** be recorded by using specific AdsML or user-defined controlled vocabularies derived from the root types.

AdsML recommends that AdsML defined controlled vocabularies **SHOULD** be used to record controlled vocabulary values wherever possible. The AdsML defined controlled vocabularies can also be used as a base for creating user-defined controlled vocabularies by derivation.

[7] Element contexts may require the use of complex types for controlled vocabulary root types in circumstances where the element requires an attribute value to also be recorded. In order to allow the controlled vocabulary root type to be substituted with a more specific AdsML or user-defined controlled vocabulary, the attribute(s) must be added to the root type. The only instance of this at the present time is for the Format element, where the Format element carries a version attribute. Furthermore, the AdsML Technical WG does not plan in the future to create any other cases that require complex root type derivation, but instead plans to use the less complicated simple type derivation approach.

Copyright © 2010 AdsML Consortium. All rights reserved.

When an AdsML Controlled Vocabulary does not contain the required values, or if the user does not want to specify and/or validate precise controlled vocabulary values, then the user **SHOULD** simply use the root type of the element in question 'as is' and so need take no further action as the root type is by nature the default type of the element in question.

Where circumstances require the specification and/or validation of precise controlled vocabulary values that are not already present in the AdsML Controlled Vocabularies, then the user **SHOULD** define their own list of controlled vocabulary values as a user-specific controlled vocabulary by deriving a new type containing those values from the relevant root type(s) or AdsML controlled vocabulary this type(s).

If a user wishes to create and use their own controlled vocabularies, then the user **SHOULD** do so by creating and using an extension schema as defined in the next chapter, User Extensions.

8.1.5 Illustrative example of the AdsML controlled vocabulary mechanism

An example of controlled vocabulary use is where the required values of an element are only 'X' and 'Y' and so these values need to be specified as the only valid values for the element. Rather than having a data type of 'string', string is restricted to allowed values of 'X' and 'Y'.

To illustrate this by example taken from the AdsML Envelope 1.1 standard, the `Encoding` element used to identify the encoding of data carried inside an `ItemContent` element is specified as `EncodingRootType`. The `EncodingRootType` is specified as `ShortTokenType`, a restriction of the `xs:token` data type to a maximum length of 50 characters. AdsML provides an AdsML controlled vocabulary for encoding types - `AdsMLEncodingCV` – which can be specified for use in the `Encoding` element context by substituting it for the root type using the *`xsi:type`* attribute of the `Encoding` element in a message instance. The code example shows schema fragments defining the `Encoding` element, the `EncodingRootType`, and two types derived from it - `AdsMLEncodingCV`, and a '`UserDefinedEncodingCV`'.

```
<xs:element name="Encoding" type="EncodingRootType"/>

<xs:simpleType name="EncodingRootType">
 <xs:restriction base="ShortTokenType"/>
</xs:simpleType>

<xs:simpleType name="AdsMLEncodingCV">
 <xs:restriction base="EncodingRootType">
  ...
  <xs:enumeration value="base64Binary"/>
  ...
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="UserDefinedEncodingCV">
 <xs:restriction base="EncodingRootType">
  ...
  <xs:enumeration value="US-ASCII"/>
  ...
 </xs:restriction>
</xs:simpleType>
```

In an instance document, the user has choice in how they specify the value of the `Encoding` element – either as a plain token, using the default root type, or as a

Copyright © 2010 AdsML Consortium. All rights reserved.

specific value, using a controlled vocabulary derived from it, substituting the derived type using the *xsi:type* attribute. In the illustration below, all three usage variants are shown – a plain (tokenized) string, the use of the AdsML Controlled Vocabulary, and the use of a user defined controlled vocabulary,

```
...
<Encoding>base64</Encoding>
...
<Encoding xsi:type="AdsMLEncodingCV">base64Binary</Encoding>
...
<Encoding xsi:type="UserDefinedEncodingCV">US-ASCII</Encoding>
...
```

8.2 User defined controlled vocabularies

As explained in the previous sections, AdsML allows users to define and use specific values in an AdsML instance wherever a root type has been set as the type of an element context. If users need to define their own controlled vocabularies to meet the requirements of their particular usage context, they **SHOULD** do so by creating a user-extension XML Schema in which the user-defined controlled vocabularies are defined. Even though such a formal schema definition is not required by AdsML, it is recommended that one be created as it will enable schema based validation and a formal, standard, definition language.

This section describes rules for creating user-defined controlled vocabularies, how to create a user extension schema, and how to use these user extensions in an AdsML instance document.

