



# XMP – Extensible Metadata Platform

**Version 1.5**

*September 14, 2001*

**ADOBE SYSTEMS INCORPORATED**


**Corporate Headquarters**

345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

<http://www.adobe.com>



Copyright © 2000–2001 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Adobe Systems Incorporated.

Adobe, the Adobe logo, Acrobat, Acrobat Distiller, Framemaker, InDesign, Photoshop, PostScript, the PostScript logo, and XMP are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Macintosh, and QuickTime are trademarks of Apple Computer, Inc., registered in the United States and other countries. UNIX is a trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. All other trademarks are the property of their respective owners.

***This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third party rights.***



# Contents

<b>Chapter 1</b>	<b>Preface</b>	<b>1</b>
1.1	About This Document	1
1.2	Audience	1
1.3	Assumptions	1
1.4	How This Document Is Organized	1
1.5	Conventions used in this Document	2
1.6	Where to Go for More Information	2
<b>Chapter 2</b>	<b>XMP – Extensible Metadata Platform</b>	<b>5</b>
2.1	Introduction	5
2.2	Background	6
2.3	Scope of XMP	7
2.4	Model and Terminology	8
2.5	Granularity of XMP Metadata Associations	13
<b>Chapter 3</b>	<b>XMP RDF Data Interchange Format<sup>15</sup></b>	
3.1	Introduction	15
3.2	Background	15
3.3	RDF Data Model	15
3.4	How XMP Uses the RDF Data Model	16
3.4.1	Description Object	16
3.4.2	Repeated Properties	17
3.4.3	Schemas and Namespaces	17
3.4.4	Localized Property Values	18
3.4.5	Extensibility	19
3.4.6	Aliasing	20
3.4.7	Resource Identification	20
3.4.8	Normalization of Metadata	21
3.5	Representation and Storage of Metadata	22
3.5.1	XML Representation Examples	23
3.5.1.1	XMP Examples in RDF	23

3.5.1.2	Simple Example of XMP Metadata in XML . . . . .	23
3.5.2	Creation of Instance IDs. . . . .	24
3.5.3	Metadata in Compound Documents. . . . .	25
3.5.4	External Storage of Metadata . . . . .	26
3.6	RDF Features Not Supported in XMP . . . . .	26
3.7	Limitations of RDF . . . . .	27
3.8	XML Packets . . . . .	27
3.8.1	Usage Hints . . . . .	31
<b>Chapter 4</b>	<b>XMP Schemas . . . . .</b>	<b>.33</b>
4.1	Introduction. . . . .	33
4.1.1	Property Value Type Representation . . . . .	33
4.2	XMP Schema Definitions. . . . .	34
4.2.1	XMP Core Schema . . . . .	34
4.2.2	XMP Media Management Schema . . . . .	35
4.2.2.1	Versions . . . . .	36
4.2.2.2	Renditions . . . . .	36
4.2.3	XMP Support Schema. . . . .	37
4.2.4	XMP Basic Job Ticket Schema . . . . .	38
4.2.5	XMP Rights Management Schema . . . . .	38
4.3	Property Value Types. . . . .	39
4.4	XMP Vocabularies . . . . .	43
<b>Chapter 5</b>	<b>XMP Property Commentary . . . . .</b>	<b>.47</b>
5.1	XMP Properties . . . . .	47
<b>Chapter 6</b>	<b>XMP Extensibility . . . . .</b>	<b>.55</b>
6.1	Making Custom Schema . . . . .	55
6.2	New Versions of Existing Schemas . . . . .	56
<b>Chapter 7</b>	<b>Application Integration Guidelines . . . . .</b>	<b>.59</b>
7.1	Supporting XMP Metadata . . . . .	59
7.1.1	Requirements . . . . .	59
7.1.1.1	Aliasing . . . . .	60
7.1.1.2	Property Categories . . . . .	60
7.1.1.3	XAP:Advisory Example . . . . .	61



7.2	Metadata Actions For Specific Application Operations . . . . .	61
7.2.1	General Comment on Schemas . . . . .	61
7.2.1.1	New Document . . . . .	62
7.2.1.2	Save and Save-As. . . . .	62
7.2.1.3	Versions . . . . .	63
7.2.1.4	Renditions . . . . .	63
7.2.1.5	Document <i>Open</i> Time . . . . .	63
7.2.2	Media Management System Actions . . . . .	64
7.2.3	Document Embedding and Metadata Preservation . . . . .	64
7.2.3.1	Placed-Image Metadata . . . . .	65
7.2.3.2	Full Unaltered Copy Embedding. . . . .	65
7.2.3.3	Subset or New Rendition Embedding . . . . .	66
7.2.3.4	Small Subset Embedding . . . . .	67
7.2.3.5	Embedding of a Document Which Had No Metadata . . . . .	67
<b>Appendix A PDF and Dublin Core Schema . . . . .</b>		<b>.69</b>
A.1	Adobe PDF Schema . . . . .	69
A.2	Dublin Core Schema . . . . .	70
<b>Appendix B Proposed Media-Type Schemas . . . . .</b>		<b>.71</b>
B.1	XMP Media-Type Schemas . . . . .	71
B.2	Property Value Types. . . . .	74
B.3	Vocabulary for Media-Specific Schema . . . . .	76





# List of Tables

Table 4.1	XMP Core Schema . . . . .	34
Table 4.2	XMP Media Management Schema . . . . .	35
Table 4.3	XMP Support Schema . . . . .	37
Table 4.4	XMP Basic Job Ticket Schema . . . . .	38
Table 4.5	XMP Rights Management Schema . . . . .	38
Table 4.6	Basic Value Types . . . . .	39
Table 4.7	Media Management Value Types . . . . .	40
Table 4.8	Adobe Support Metadata Value Types . . . . .	43
Table 4.9	Basic Job/Workflow Value Types . . . . .	43
Table 4.10	XMP Vocabularies . . . . .	44
Table 5.1	XMP Core Schema Metadata Properties . . . . .	47
Table 5.2	Adobe Media Management Metadata Properties . . . . .	50
Table 5.3	Adobe Support Schema Metadata Properties . . . . .	53
Table 5.4	XMP Basic Job Properties . . . . .	54
Table A.1	Adobe PDF Schema . . . . .	69
Table A.2	Dublin Core Schema . . . . .	70
Table B.1	XMP Graphics Schema . . . . .	71
Table B.2	XMP Graphics: Image Schema . . . . .	71
Table B.3	XMP Dynamic Media Schema . . . . .	72
Table B.4	XMP Dynamic Media: Video Schema . . . . .	72
Table B.5	XMP Dynamic Media: Audio Schema . . . . .	72
Table B.6	XMP Text Schema . . . . .	73
Table B.7	XMP Text: Paged-Text Schema . . . . .	73
Table B.8	Basic Value Types for Media-Specific Schemas . . . . .	74
Table B.9	XMP Vocabularies . . . . .	76





# 1

## Preface

### 1.1 About This Document

This document describes XMP™ — the Extensible Metadata Platform.

This section contains information about this document, including how it is organized, conventions used in the document, and where to go for additional information.

### 1.2 Audience

The primary audience for this document is developers of applications that will generate, process, or manage files containing XMP metadata.

### 1.3 Assumptions

This document assumes that you are either familiar with XML and RDF, or that you will reference those specifications and related standards while reading this document.

### 1.4 How This Document Is Organized

In addition to this preface, this document consists of the following chapters and appendices:

#### **Chapter 2: XMP – Extensible Metadata Platform**

Explains XMP metadata and the model for how it is used, and provides a conceptual model of how metadata is created and managed. It explains the background and scope of the XMP model, and defines basic terms and concepts. It also describes how new schema may be defined to meet needs beyond what is supported by the existing model. A number of code samples are shown to illustrate how XMP metadata is represented.

#### **Chapter 3: XMP RDF Data Interchange Format**

Describes how XMP uses the RDF format for data representation and how to embed metadata using XML Packets to make it easy for applications to locate metadata in application files.

#### **Chapter 4: XMP Schemas**

Specifies all currently supported schema for XMP core metadata. It also specifies the value types used for all properties, and contains the [XMP Vocabularies](#), which specify the set of allowed values for each property. The schema tables also specify the category of each property, and list aliases to properties in other schema.

**Chapter 5: XMP Property Commentary**

Discusses the XMP properties and describes their nature and expected use. Also, suggestions are provided for how certain properties should be handled when documents are embedded in other documents.

**Chapter 6: XMP Extensibility**

Explains the extensibility features of XMP, including how to extend schemas, add new schemas, and add private data.

**Chapter 7: Application Integration Guidelines**

Describes what applications must do to implement support for metadata, and what actions they have to perform for common application operations. It explains what media management systems need to do to support XMP metadata, and how to support the embedding of one document in another.

**Appendix A: PDF and Dublin Core Schema**

Specifies the PDF and Dublin Core schemas, and specifies which properties are aliased to properties in the core XMP schemas.

**Appendix B: Proposed Media-Type Schemas**

The schemas in this appendix are proposals for basic media-type (content specific) metadata.

## 1.5 Conventions used in this Document

The following type styles are used for specific types of text:

Typeface Style	Used for:
Sans serif regular	XMP property names. For example: xap:CreationDate
Monospaced Regular	All RDF (XML) code

## 1.6 Where to Go for More Information

The following is a list of Internet standards on which XMP Metadata is based:

Dublin Core Metadata Initiative:

<http://purl.org/DC/>

Extensible Markup Language (XML):

<http://www.w3.org/XML/>

IETF Standard for Language element values (RFC 1766):

<http://www.ietf.org/rfc/rfc1766.txt?number=1766>

Internet Engineering Task Force (IETF):

<http://www.ietf.org/>

ISO 639 Standard for Language Codes:

<http://www.loc.gov/standards/iso639-2/>

ISO 3166 Standard for Country Codes:

<http://www.din.de/gremien/nas/nabd/iso3166ma/>

Naming and Addressing: URIs, URLs, etc.:

<http://www.w3.org/Addressing/>

Resource Description Framework (RDF):

<http://www.w3.org/RDF/>

Resource Description Framework (RDF) Model and Syntax Specification:

<http://www.w3.org/TR/REC-rdf-syntax/>

Unicode:

<http://www.unicode.org>

Web Distributed Authoring and Versioning (WebDAV):

<http://www.webdav.org/>

XML Namespaces:

<http://www.w3.org/TR/REC-xml-names/>

XML Path Language (XPath):

<http://www.w3.org/Tr/xpath>



# 2

## XMP – Extensible Metadata Platform

### 2.1 Introduction

This document describes XMP™ – the Extensible Metadata Platform, which provides a standardized method for the creation, processing and interchange of metadata.

**NOTE:** Many namespaces, keywords, and related names in this document are prefaced with the string “XAP”, which was an early internal code name for XMP metadata. Because the Acrobat 5.0 product shipped using those names and keywords, they were retained for compatibility purposes.

Metadata is becoming increasingly important in the production, management, and publication of multimedia documents. Documents containing metadata can greatly increase the utility of managed assets in collaborative production workflows.

An example of metadata used in a workflow environment would be an image file that contains metadata such as the image’s working title, image description, thumbnail image, and intellectual property rights data. This metadata about the file contents would enable workflow users, as well as asset management systems, to use resources much more effectively and to streamline workflow processes. Without the metadata, it might be difficult to associate images with their file names, to locate image captions, or to determine copyright clearance to use an image. While these operations are sometimes handled by various applications using their own metadata format, it is the interchange of the metadata, for use in multiple workflows, that is not so easy.

XMP standardizes the definition and creation of metadata, and defines an extensible representation that allows applications and tools to access and understand metadata about documents that they manipulate. XMP metadata defines a core set of metadata properties that are relevant for a wide range of applications including all of Adobe’s authoring and publishing products, as well as for applications from a wide variety of vendors.

XMP also provides a file embedding mechanism, called a XML Packet, that allows applications to easily locate metadata in files by simple scanning, rather than needing to parse a specific application’s file format. This feature makes the metadata more easily accessible, and aids document interchange and asset management.

XMP includes the following key features:

- It provides a lightweight distributed implementation, thus avoiding a single monolithic implementation that all applications must obey.
- It accommodates a wide variety of workflows and tool environments.
- It supports extension by addition of standard schemas, by addition of private or application-specific schemas, and by updating schemas to new versions.

- It supports aliasing between multiple schemas, so it is efficient in the storage of, and access to, items that are logically equivalent (for example, PDF's [pdf:Author](#) and Dublin Core's [dc:creator](#) properties).
- It is localizable and supports Unicode and other standard international text encodings.

XMP metadata can be encoded as an XML formatted string using the W3C standard Resource Description Framework (RDF), which is described in [Chapter 3, “XMP RDF Data Interchange Format.”](#) Ideally, the metadata string is embedded in an application data file and hence appears as part of the resource data stream itself. This has the advantage that the metadata stays with the application data file, even if the file is moved.

If embedding is not possible (typically because of file formats that do not accommodate extensibility), the metadata stream can be stored in a separate file that is associated by convention with the application data file. This works universally, but has the disadvantage that the metadata can be lost in a processing step if the metadata file is not kept together with the application data file.

In addition, a media management system, if present, can store the metadata in its own internal database. Some environments may include database systems in which metadata is to be stored.

## 2.2 Background

The following is an excerpt from the introduction section in the [RDF specification](#) that explains the value of metadata, and also the role played by the RDF format (which is described in more detail in [Chapter 3, “XMP RDF Data Interchange Format”](#)). While it presents a very Web-centric view of metadata, it also applies to other domains of usage as well, including print publishing.

The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, this data is not machine-understandable. It is very hard to automate anything on the Web, and because of the volume of information the Web contains, it is not possible to manage it manually. The solution proposed here is to use metadata to describe the data contained on the Web. Metadata is “data about data;” (for example, a library catalog is metadata, since it describes publications) or specifically in the context of this specification “data describing Web resources.” The distinction between “data” and “metadata;” is not an absolute one; it is a distinction created primarily by a particular application, and many times the same resource will be interpreted in both ways simultaneously.

Resource Description Framework (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources. RDF can be used in a variety of application areas; for example: in resource discovery to provide better search engine capabilities, in cataloging for describing the content and content relationships available at a particular Web site, page, or digital library, by intelligent software agents to facilitate knowledge sharing and exchange, in content rating, in describing collections of pages that

represent a single logical “document”, for describing intellectual property rights of Web pages, and for expressing the privacy preferences of a user as well as the privacy policies of a Web site.

XMP was created by Adobe to support the needs of digital asset management as part of Adobe’s Network Publishing initiative. XMP is concerned with describing any digital asset (resource), with or without an explicit Internet presence. This contrasts with RDF’s Web-centric view. As a framework for a specific application domain, XMP is less concerned than RDF with absolute generality and expressiveness.

These differences are most visible in portions of RDF that are not supported by XMP and by differing notions of resource identification. XMP does not support portions of RDF, such as reification, that add significant complexity but not significant value to practical asset management. Resource IDs in XMP are generally a form of GUID that can be generated locally, as opposed to the bias towards URLs in RDF.

## 2.3 Scope of XMP

XMP consists of the following components:

- A XMP Metadata Model (Chapter 2.4, “Model and Terminology”)
- An XML-based interchange representation, based on the data model of the Internet standard *RDF*, for metadata (Chapter 3, “XMP RDF Data Interchange Format”).
- A set of *schemas* for XMP metadata (Chapter 4, “XMP Schemas”).
- A set of conventions and rules for extending the metadata schema beyond what is defined by the XMP schemas (Chapter 6, “XMP Extensibility”).
- A set of conventions and rules for using the XMP metadata. (See Chapter 5, “XMP Property Commentary” and Chapter 7, “Application Integration Guidelines”).
- Guidelines and suggestions on how to integrate support for XMP metadata into your application (Chapter 7, “Application Integration Guidelines”).

There are a number of areas, listed below, that are outside the scope of XMP, and should be under the control of the applications and tools that support XMP metadata. In some of these cases, even though the item is outside the scope, some recommendations are made in this document. In those cases, a reference to the recommendations follows the bullet item in the list below.

The following are outside the scope of XMP:

- What metadata each specific application sets (XMP recommendations: Chapter 2, “XMP – Extensible Metadata Platform,” and Chapter 7, “Application Integration Guidelines”).
- The user interface to metadata, if any.
- The operation of any media management systems (XMP recommendations: see Chapter 7, “Application Integration Guidelines”).
- Which schemas beyond those defined by XMP are defined and present.

- Validity and consistency checking on metadata properties.
- Requiring that users set or edit metadata.

XMP schemas define a set of possible metadata. Not all XMP metadata will be relevant to all documents and it is expected that only the relevant subset of metadata properties would be present in a particular document. Some XMP properties are defined to provide a standard method to store certain kinds of metadata. If that metadata is defined and relevant, the XMP property should be used to store it.

Following the XMP schema and guidelines presented in this document cannot guarantee the integrity of metadata or metadata flow; that integrity must be accomplished and maintained by a specific set of applications and tools. An application's support for XMP refers to its ability to preserve and generate XMP format metadata, to give the user access to the metadata, and to support extension capabilities.

## 2.4 Model and Terminology

This section introduces the model and concepts that underlie XMP.

*Metadata* is information about the data that is contained in a file. The metadata may include information that is redundant with the content, that is a direct function of the content, and that is independent of the content. From a practical standpoint, defining metadata that is redundant with document content is avoided unless that metadata is of general interest to a variety of users and it is constant in size (that is, it doesn't grow in size with the document size). Metadata that is redundant with, or a function of document content, is called *internal metadata*. Metadata that supplements document content is called *external metadata*.

A *document* is any media object that a user might consider such as an image, text document, compound document (containing multiple text and image components), audio, video, etc. A document can be thought of as an abstraction. The title may change; the file in which it is stored may be renamed; one version of it may be in a database, another stored online; it may be rendered at various resolution to screen or various printers; a copy may be sent to someone by e-mail; and so on, but it remains the same document.

A *compound* document or *aggregate document* is one that has been constructed by incorporating other documents into a single document, generally with additional local content. This may be done by physical inclusion, which is often the case for publishing applications. Or it may be done by reference, which is often the case for multimedia applications such as video editors. The other documents are *contained documents* in the compound document. An *aggregating application* is one that creates compound documents.