8.2.1 Rules for creating user-defined controlled vocabularies

The following rules **MUST** be followed when deriving new controlled vocabularies:

- All derivation **MUST** be by restriction only, so as to ensure that the values of derived types are a subset of their base root types
- When deriving from a complex root type, a restriction **MUST NOT** restrict any attribute(s) declared on the complex root type. The occurrence of an attribute **MUST NOT** be changed.

In addition, the following rules **MUST** be followed when deriving new controlled vocabularies from pre-defined AdsML controlled vocabularies:

- All derivation can be done using either restriction or extension but **MUST** be by restriction from the original root type used as a base for the AdsML controlled vocabulary.

New types for controlled vocabulary values are defined by manipulating the base type's 'constraining facets'. The constraining facets are the properties of a data type that can be used to constrain the value space of a data type as defined by W3C in “XML Schema Part 2: Data types”.

8.2.2 Creating user-defined controlled vocabularies

This section details how a user creates a user-defined controlled vocabulary by deriving from a root type defined in the AdsML Type Library schema, deriving by restriction only. As detailed in the previous section, root types can be simple or complex, according to the context in which they appear.

Copyright © 2010 AdsML Consortium. All rights reserved.

Deriving to specify properties for use in the `Property` element is identical to the method for deriving from a simple root type, but conceptually differs in that the name of the derived type is used to provide the name part of the property name:value pair. Due to this, deriving property types has a separate section.

8.2.2.1 Deriving from a simple root type

When creating a controlled vocabulary from a simple root type and defining it using XML Schema, the procedure is as follows,

- Create a new simple type, deriving the new type by restriction, specifying the root type as the base
- Define the required values by using the constraining facets that can be applied to the root type - i.e. the constraining facets that can be applied to the base type of the root type

To illustrate, a user deriving from the `ContactRoleRootType` to create a list of the people they are likely to contact - e.g. 'Advertiser', 'AdSpaceBuyer', 'AdAgency', 'ReproHouse', 'Deliverer', and 'Publisher'. The user,

1. Creates a new simple type by restriction, giving an appropriate name to the simple type – e.g. '`AdvertisingContactCV`'. The root type - `ContactRoleRootType` - is specified as the base type of the new type using the *`base`* attribute of the `xs:restriction` element nested inside the `xs:simpleType` element.

```
<xs:simpleType name="AdvertisingContactCV">
 <xs:restriction base="ContactRoleRootType"></xs:restriction>
</xs:simpleType>
```

2. Decides which facet(s) to constrain in order to create the desired values and then performs the necessary restriction. In this case, the root type `ContactTypeRootType` is a string type derived from `LongStringType` and so the user is able to specify values using the *`length`*, *`minLength`*, *`maxLength`*, *`pattern`*, *`enumeration`* and *`whiteSpace`* constraining facets. The user decides to specify the desired values of '`Advertiser`', '`AdSpaceBuyer`', '`AdAgency`', '`ReproHouse`', '`Deliverer`', and '`Publisher`' as enumerated values, and specifies this using the *`enumeration`* facet, enumerating the desired values using `xs:enumeration` elements nested inside the `xs:restriction` element.

```
<xs:simpleType name="AdvertisingContactCV">
 <xs:restriction base="ContactTypeRootType">
  <xs:enumeration value="Advertiser"/>
  <xs:enumeration value="AdSpaceBuyer"/>
  <xs:enumeration value="ReproHouse"/>
  <xs:enumeration value="AdAgency"/>
  <xs:enumeration value="Deliverer"/>
  <xs:enumeration value="Publisher "/>
 </xs:restriction>
</xs:simpleType>
```

An instance would appear like this,

```
...
<Subject xsi:type="AdvertisingContactCV">Advertiser</Subject>
...
```

If greater flexibility is desired, the user specifies multiple facets. In this example, the user derives a controlled vocabulary for recording advertising contacts. The user doesn't want to specify precise values, rather to set minimum and maximum

Copyright © 2010 AdsML Consortium. All rights reserved.

lengths values of '5' and '10' so that only values between the lengths of 5-10 characters can be used.

```
<xs:simpleType name="UserMinMaxLengthAdvertisingContactCV">
 <xs:restriction base="ContactRoleRootType">
  <xs:minLength value="5"/>
  <xs:maxLength value="10"/>
 </xs:restriction>
</xs:simpleType>
```

An instance could appear like this,

```
...
<Subject
xsi:type="UserMinMaxLengthAdvertisingContactCV">UpToTenCha</Subject>
...
```

For more information about defining simple types, see *XML Schema Part 1: Structures*, Section 3.14 Simple Type Definitions (http://www.w3.org/TR/xmlschema-1/#Simple_Type_Definitions).