*Document-level metadata* is information about a document as a whole. *Fine-grained metadata* is information about individual small elements of a document such as a paragraph or short sequence of words. XMP is concerned primarily with document-level metadata. Some of the properties defined by XMP can be used for fine-grained metadata, but the overall structure defined by XMP is not really appropriate for fine-grained metadata.



A *metadata schema* is a set of specific metadata property definitions.

A *version* of a document is analogous to an edition of a book, one of a series of revised forms of the document. Versions are explicitly identified by being written to a filestore and being flagged as a new version of some previous version of a document. Each version is distinguished by having a version identifier. Change history information is associated with a version that indicates changes from its predecessor version.

A *rendition* of a document is a variation of the document derived from a particular version by changing the format or content in some well defined way. Common examples include the creation of a thumbnail or the generation of the same image in a different format or resolution. Each rendition is distinguished by having a name that is called the *rendition class name*. There may be many different renditions. Each kind of rendition has a different rendition class name, for example, *thumbnail*, *low-res*, or *French*.

An *instance* of a document is a snapshot of a version and rendition, a particular value of the document written to a filestore. Correcting a misspelled word would generally not create a new version, it would create a new instance. The XMP model supports multiple versions of a document, multiple renditions of each version, and multiple instances of each rendition.

A *resource* is the object that stores a particular instance of a document and with which a set of metadata is associated. You can think of resources as the storage container (usually a file) for versions and renditions of documents. Resources could also store other kinds of objects.

The XMP schema defines *metadata* that is about a resource. The metadata consists of a number of properties; each *property* is associated with a resource and makes some statement about it. The statement has a *property name* and a *value* and has the form “the <property name> of <resource> is <value>.” For example, the “author” of a book titled *The Programmer’s New Age Cookbook* might be “John Doe.”

A *value* can be a simple value:

- a boolean value (*TRUE* or *FALSE*)
- a text string
- date
- integer
- a real number (with an optional binary representation specified)
- a value chosen from a vocabulary of possible values (a *choice*)
- a value chosen from one of several vocabularies (an extensible *choice* or *xchoice*)

or it can be a structured value:

- an ordered sequence (*seq*) of values
- an unordered sequence (*bag*) of values
- a set of alternative values (*alt*)
- a nested structure with named fields each of which is itself a property.

Most simple values end up being text. There may be conventions or restrictions on what values are legal for a particular property.

Text strings and choice values can include a *vocabulary qualifier* that names a vocabulary from which the string is chosen.

A *vocabulary* is a set of possible values with some (informal) information about their semantics. A property value can be restricted to have values only from a vocabulary (or set of vocabularies). This is called a *closed vocabulary*. The property may also allow other values in addition to those listed in a vocabulary. This is called an *open vocabulary*. (See [Section 4.4](#) for more information on XMP vocabularies.)

A sequence of values (*seq*) is simply a list of zero or more values. The order of the list has significance. A *bag* of values is similar to a sequence but the order does not have significance. A set of tagged alternative values (*alt*) is a list of one or more alternative values where one can be chosen based on some criterion. The most common criterion is language, used to provide localization of textual properties.

The properties are grouped into *schemas*, each of which consists of a set of properties, a schema name, and a schema namespace prefix. The *schema name* serves to uniquely identify the schema, and, although it looks like a URL, it is simply a unique string. The *schema namespace prefix* is a short abbreviation for the full schema name. The schema namespace prefixes used here are not formal. Following the rules of XML namespaces, the schema namespace prefix is simply shorthand for the schema name and is local to the scope of the `xmlns` attribute that declares it.

Properties in the same or different schemas can be *aliased*, which means that they represent the same value. Aliased properties imply that they should always have the same value. The types of the two aliased properties must be the same, and changing one value means that all aliased values should be changed. The alias relationship is transitive. The reference to the aliased property uses XPath syntax. (see <http://www.w3.org/TR/xpath>)

Schema properties are categorized to help ensure that processing gives predictable results. These categories tell the authoring application what to do when opening and closing files, and how to changes to property values made by other applications.

- **Internal:** metadata which is a reflection of the information contained within the resource. External modifications to internal properties should be ignored by the authoring application. The appropriate value for internal properties is written on output. An example would be `xap:ModifyDate`.
- **External:** metadata consisting largely of annotation information. External modifications should be displayed by the authoring application but are not acted upon. Unless changed by the user, external properties are preserved on output. An example would be `xap:Author`.
- **Relational:** a subset of internal metadata that pertains to the relationship of this resource with outside systems. Examples might include management markers or print resolution. External

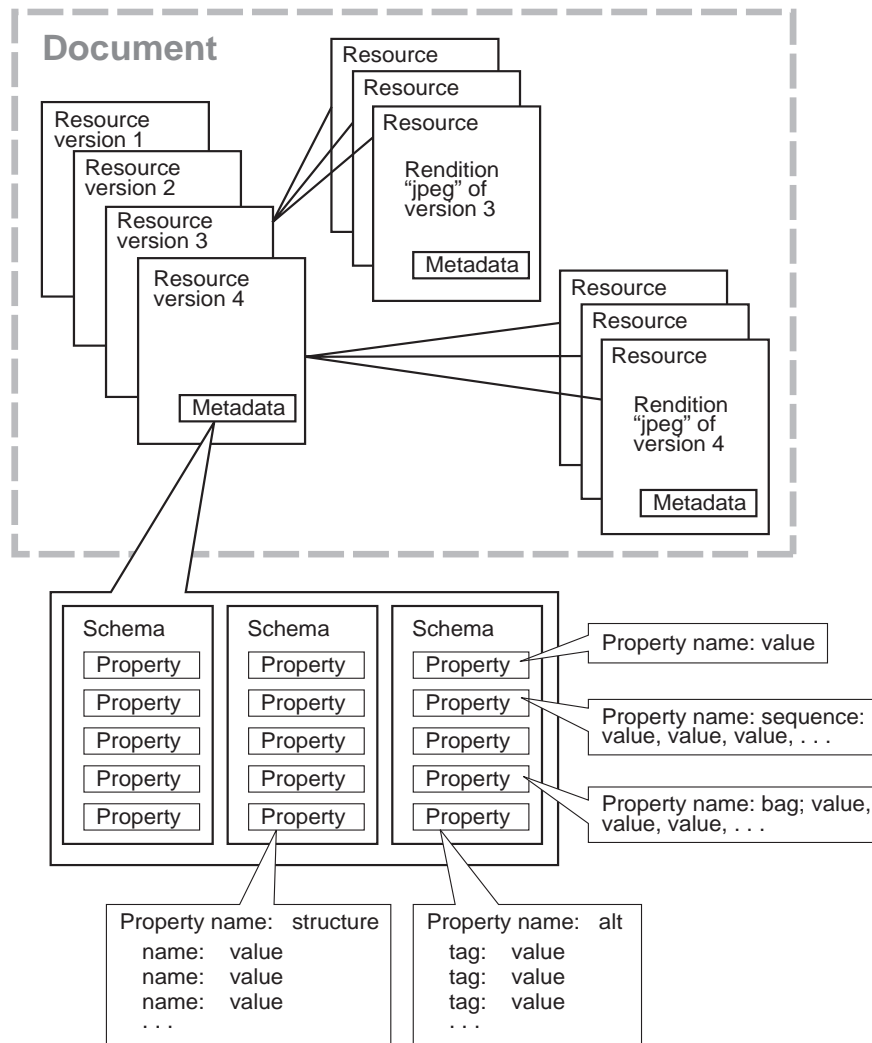
modifications to these properties should be resolved against the internal information in the document.

Properties from a schema typically have values, or they may instead be absent in the metadata of a given resource. Properties are absent until they are given a value for the first time.

Properties may also be deleted. The presence or absence of a property is visible to application programs. Every present RDF property has a value, even if it is just the empty string. Note that a present property with an empty string value is different from an absent property.

XMP metadata can be stored within a resource, in a separate resource related by convention to a resource, and/or in a media management system.

Figure 2.1 shows the relationship between a document and the components of the metadata. It reflects the model that a document is an abstraction, and the implementation of it may consist of multiple renditions, versions, and renderings, represented by a number of resources or files, each containing metadata. It does not attempt to illustrate how the metadata or versions are managed (see [Chapter 7, “Application Integration Guidelines](#), for more information on that topic).

**FIGURE 2.1 Documents and Metadata Component Relationships**

In summary:

- XMP metadata is associated with a resource, which stores a particular version and rendition of a document.
- XMP metadata is organized into schemas each of which has a schema name, a schema namespace prefix and a set of properties.
- Each property has a property name and value and says something about the resource with which it is associated.

- Property values can be text, dates, booleans, integer or real numbers, values chosen from one or more vocabularies, sequences of values, bags of values, a set of alternative values, or nested structure containing more properties.

## 2.5 Granularity of XMP Metadata Associations

XMP metadata can be associated with any structure or substructure from servers and file systems, to files and database entries, to pages, lines, characters, styles and other finer grained structures stored within a file. Keep in mind the practical issues which will argue against using XMP for very fine-grained metadata:

- XMP works best if the document containing the metadata has an identity. Assigning and tracking identities for individual words and characters is largely impractical.
- XMP includes a certain amount of overhead given its XML representation (tag names, angle brackets, timestamps, etc.). Thus, storage overhead can be considerable if XMP is associated with very small structures.