8.2.2.2 Deriving to specify Properties

Trading partners wishing to specify properties using the `adsml:Property` element will have to derive and create their own property types. This is because AdsML relies on the name of the derived type to provide the name part of a name:value property pair. When using properties, the name is given by the *`xsi:type`* attribute of the `adsml:Property` element in the instance, and the value is given by the element data content of the `adsml:Property`. AdsML asserts this constraint to control undefined property use in AdsML. The constraint effectively restricts property use in AdsML to only specific properties that trading partners have agreed to use in their business context and have specified in their extension schemas. Consequently, if trading partners wish to use and validate properties then they **MUST** specify a user extension schema.

Each required property is derived from `PropertyRootType`, with the name of the derived property type being the name of the property as it will appear in the instance. The procedure for deriving from `PropertyRootType` is as described in the section '*Deriving from a simple root type*'.

8.2.2.3 Deriving from a complex root type to specify element content

A complex root type is different from a simple root type because it is derived by extension, the extension used to add attribute(s) to the base type. These attributes **MUST NOT** be altered when deriving from the complex root type to create specific controlled vocabulary values. AdsML enforces this constraint by not allowing extension of complex root types; root types **MUST** only be derived from by restriction. Further attributes or children elements cannot be added to complex root types, thereby ensuring that the underlying information model of AdsML is not changed and so enforcing the interoperability of AdsML. The constraint is enforced in the AdsML Schema by giving the *`block`* property of each complex root type the value of '`extension`', thereby preventing extension.[8]

[8] Blocking extension on a complex root type prevents further attribute(s) being added to the root type during the definition of a user-specific controlled vocabulary. Such an extension would not be an 'extension by restriction' as AdsML allows, but an extension to the information model of AdsML – i.e. changing that information model by the addition of one or more attributes – and so is not allowed. Setting block to extension enforces the AdsML 'extension by restriction' only constraint & ensures the interoperability of AdsML.

Copyright © 2010 AdsML Consortium. All rights reserved.

NOTE: Complex root type derivation is currently applicable to only one context, the FormatRootType. The AdsML Technical WG will avoid introducing any other cases of complex root type derivation, as the less complicated simple type approach has been found to be sufficient.

When creating a controlled vocabulary from a complex root type, the procedure is as follows,

- Create a new complex type, specifying the type as simple content. The new type is derived by restriction, specifying the root type as the base
- Define the required values by using the constraining facets that can be applied to the root type - i.e. the constraining facets that can be applied to the base type of the root type

To illustrate, a user deriving from the `FormatRootType` to create a list of the formats they use - e.g. '`AdsMLBookings`', and two proprietary formats - '`InHouseXML`', and '`PublisherProprietaryFormat`'. The user does this as follows,

1. Creates a new complex type with simple content by restriction, giving an appropriate name to the complex type – in this case, for example, '`UserFormatsCV`'. The root type - `FormatRootType` - is specified as the base type of the new type using the *`base`* attribute of the `xs:restriction` element nested inside the `xs:simpleContent` child of the `xs:complexType` element.

```
<xs:complexType name="UserFormatsCV">
 <xs:simpleContent>
  <xs:restriction base="FormatRootType"></xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

2. Decides which facet(s) to constrain in order to create the desired values and then performs the necessary restriction. In this example, the user decides to specify the desired values of '`AdConnexion`', '`InHouseXML`', and '`PublisherProprietaryFormat`' as enumerations, using `xs:enumeration` elements nested inside the `xs:restriction` element.

```
<xs:complexType name="UserFormatsCV">
 <xs:simpleContent>
  <xs:restriction base="FormatRootType">
   <xs:enumeration value="AdsMLBookings"/>
   <xs:enumeration value="InHouseXML"/>
   <xs:enumeration value="PublisherProprietaryFormat"/>
  </xs:restriction>
 </xs:simpleContent>
</xs:complexType>
```

An instance would appear like this,