# 3

## XMP RDF Data Interchange Format

### 3.1 Introduction

This chapter describes the Web-standard technology Resource Description Framework (RDFMS 1.0) which is used as the data interchange format. This chapter also includes a specification of XML Packets, which is a method for embedding metadata packets in application files so that they can be easily located by simple scanning software.

XMP uses a profile (subset) of the RDF specification developed by W3C. Use of unsupported features of RDF may result in incorrect operation of XMP-enabled applications. See Section 3.6, “RDF Features Not Supported in XMP” for more details.

**NOTE:** This chapter assumes that you are familiar with basic RDF concepts. For more information, see <http://www.w3.org/TR/REC-rdf-syntax/>.

### 3.2 Background

XMP was designed with the goal of establishing a standard for the representation and interchange of metadata between applications. Rather than invent a new standard, it made sense to adopt the RDF standard, and benefit from the documentation, tools, and shared implementation experience that come with an open Internet standard. The RDF syntax is based on XML.

Most documentation about RDF uses the term *resource* as the thing that is being described by the metadata. *Resource* is used here in the same sense as it is used on the Web: it is the abstraction that contains information; it may or may not be a file.

### 3.3 RDF Data Model

The following sections explain the RDF data model, and how XMP uses the RDF model to support its goals.

There are three basic object types: Resources, such as documents and media files or literal values, Properties, such as Title and Author, and Statements, which relate Properties to Resources. All Statements in RDF can be expressed as triples: {Subject, Predicate, Object}. For simple name/value pairs, the Predicate is the name, the Object is the value, and the Subject is the document or media file that the name/value pair describes. RDF represents these relationships as directed graphs, where the Subject and Object are nodes, and the Predicate is a named directed arc from the Subject to the Object.

Consider as a simple example the sentence:

*Jane Doe is the creator of the resource <http://www.thedoefamily.org/home/jane>.*

This sentence has the following parts:

Subject (Resource):	<a href="http://www.thedoefamily.org/home/jane">http://www.thedoefamily.org/home/jane</a>
Predicate (Property):	Creator
Object (literal):	Jane Doe

In RDF diagrams, the nodes (drawn as ovals) represent resources and arcs represent named properties. Nodes that represent string literals will be drawn as rectangles. The sentence above would thus be diagrammed as shown in Figure 3.1.

**FIGURE 3.1** Simple node and arc diagram



## 3.4 How XMP Uses the RDF Data Model

This section describes how XMP uses the RDF data model. It explains the basic model, discusses how schemas are named, shows how to provide localized values for textual properties, and illustrates various features with sample code. The issues of extensibility and aliasing are also explained.

### 3.4.1 Description Object

A Description object is an RDF element that contains statements about a resource. Properties from various schemas are attached to the Description object. In other words, if you took all the statements in a model, and factored them by common subjects (all about the same resource), you would have a set of top-level Description objects, one for each unique subject resource.

Metadata can be associated with resources either internally or externally. *Internal association* usually means that the actual Description object is embedded in the data of the resource somewhere. *External association* usually means that the Description object refers to the resource with some kind of pointer, such as a URL. External associations can also be managed by a database application, which can maintain statement triples directly. All of these methods are supported by this data model.



**NOTE:** Internal and external associations are RDF model concepts, not to be confused with the internal and external categories for properties defined in section 2.4, “Model and Terminology.”

### 3.4.2 Repeated Properties

The XMP subset of the RDF data model includes the container mechanism (described in [section 3](#) of the RDF specification) for modeling properties with multiple values. The *syntax* also allows properties to be repeated, but this practice is discouraged for XMP in order to avoid ambiguity between the two cases of a single property with multiple values, and multiple instances of the same property with single values. Avoiding repeated values also makes it easier to implement XMP for protocols like WebDAV (an IETF Proposed Standard, published as RFC 2518) which forbid repeating properties. Repeated properties in serialized XMP metadata are mapped into an equivalent structured container type, which is described by the schema. When no schema description is available, the default is a sequence type.

### 3.4.3 Schemas and Namespaces

Metadata can usually be organized into related groups of properties. These groups are relevant only for particular types of documents, or perhaps only for certain stages of a workflow. These groups are implemented by defining schemas for them. Each schema defines a new namespace.

A schema is a set of specific metadata property definitions; it includes, but is not limited to:

- A vocabulary of elements (property names)
- The legal values for elements (constraints on property values)
- The relationship between properties, if any (there’s none if they are simple and flat)

The vocabulary of properties is defined within a *namespace*. A namespace helps to differentiate between property names that are the same, but which have different meanings depending on the schema. For example, in one schema, *Creator* may mean the person who created a resource. In another schema, *Creator* may mean the application used to create a resource. The schemas may have been defined independently, and each has a legitimate claim on *Creator* as a property name. Confusion is avoided by qualifying the property names with a schema-specific namespace.

In XMP metadata, each schema is named. The URI of the schema is considered to be the name of the namespace.

Some properties have values that contain more than one component (structured values). The names of the components use tags from an XML namespace associated with the structure. An example of such a property would be the `xapG:NaturalDimensions` property. Each property requiring specification of dimensions needs a width, a height, and an indication of the units used for the numeric values of width and height. Each property describes a different

interpretation of dimension: physical dimensions, pixel sample dimensions, desired rendering dimensions, etc., and all use tags from their own namespace. This namespace is just a simple factoring of the vocabulary into useful, independent modules. This is more for the convenience of programmers and human readers of metadata specifications than for any other reason.

Let's suppose we have an image resource "myPhoto.gif." We want to describe the dimensions in pixels, as well as the natural presentation dimensions (desired size in inches when displayed at 100% scale). These are two different properties which share the same value type.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="myPhoto.gif"
    xmlns:xapG="http://ns.adobe.com/xap/1.0/g/"
    xmlns:stDims="http://ns.adobe.com/xap/1.0/sType/Dimensions#">
    <xapG:NaturalDimensions rdf:parseType='Resource'>
      <stDims:w>4</stDims:w>
      <stDims:h>3</stDims:h>
      <stDims:unit>inches</stDims:unit>
    </xapG:NaturalDimensions>
  </rdf:Description>

  <rdf:Description about="myPhoto.gif"
    xmlns:xapGImg="http://ns.adobe.com/xap/1.0/g/img/"
    xmlns:stDims="http://ns.adobe.com/xap/1.0/sType/Dimensions#">
    <xapGImg:Dimensions rdf:parseType='Resource'>
      <stDims:w>640</stDims:w>
      <stDims:h>480</stDims:h>
      <stDims:unit>pixels</stDims:unit>
    </xapGImg:Dimensions>
  </rdf:Description>
</rdf:RDF>
```

The first description uses the schema for XMP Graphics (<http://ns.adobe.com/xap/1.0/g/>). The property is called `xapG:NaturalDimensions`, and its value is a nested description (which is what `rdf:parseType='Resource'` means). The nested description uses the namespace "<http://ns.adobe.com/xap/1.0/sType/Dimensions#>."

The second description uses the schema for **XMP Graphics: Image Schema**:

```
(http://ns.adobe.com/xap/1.0/g/img/)
```

The property is called `xapGImg:Dimensions`, and its value is also a nested description. The nested description also uses the namespace <http://ns.adobe.com/xap/1.0/sType/Dimensions#>.

### 3.4.4 Localized Property Values

Support for multiple languages in text is relatively straightforward in XMP. Because it is based on RDF and XML, the default encoding for text is UTF-8 with support for all Unicode characters. Multibyte Unicode encodings may also be used.

In addition, individual properties may have multiple localized values through use of the RDF alternative container. This allows the metadata to contain multiple values for a property, with one value being selected for use at a time. In the case of localized text values, this selection uses an `xml:lang` attribute given with each value.

The example below shows the `dc:title` property (in isolation) with English, French, and Italian values:

```
<dc:title>
  <rdf:Alt>
    <rdf:li xml:lang="en">XMP - Extensible Metadata Platform</rdf:li>
    <rdf:li xml:lang="fr">XMP - Une Plateforme Extensible pour les Métadonnées</rdf:li>
    <rdf:li xml:lang="it">XMP - Piattaforma Estendibile di Metadata</rdf:li>
  </rdf:Alt>
</dc:title>
```

The RDF defines the first element in the alternative container as the default. For example, on a German system, the English title would be displayed because there is no German title. XMP defines the ‘`x-default`’ language code as a means to explicitly denote a default value. The `x-default` item should be the first one so that general RDF processors unaware of XMP will also use it. Having an `x-default` item also considerably simplifies changing the default. With it you only have to change that item’s value, you do not have to reorder the entire container.

The recommended practice in XMP is to define the default explicitly, in addition to the other localized values. Adding a French default to the above example:

```
<dc:title>
  <rdf:Alt>
    <rdf:li xml:lang="x-default">XMP - Une Plateforme Extensible pour les Métadonnées</rdf:li>
    <rdf:li xml:lang="en">XMP - Extensible Metadata Platform</rdf:li>
    <rdf:li xml:lang="fr">XMP - Une Plateforme Extensible pour les Métadonnées</rdf:li>
    <rdf:li xml:lang="it">XMP - Piattaforma Estendibile di Metadata</rdf:li> Metadaten</rdf:li>
  </rdf:Alt>
</dc:title>
```

### 3.4.5 Extensibility

XMP Metadata schemas may be added by the procedure described in [Chapter 6, “XMP Extensibility,”](#) which also shows examples of extensions.

The data model makes no distinction between standard and non-standard schemas. All properties are associated with a namespace URI, and that URI is the name of the schema.

The version of a schema is considered to be a part of the schema name. Therefore, `foo:/schema/1.0/` and `foo:/schema/2.0/` are considered to be two completely different schemas with no relationship between them. This leaves applications free to define the semantics for schema versions.

The data model does not require the presence of a schema in order to instantiate and explore statements.

### 3.4.6 Aliasing

One of the goals of XMP is to efficiently handle properties from different schemas which have the same logical meaning. This goal is achieved through the aliasing of properties.

Two properties should be aliased if and only if they can truly be considered identical substitutes. They must have the same value type and meaning. Aliasing literally creates multiple names for one logical value.

Aliases are typically defined by pairs with an alias property pointing to an actual property. For example, the alias property A is said to be *aliased to* the actual property B, or B is said to be *aliased by* A. Aliasing is transitive, if B is aliased to C, then A is also aliased to C. Collections of aliases should have a *base* member, which is the preferred form for serialized output.

Properties may be aliased across namespaces, for example xap:Authors is aliased to dc:creator. Aliases may only be created for top level properties, not between general components of structured properties. One exception is that the base of a collection of aliases may be the first or default item in a bag, sequence, or alternative. This allows simple schema to interoperate with more sophisticated schema. For example, pdf:Author is aliased to dc:creator/\*[1], the first item in the dc:creator container.

### 3.4.7 Resource Identification

One way in which the web-centricity of RDF shows is in the way resources are identified. About attributes in the outermost `rdf:Description` elements identify the resource being described, the RDF examples use URLs almost exclusively. This is natural and appropriate given RDF's mission "to describe the data contained on the Web." RDF also allows about or ID attributes on inner `rdf:Description` elements to define inline resources. We are not concerned with that here, in fact XMP does not support that aspect of RDF.

The use of URLs is not appropriate for XMP, which is oriented towards general digital asset management. The assets in many environments may not have URLs. Perhaps more serious, the URL/web-page model does not address the document/version/rendition model presented in [Chapter 2](#) of this document. Web pages do not have versions or renditions expressed in their URLs. The actual location of a resource may be constantly changing as it moves from one person to another in an office workflow. It may well change throughout the day as one person works in different locations.

Two important aspects of a resource ID are persistence and specificity. A persistent ID is one which is stable over time, allowing references to a resource to be resolved even though that resource may have changed in some regard. The URL for a web page has some aspects of persistence, the URL does not change if the content changes. The URL lacks persistence as an

abstract ID though, as evidenced by the number of broken links on the web due to file movement. A specific ID is one that differentiates between two different instances of a resource. For example between two renditions of the same version of a document, say high and low quality JPEG renditions of the latest company logo.

These two aspects are often at odds, gaining persistence often means losing specificity. Different queries about the same resource may prefer one aspect over another. Continuing the above example, preparing marketing materials may require location of the appropriate rendition of the company logo while changing the logo may require locating all uses of any version or rendition of the logo. Other uses of the metadata, such as in compound documents, may require very precise and unambiguous reference to an exact instance of a resource.

The recommended approach in XMP is to have multiple parts to the abstract identification of a resource, ranging from very persistent to very specific. Applications should generate a unique *instance ID* whenever saving a document to disk and use this ID in the about attributes of the `rdf:Description` elements. Details for the instance ID are given in [Section 3.5.2](#). In addition, the `DocumentID`, `VersionID`, and `RenditionClass` properties in the XMP Media Management schema should be set to provide more persistent identification. Other properties may also be used. For example the `ResourceID` property in the XMP Support schema would be used by an asset management system to record its own internal ID for a resource that is “checked out” from the system.

### 3.4.8 Normalization of Metadata

One of the primary advantages of publishing the XMP specification and basing XMP on RDF is to encourage and facilitate interchange and broader use of metadata. Given the relative immaturity of RDF usage overall and of emerging schema standards such as Dublin Core, different implementations will almost certainly create different RDF. Some of these differences may endure as legitimate differences in expression or locally constrained usage.

For example, the proposed guide for expressing Dublin Core in RDF:

<http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf/>

does not specify how to represent multiple authors. Should they be specified as repeated simple `dc:creator` properties, or as a bag or sequence container? Should localizable properties such as `dc:title` always be represented as alternative containers?

It would be a tremendous burden if each application dealing with metadata had to specifically allow for all reasonable variations of metadata that it encounters. A better approach is to normalize the metadata on input, so that the application can be written to deal with one canonical representation. Where possible, the metadata should be returned to its original representation when saved again. In addition, some special case simplification of the serialized RDF may help interchange.

Ideally this normalization is defined by the schema. The following normalizations are recommended:

- When a property is represented by start and end tags, e.g. “<prop>value</prop>”, whitespace at the start and end of the value should be removed. If the value consists of nothing but whitespace, it should be reduced to a single blank (U+0020) character.
- When a property is represented as an attribute, the value is the entire quoted attribute value including all whitespace.
- Properties defined as sequences or bags may be input as repeated simple properties and normalized to a sequence or bag according to the schema. The degenerate case of a single simple property where a bag or sequence is expected should, of course, be accepted and normalized.
- Repeated properties in the input should be normalized to a sequence container if there is no schema.
- Bags and sequences with just one element may be output as a single simple property if the schema does not specify otherwise.
- Localizable properties with only one localization (value) should be accepted as a simple property. This should be normalized to an alternative container with one item having the ‘x-default’ language.
- Localizable properties with just an x-default value may be output as a simple property if the schema does not say otherwise.

### 3.5 Representation and Storage of Metadata

XMP metadata uses the XML serialization of RDF as the standard for metadata interchange. In addition, the XML data can be embedded into any kind of document using the XML Packet format described in [Section 3.8](#), “XML Packets.”

The use of XML packets is encouraged for all document formats, even native XML. This allows software such as generic workflow systems to locate and extract metadata without knowing the specific document format.

Use of an `x:xapmeta` element is suggested as a means to simplify locating XMP metadata in general XML streams. It has no meaning other than to focus attention. It should be the outermost XML element in the serialized XMP data. Any additional non-XMP XML should be placed outside of the `x:xapmeta` element. The format is:

```
<x:xapmeta xmlns:x='adobe:ns:meta/' >
  ...the serialized XMP metadata
</x:xapmeta>
```

The `xapmeta` element may have any number of attributes, in any order. All unrecognized attributes should be ignored, and there are no required attributes. The only defined attribute at present is `x:xaptk`. This is written by the XMP toolkit, the value is the version of the toolkit.

Remember that the `rdf:RDF` element, and by inference the `x:xapmeta` element, can appear anywhere in an XML tree. They need not be at the outermost levels.

### 3.5.1 XML Representation Examples

The following examples of XMP representation are written in XML, which should be familiar to anyone who has done any hand-coding of HTML.

#### 3.5.1.1 XMP Examples in RDF

Familiarity with [XML](#) and [RDF](#) would be helpful, but even if you are not familiar with those standards, you can get the flavor for how XMP would use RDF through these examples. It is also helpful to know about [XML namespaces](#).

One hint that might help: the XML namespace mechanism is invoked by defining an attribute called “xmlns”. The format is:

```
xmlns:foo="bar"
```

This means that any element or attribute that begins with the prefix “foo:” is part of the “bar” namespace. So, for example:

```
foo:Author="Bubba"
```

means the element or attribute *Author* is in the namespace called *bar*. The *foo* part can be any legal XML name (without colons), and we usually pick a short prefix since it could be used a lot. If two elements have the same name, but different namespaces, they are considered to be different elements in XML:

```
<EXAMPLE xmlns:foo="bar" xmlns:oof="rab" xmlns:boo="bar">
  <DIFFERENT foo:Author="Bubba" oof:Author="Not Bubba"/>
  <SAME foo:Author="Bubba" boo:Author="Bubba"/>
</EXAMPLE>
```

#### 3.5.1.2 Simple Example of XMP Metadata in XML

A first example shows some metadata for this document. The author is John Doe, and the title is “XMP – Extensible Metadata Platform” An RDF Description element groups the properties together. Here is the complete RDF:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
    <pdf:Author>John Doe</pdf:Author>
    <pdf:Title>XMP – Extensible Metadata Platform</pdf:Title>
  </rdf:Description>
</rdf:RDF>
```

The basic RDF syntax is used above, which is needed when you have complex metadata to express. However, in the simple case above, which is just two flat name/value pairs, there is an alternative syntax which is more concise:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/"
    pdf:Author="John Doe"
    pdf:Title="XMP – Extensible Metadata Platform"/>
</rdf:RDF>
```



Each property belongs to a schema, represented using the XML namespace notation. The formal URI for the namespace and the local shorthand prefix are given in the attributes beginning with “xmlns:”. For the PDF schema, the formal URI is “http://ns.adobe.com/pdf/1.3/” and the shorthand prefix is pdf.

**NOTE:** The formal part of the namespace is a URI, not a URL. There may or may not be an actual web page at the URI. In the case of Adobe namespaces, currently there is no corresponding web page.

Let’s say we wanted to add some XMP core metadata, such as the MIME type of this document (`xap:Format`), and the *language* it is in (`xap:Locale`).

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <rdf:Description about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/"
    pdf:Author="John Doe"
    pdf:Title="XMP - Extensible Metadata Platform"/>
  <rdf:Description about="" xmlns:xap="http://ns.adobe.com/xap/1.0/"
    xap:Format="application/pdf" xap:Locale="en"/>
</rdf:RDF>
```

By convention, XMP places the properties from each schema in separate `rdf:Description` elements. This is not a requirement, just a means to improve readability.