```
...
<Format version="1.1" xsi:type="UserFormatsCV">AdsMLBookings</Format>
...
```

For more information about defining complex types, see *XML Schema Part 1: Structures*, Section 3.4 Complex Type Definitions (http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions).

8.2.3 Creating a user extension schema

User-defined controlled vocabularies expressed using XML Schema are defined in a user extension schema, with the user extension schema used in place of the

Copyright © 2010 AdsML Consortium. All rights reserved.

relevant AdsML schema during instance creation and validation. Users **MUST** agree any user-defined extensions to be used by them in the Trading Partnership Agreement (TPA) that governs their business relationship and the Process Partnership Agreement (PPA) that governs their AdsML processing arrangements. Any user extension schema created **MUST** be specified in the Process Partnership Agreement (PPA) so that they can be identified and accessed at the system level by AdsML processors.

When creating an extension schema AdsML recommends that a target namespace **SHOULD** be set for the extension schema, as per the reasons set out for namespace use in the *AdsML Envelope* specification *Section 2.5.5 AdsML namespace*. AdsML makes no recommendations as to whether users specify schema version, identification, and language management for user-defined extension schema (See the *AdsML Envelope* specification, *Section 2.5.4 Schema – version, identification, and language*); users should record this information as required by their business circumstances.

The procedure for creating a user extension schema is as follows,

- Create a new XML Schema, naming the schema to indicate that it is a user-specific extension of the AdsML schema in question. For example, '`AdCoExtension-AdsMLEnvelope-1.1.xsd`'

- Within the root `xs:schema` element, the user **MUST**,
    a. Specify the namespace(s) of the AdsML Framework standards required for the extension

- Within the root `xs:schema` element, the user **SHOULD**,
    b. If specifying a namespace for the user extension schema, add a *`targetNamespace`* attribute, assigning this attribute the value of the namespace for the user extension schema. The user **MAY** assign a prefix to the user namespace
    c. Qualify the schema to show element and type namespace qualifiers by assigning the *`elementFormDefault`* attribute the value of '`qualified`'
    d. Qualify the schema to hide attribute namespace qualifiers by assigning *`attributeFormDefault`* attribute the value of '`unqualified`'

- Within the root `xs:schema` element, the user **MAY**,
    e. If specifying version, identification, and language information for the user extension schema, specify this using the optional *`version`*, *`id`*, and *`xml:lang`* attributes available for this purpose (See the *AdsML Envelope* specification, *Section 2.5.4 Schema – version, identification, and language*).

- Immediately after the `xs:schema` element, import the required AdsML Framework Schema using `xs:import` element(s). The *`namespace`* attribute of `xs:import` is assigned the value of the required AdsML schema namespace. The *`schemaLocation`* attribute of `xs:import` may be assigned the location of where the AdsML Schema being referenced is hosted.

- Define the required controlled vocabularies in the user extension schema, as per the rules and procedures specified in this section.

To illustrate, user extension schema can appear as follows:

Namespace user extension schema declaration

Copyright © 2010 AdsML Consortium. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.user.com/adsml/userextension"
xmlns="http://www.user.com/adsml/userextension"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:adsml-en="http://www.adsml.org/adsmlenvelope/1.1"
xmlns:adsml="http://www.adsml.org/typelibrary/1.1"
xmlns:adsml-cv="http://www.adsml.org/controlledvocabularies/2.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:import namespace="http://www.adsml.org/adsmlenvelope/1.1"
schemaLocation="AdsMLEnvelope-1.1-Main-AS.xsd"/>
 <xs:import namespace="http://www.adsml.org/typelibrary/1.1"
schemaLocation="AdsMLTypeLibrary-1.1-AS.xsd"/>
 <xs:import namespace="http://www.adsml.org/controlledvocabularies/2.0"
schemaLocation="AdsMLControlledVocabularies-2.0-AS.xsd"/>
<!-- controlled vocabulary type definitions here. -->
</xs:schema>
```

A no namespace version of the above would look almost identical, except for not having the *targetNamespace* and default namespace defined (the *xmlns* attribute).

8.2.4 Validating against user defined controlled vocabularies

When validating against the controlled vocabularies defined in the user extension schema, trading partners must replace the original AdsML schema with the user extended schema importing the original AdsML schema. With this exception, the process is otherwise identical to validation using original AdsML schemas.

Copyright © 2010 AdsML Consortium. All rights reserved.

9 References

- [IETF RFC 2387] E. Levinson. *The MIME Multipart/Related Content-type*. Internet Engineering Task Force (IETF), August 1998 (http://www.ietf.org/rfc/rfc2387.txt)
- [IETF RFC 1766] H. Alvestrand. *Tags for the Identification of Languages*. Internet Engineering Task Force (IETF), Request for Comments: 1766, March 1995 (http://www.ietf.org/rfc/rfc1766.txt)
- [ISO/IEC 10646] ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane*. International Organization for Standardization, 1993
- [W3C] W3C (World Wide Web Consortium). Ed. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation, 16 August 2006, edited in place 29 September 2006. (http://www.w3.org/TR/REC-xml).

10 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,
    - Christian Ratenburg (CCI Europe) – cr@ccieurope.com

Copyright © 2010 AdsML Consortium. All rights reserved.