If the document has multiple authors we could use the `dc:creator` property with an RDF sequence container for its value:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator>
    <rdf:Bag>
      <rdf:li>John Doe</rdf:li>
      <rdf:li>Jane Smith</rdf:li>
    </rdf:Bag>
  </dc:creator>
</rdf:Description>
</rdf:RDF>
```

The `rdf:li` elements are meant to remind you of HTML `<li>` list items. An `rdf:Seq` is the name of the ordered collection container type. The important point here is that multiple values can be specified for a property, and we can say something about how those values are structured.

### 3.5.2 Creation of Instance IDs

The instance ID used in the `rdf:Description` element’s `about` attribute should be a GUID/UUID style ID. The basis of the ID is a 128-bit number that is guaranteed to be globally unique. This guarantee may not be absolute; it is sufficient that the probability of a collision is so enormously remote as to be effectively impossible. Although there are various common schemes for generating the unique 128-bit number, XMP does not require any specific



scheme. One approach is to use physical information such as a local ethernet address and a high resolution clock. Another would be to use a variety of locally unique and random data, then compute an MD5 hash value, which avoids privacy concerns about the use of ethernet addresses. It also allows for regeneration of the ID in some cases, for example if the MD5 hash is computed using the image contents for a resource that is a digital photograph.

Because the `rdf:Description` element's `about` attribute is the only identification of the resource from the RDF point of view it is useful if its value is formatted in a standard manner. This lets other RDF-aware software realize what kind of URI is used, in particular that it is not a URL. Unfortunately there is no formal W3C recommendation for URIs that are based on an abstract UUID. One proposal specifically addressing this was made several years ago, but it seems to have not progressed since then. It may be found at:

<http://www.globecom.net/ietf/draft/draft-kindel-uuid-uri-00.html>

Another proposal for “vendor-specific” URL schemes may become a standard. It may be found at:

<http://www.ietf.org/internet-drafts/draft-king-vnd-urlscheme-02.txt>.

Other IDs in XMP, notably the `xapMM:DocumentID` property, should be based on a GUID/UUID number. Since these IDs are meaningful only to software that is aware of XMP metadata, they may be formatted with more information – for example, a signature for the authoring application.

### 3.5.3 Metadata in Compound Documents

A number of common applications have a notion of compound document, created by incorporating other documents into a single document, generally with additional local content. This may be done by physical inclusion, which is often the case for publishing applications. Or it may be done by reference, which is often the case for multimedia applications such as video editors. Defining and understanding the metadata for compound documents presents a number of challenges for the applications themselves and other software such as asset management systems.

The essential difficulty is that the metadata should describe the compound document structure. It should be possible to determine the document containment hierarchy from just the metadata. Resource tracking or usage queries should be able to locate physically embedded instances. This should be robust, and degrade gracefully in the face of incomplete metadata for included documents.

The references from the compound document to the contributing documents should use the `xapMM:ContainedResources` and `ContributingResources` properties. These are quadruples consisting of the `xapMM:DocumentID`, `VersionID`, `RenditionClass`, and instance ID (`about` attribute) from the contributing document. No reference should be created if none of this information is available.

The metadata from each contributing document and for the compound document itself should be in separate XML packets, with of course a single `rdf:RDF` element in the packet surrounding the `rdf:Description` elements for that component. This makes it easy to avoid confusion if one or more of the contributing documents lacks a unique instance ID. (For example, by having a null string as the value of the `about` attribute!) The authoring application should place the packet for the compound document itself last in the stored document. This also aids in disambiguation where some of the metadata does not have an instance ID, or at least a locally unique “about” string.

**NOTE:** The heuristic of considering the last packet in a file to be the primary packet is also useful for some non-compound documents. For example, the PDF format allows edits to be stored through sequential appends to the file. The primary dictionary is always at the end of the file. In this case, a PDF document that has been modified and saved multiple times may have multiple XML packets, with the last being the most recent.

### 3.5.4 External Storage of Metadata

While there is no overt bias in XMP, it is suggested that metadata should generally be embedded with the primary content as XML packets. There are cases where this is not appropriate or possible, such as database storage models, extremes of primary content size, or due to format and access issues. Small content intended to be frequently transmitted over the Internet may not tolerate the overhead of embedded metadata. Archival systems for video and audio may not have any means to represent the metadata.

When the metadata is stored separately, the question arises of how to associate the metadata with the primary content. Applications should:

- Write external metadata as though it were embedded and then have the XML packets extracted and catenated by a post processor.
- Place the `ResourceRef` within the primary content so that format-savvy applications can make sure they have the right metadata.
- Place the string used in the `rdf:Description` “about” attributes within the primary content. This allows software that understands the file format to verify that they have the right metadata. Use of a unique instance ID for the `about` attribute value can improve the reliability.

## 3.6 RDF Features Not Supported in XMP

The following features of the [RDFMS 1.0](#) specification are not supported by XMP:

- The optional `rdf:RDF` element (`rdf:RDF` is required)
- Top-level containers (top-level must be `rdf:Description` or `typedNode` elements)
- The `rdf:ID` attribute (ignored on all `rdf:Description` or `propertyElt` elements)

- The `rdf:bagID` attribute (ignored)
- The `rdf:aboutEach` or `rdf:aboutEachPrefix` attributes (entire `rdf:Description` ignored)
- The `rdf:parseType='Literal'` attribute
- Reified statements (they are not generated in the model).

Furthermore, all `rdf:about` attributes must appear on top-level `rdf:Description` or top level typed Node elements only. In general the value of the `rdf:about` attribute should be a meaningful URI, the document ID, or the instance ID. This allows the metadata to remain meaningful when separated from the document or when this document is included within a compound document. The `rdf:about` value may be the empty string (" " or ""), indicating that the metadata is about the current document. The `rdf:about` values must all be consistent within a single `rdf:RDF` element. All non-empty `rdf:about` values must be the same URI.

**NOTE:** The restriction to consistent *about* values in a single `rdf:RDF` element is a current implementation restriction of XMP. The intent is to relax this restriction eventually.

Otherwise, all other features of RDFMS 1.0 are supported in XMP.

### 3.7 Limitations of RDF

RDF is not well suited for, or was not designed to handle, the following features:

- Inline binary objects (primarily due to XML syntax).
- Optimal (minimal) size of the representation.
- Knowledge representation, in the most general artificial intelligence sense.
- Validation against a DTD.

### 3.8 XML Packets

The XML Packet format was developed to enable simple scanners to find XML data embedded in files with formats that a simple scanner may not understand, such as Photoshop<sup>®</sup> or PDF files. The format uses a syntax that is as close to XML as possible to minimize the filtering burden on the simple scanner.

The XML Packet format was designed to accomplish the following:

- Support embedding in binary and text formats, including the various Unicode encodings.
- Deal with arbitrary positioning within a byte stream (so as not to rely on machine word boundaries, etc.).
- Enable multiple XML packets to be embedded in a single data file.

- Provide easy-to-scan markers for delimiting the XML packet. Such markers should be XML syntax-compatible to allow transmission to an XML parser without additional filtering.
- Enable in-place editing, including expansion, of metadata embedded in XML packets.

The procedure for creating a XML Packet is described in this section. The packet includes a header and trailer (see Figure 3.3). The header provides byte ordering information, and optional encoding information.

**NOTE:** Be aware that an XML packet might contain valid XML that is not necessarily XMP-compliant RDF. It is desirable to preserve such non-XMP XML if possible.

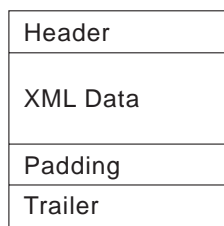
Here is a sketch of an XML packet showing the text of the header and trailer:

```
<?xpacket begin='■' id='W5M0MpCehiHzreSzNTczkc9d'?>
... 700 bytes of XML data text ...
... 500 bytes of XML whitespace as padding ...
<?xpacket end='w'?>
```

Where ‘■’ represents the Unicode “zero width non-breaking space character” (U+FEFF) used as a byte-order marker.

Figure 3.3 shows a schematic of an XML Packet.

**FIGURE 3.3 XML Packet Schematic**



The entire packet must conform to the Well-Formedness requirements of the XML specification, except for the lack of an XML declaration at its start. Also, there are additional constraints:

- Different packets may be in different character encodings.
- Packets must not nest.
- Data attributes in the header and trailer processing instructions are separated by exactly one blank (U+0020) character.

The following sections describe the parts of the packet illustrated in Figure 3.3.

### **Header**

The Header is an XML processing instruction:

```
<?xpacket ... ?>
```

The remainder of the processing instruction contains information about the packet. The syntax observes XML attribute syntax, which is production [41] Attribute, which is roughly:

```
Attribute ::= Name '=' AttValue
AttValue ::= '"' ([^<&"] | Reference)* '"' | "'" ([^<&' ] | Reference)* "'"
```

Note the use of either matching single or double quotes. Otherwise, a common error would be the use of the wrong quote character.

The header processing instruction must have two or more attributes. The first attribute must be the *begin* attribute, the second must be the *id* attribute. Other attributes may appear in any order, and unrecognized attributes should be ignored.

The description of each attribute follows.

#### *Attribute: begin*

This mandatory attribute is present only in the initial processing instruction, and indicates that it is the beginning of a new packet. The value of this attribute is the Unicode zero width non-breaking space character U+FEFF in the appropriate encoding (UTF-8, UTF-16, or UTF-32). This serves as a byte order marker, where the character is written in the natural order of the authoring/generating application (consistent with the byte order of the XML data encoding).

For backwards compatibility with earlier versions of the XML packet specification the value of this attribute may be the empty string, indicating an 8-bit encoding.

As described in the *Usage Hints* below, an XML Packet processor should be reading a single byte at a time until it has successfully interpreted a valid packet header. While processing the value of the *begin* attribute, if the processor detects the byte value '0xFE' followed by '0xFF,' it knows that the packet is big-endian order. If the processor detects the byte value '0xFF' followed by '0xFE,' it knows that the packet is little-endian order. If the processor detects the byte value '0xEF,' followed by '0xBB,' followed by '0xBF,' it knows this is UTF-8. If the attribute has no value (quote or double quote followed immediate by another quote or double quote), the byte order is irrelevant and the overall character encoding *must not* be any 16- or 32-bit Unicode encoding (that is, it must be UTF-8, US-ASCII, etc.).

#### *Attribute: id*

Next, there is a mandatory *id*. For all packets defined by this version of the syntax, the value of the *id* is the following string of 7-bit ASCII characters:

```
w5M0MpCehiHzreSzNTczkc9d
```

The value of the attribute must be encoded in the character encoding of the overall packet (see below). Thus, if the overall encoding is big-endian UTF-16, the *id* value should be converted from 7-bit ASCII to UTF-16 by inserting nulls.

*Attribute: bytes*

An optional `bytes` attribute may be present, specifying the total length of the packet in bytes. If the length extends beyond the end of the trailer processing instruction, the additional bytes must be properly encoded Unicode whitespace and are considered padding.

**NOTE:** Earlier versions of this specification recommended placement of the padding after the trailer processing instruction. This is now discouraged, the padding should come before the trailer. Placing the padding before the trailer and omitting the `bytes` attribute has always been valid, it is now the only recommended practice. Use of the `bytes` attribute is dangerous for XML packets embedded in text files. For example, moving a text file from a Macintosh or UNIX system to Windows typically causes all single byte line endings (CR or LF) to become 2 bytes (CRLF). This would invalidate the length given by the `bytes` attribute.

*Attribute: encoding*

The `id` attribute may be followed by an optional `encoding` attribute. It is identical to the `encoding` attribute in the XML declaration (see productions [23] and [80] in the [XML specification](#)). It specifies the character encoding of the entire packet. If this attribute is omitted, the encoding of the packet must be UTF-8.

The following is a simplified BNF syntax for the `encoding` attribute:

```
[A-Z a-z] ([A-Z a-z 0-9._] | -)*
```

**XML Data**

The bytes of the XML data are placed here. If the encoding is specified in the Header, the encoding of the XML data must match. If the encoding was omitted from the Header, the encoding of the XML data must be UTF-8.

You should omit the XML declaration for the XML data when using this packet syntax for embedding. The XML specification requires that the XML declaration be “the first thing in the entity.” This will never be the case for an embedded XML Packet, the somewhat ambiguous definition of “entity” with respect to embedding notwithstanding. You may preserve the information contained in your XML declaration by translating it into a comment or a processing instruction, such as:

```
<?was-xml version="1.0" standalone="yes"?>
```

**Padding**

In order to enable in-place edits and expansion of the embedded XML, padding should be added to the packet so that additions and edits may be easily made to the packet without overwriting existing application data. It is recommended that applications allocate 50% of the XML data size as padding, with a minimum of 4 KB. This padding must be XML compatible

whitespace. The recommended practice is to use the blank character (U+0020) for padding, in the appropriate encoding, with a newline about every 100 characters.

### **Trailer**

This mandatory processing instruction indicates the end of the XML packet.

```
<?xpacket ... ?>
```

This processing instruction has one mandatory attribute, described below. The end attribute must be the first attribute. Other unrecognized attributes may follow and should be ignored.

#### *Attribute: end*

This mandatory attribute indicates that this is the trailer. The value of the attribute is either “r” or “w”. If “r”, the packet is “read-only” and should not be updated in-place. If “w”, the packet may be updated in-place if and only if there is available space through the padding. If the size of the Header+XML data+Trailer is less than it was before the update, the padding should be increased accordingly so that the overall packet size remains constant. Use the value “r” for file formats which compute invariants over all of their contents, such as checksums. If in doubt, use “r”.

## **3.8.1 Usage Hints**

A file should be scanned byte-by-byte until a valid header is found.

A simple scanner should begin its scanning state machine by looking for one of the following byte patterns (which represents “<?xpacket begin=”):

16-bit encoding (UCS-2, UTF-16): (either big- or little-endian order)

```
0x3C 0x00 0x3F 0x00 0x78 0x00 0x70
0x00 0x61 0x00 0x63 0x00 0x6B 0x00 0x65 0x00 0x74 0x00 0x20 0x00
0x62 0x00 0x65 0x00 0x67 0x00 0x69 0x00 0x6E 0x00 0x3D [0x00]
```

8-bit or multibyte encoding (UTF-8, ASCII 7-bit, ISOLatin-1):

```
0x3C 0x3F 0x78 0x70 0x61 0x63 0x6B
0x65 0x74 0x20 0x62 0x65 0x67 0x69 0x6E 0x3D
```

The 32-bit UCS-4 pattern is similar to the UCS-2 pattern above, only with three 0x00 bytes for every one in the UCS-2 version.

For 16-bit encodings, a simple scanner cannot be sure whether the 0x00 values are in the high or low order half of the character until you read the byte order mark (the value of the begin attribute). As you can see from the pattern, it starts with the first non-zero value, regardless of byte order, which means that there may or may not be a terminal 0x00 value.

A simple scanner may choose to simply skip 0x00 values and search for the 8-bit pattern. Once the byte order is established, the scanner should switch to consuming characters rather

than bytes. After finding a matching byte pattern, the scanner must consume a quote or double-quote character. The scanner is now ready to read the value of the begin attribute.

16-bit encoding, big-endian: 0xFE 0xFF

16-bit encoding, little-endian: 0xFF 0xFE

UTF-8: 0xEF 0xBB 0xBF

Then the scanner consumes the closing quote or double-quote character. The scanner now has enough information to process the rest of the header in the appropriate character encoding. This is a potential packet.

#### *Single or Double quote*

Remember that the attribute values in the processing instruction may have either single or double quotes. The following header is well-formed:

```
<?xpacket begin="■" id='W5M0MpCehiHzreSzNTczkc9d' encoding="UTF-8"?>
```



# 4

## XMP Schemas

### 4.1 Introduction

This chapter defines the XMP Schemas, and specifies the property value types for each schema, and the associated schema vocabularies which list the allowed values. The sections include:

- Section 4.2, “XMP Schema Definitions”
- Section 4.3, “Property Value Types”
- Section 4.4, “XMP Vocabularies”

Chapter 5, “XMP Property Commentary” describes the use of, and rationale for XMP metadata properties. In addition, [Appendix A](#) specifies the PDF and Dublin Core Schemas, and [Appendix B](#) contains a proposal for schemas for media-type metadata such as for images, audio, video, etc.

#### 4.1.1 Property Value Type Representation

In the *Value Type* column of each schema table, a specific notation is used. A type preceded by a modifier, which is derived from RDF nomenclature, is defined as follows:

<i>alt</i>	An alternation. The first value in the sequence is the preferred value; all subsequent values are alternatives.
<i>bag</i>	An unordered collection of values of the specified type.
<i>seq</i>	An ordered sequence of values of the specified type.

Most of the Value Types are self-explanatory, but some need additional explanation, or are in fact references to structures. All value types are defined in Section 4.3, “Property Value Types.” For an explanation of the Category column, see “Property Categories” on page 60.

For properties whose values are strings, for which the language is unknown, the following notation is used to indicate the default value to be used (for example, see the entry for [pdf:Title](#) in the PDF Schema in the appendix):

Property Reference:	Interpretation:
xap:title/*[@xml:lang='x-default']	alternation by language; selects ‘x-default’

## 4.2 XMP Schema Definitions

Each XMP schema described in this chapter is shown in table form, along with the namespace string that identifies the schema. Each table lists all properties defined for that schema as well as the value type, category, and whether the property is aliased to any other XMP property. Additional information and commentary is given for most properties in [Chapter 5, “XMP Property Commentary.”](#)

**NOTE:** If you are viewing this document online, *Property* column entries in the following tables are linked to the corresponding entries in the Metadata Property Commentary table in the next chapter. Also, entries in the Value Type column are linked to the corresponding table entries in Section 4.3, “Property Value Types.”

### 4.2.1 XMP Core Schema

The XMP Core Schema contains metadata properties that are common to all applications.

**TABLE 4.1** XMP Core Schema

The namespace prefix is *xap*. The schema name is *http://ns.adobe.com/xap/1.0/*.

Property	Value Type	Description	Category
<a href="#">xap:Advisory</a>	bag <a href="#">XPath</a>	List of properties edited outside of the authoring application.	External
<a href="#">xap:Author</a>	<a href="#">ProperName</a>	Primary author of the document. Aliased to <a href="#">xap:Authors/*[1]</a> .	External
<a href="#">xap:Authors</a>	seq <a href="#">ProperName</a>	Authors of the document, in order of precedence.	External
<a href="#">xap:BaseURL</a>	<a href="#">URL</a>	base URL for relative URLs.	Relational
<a href="#">xap:CreateDate</a>	<a href="#">Date</a>	Document creation time.	Internal
<a href="#">xap:CreatorTool</a>	<a href="#">AgentName</a>	Tool that created the resource.	Internal
<a href="#">xap:Description</a>	alt <a href="#">Text</a>	A textual description of the content.	External
<a href="#">xap:Format</a>	<a href="#">MIMEType</a>	Distinguish various resource formats.	Internal
<a href="#">xap:Keywords</a>	bag <a href="#">Text</a>	List of descriptive phrases.	External
<a href="#">xap:Locale</a>	bag <a href="#">Locale</a>	Languages used in the content of the document.	Internal
<a href="#">xap:MetadataDate</a>	<a href="#">Date</a>	Last metadata modification time.	Internal
<a href="#">xap:ModifyDate</a>	<a href="#">Date</a>	Last resource modify time.	Internal

Property	Value Type	Description	Category
<a href="#">xap:Nickname</a>	Text	Short informal name for resource.	External
<a href="#">xap:Title</a>	alt Text	Document Title.	External

## 4.2.2 XMP Media Management Schema

Resources are identified by a document identifier, a version identifier, and a rendition class name. All versions and renditions of a document share the same document identifier. The triple (document identifier, version identifier, rendition class name) forms a key which can be used to uniquely identify each media resource. A media management tool (or simple convention for that matter) can map the triple to a filename or URL.

The creating application should assign the DocumentID. Other tools and applications that operate on the document should preserve the assigned DocumentID. If an application has media management plug-in interfaces, it may operate in conjunction with the media management system to assign the DocumentID. The DocumentID should be based on a locally generated GUID/UUID style number (see [Section 3.5.2, “Creation of Instance IDs”](#) for more information).

**TABLE 4.2** XMP Media Management Schema

The namespace prefix is *xapMM*. The namespace is *http://ns.adobe.com/xap/1.0/mm/*.

Property	Value Type	Description	Category
<a href="#">xapMM:ContainedResources</a>	bag <a href="#">ResourceRef</a>	A bag referencing all resources known to be contained in this one.	Internal
<a href="#">xapMM:ContributorResources</a>	bag <a href="#">ResourceRef</a>	Resource that in some way contributed to this one, implying a dependency.	Internal
<a href="#">xapMM:DocumentID</a>	URI	The common identifier for all versions and renditions of a document.	Internal
<a href="#">xapMM:History</a>	seq <a href="#">ResourceEvent</a>	A sequence of high-level user actions that resulted in this resource.	Internal
<a href="#">xapMM&gt;LastURL</a>	URL	Last place this resource was written.	Relational
<a href="#">xapMM:Manager</a>	<a href="#">AgentName</a>	Name of the software agent that manages this resource.	Relational

Property	Value Type	Description	Category
xapMM:ManageTo	URL	Location of managed rendition of asset.	Relational
xapMM:RenditionClass	RenditionClass	Rendition class name of this resource.	Internal
xapMM:RenditionOf	ResourceRef	Resource that this resource is a rendition of.	Internal
xapMM:SaveID	Integer	Incremented on each write to LastURL.	Relational
xapMM:VersionID	Text	The document version identifier for this resource.	Internal
xapMM:Versions	seq Version	Part of the version history of this document.	Internal

Authoring applications should maintain all of the xapMM properties except Manager and Versions which are primarily for use by a media management system. By setting these properties, an audit trail, embedded in the document itself, is maintained that allows a follow-on media manager to reconstruct the relationships among documents found on a user's unmanaged storage system.

A media management system should preserve media management metadata unchanged and update the xapMM metadata to be consistent with the results of operations performed in the media management system.

#### 4.2.2.1 Versions

Creation of a new version usually requires some explicit action on the part of the user and there consequently should be some user interface that tells the application to create a new version. This might commonly be the check-in operation for a media management system.

From the point of view of XMP metadata, all that is required to change the version number is to assign a new value for `xapMM:VersionID`, typically by incrementing it.

The media management system is responsible for storage of any old versions of the document. XMP is not involved in mapping resources to storage pathnames or in retention of old versions.

#### 4.2.2.2 Renditions

Renditions of a version of a document are created directly by some explicit user action or indirectly by some tool that is carrying out some user action. Renditions may be stored as normal resources or files, or may be embedded inside another resource by an application producing a compound document.

Creation of a rendition involves changing the `xapMM:RenditionClass` to a value that identifies the kind of rendition being created, and changing the `xapMM:RenditionOf` property to indicate the resource from which the new rendition is derived.

If an application creates a new rendition of a resource as part of embedding it in a compound document, the metadata for the modified resource should be changed as indicated in the previous paragraph and the `xapMM:ContainedResources` property of the compound document should be modified to include the new, embedded rendition.

### 4.2.3 XMP Support Schema

The following properties support cataloging and media management features. These properties do not describe a resource directly, as Author and Title do. Rather, they describe information that is related to the resource, such as the way it is identified in a storage management system.

These properties should never be embedded in the resource they describe. They are strictly for metadata managers that control metadata external to the resource file. It is expected that specific media management systems will define additional properties to support their individual features.

**TABLE 4.3** XMP Support Schema

The namespace prefix is `xapS`. The namespace is `http://ns.adobe.com/xap/1.0/s/`.

Property	Value Type	Description	Category
<code>xapS:EntityTag</code>	Text	Supports cache validation when comparing the “content value” of two blocks of metadata. Follows a convention based on the “entity” concept defined in HTTP/1.1. See <a href="http://www.ietf.org/rfc/rfc2616.txt">http://www.ietf.org/rfc/rfc2616.txt</a>	Relational
<code>xapS:FileDisposition</code>	alt <code>FileDisposition</code>	Preferred file name to save on disk, or preferred URL for publishing on a Web server, by OS.	Relational
<code>xapS:ResourceID</code>	URI	The unique identifier for this particular resource. Used for storage systems which provide a unique identifier for a stored resource which is incompatible with the format defined in <code>xapMM</code> .	Relational
<code>xapS:Size</code>	Integer	File size in bytes	Relational

#### 4.2.4 XMP Basic Job Ticket Schema

The following schema describes very simple workflow or job information.

**TABLE 4.4** XMP Basic Job Ticket Schema

The namespace prefix is *xapBJ*. The namespace is *http://ns.adobe.com/xap/1.0/bj/*.

Property	Value Type	Description	Category
<a href="#">xapBJ:JobRef</a>	bag <a href="#">Job</a>	Job(s) with which this document is associated.	External

#### 4.2.5 XMP Rights Management Schema

This schema deals with rights management. All properties are optional. All alternations are for multiple languages.

**TABLE 4.5** XMP Rights Management Schema

The namespace prefix is *xapRights*. The namespace is *http://ns.adobe.com/xap/1.0/rights/*.

Property	Value Type	Description	Category
<a href="#">xapRights:Certificate</a>	<a href="#">URL</a>	Online rights management certificate.	External
<a href="#">xapRights:Copyright</a>	alt <a href="#">Text</a>	Legal copyright notice. The alternation is for different languages.	External
<a href="#">xapRights:Marked</a>	<a href="#">Boolean</a>	Indicates that this is a rights managed resource.	External
<a href="#">xapRights:Owner</a>	bag <a href="#">ProperName</a>	Legal owner of a resource	External
<a href="#">xapRights:UsageTerms</a>	alt <a href="#">Text</a>	Text instructions on how a resource can be legally used.	External
<a href="#">xapRights:WebStatement</a>	<a href="#">URL</a>	Location of web page describing the owner and/or rights statement for this resource. (Photoshop “Image URL”).	External

## 4.3 Property Value Types

The following tables list the value types used in the XMP schemas. Some value types are represented by structures; where that occurs, a sub-heading of: *Name*, *Type*, and *Comments* is given to list each element of the structure.

**TABLE 4.6 Basic Value Types**

Type	Representation	Notes						
Boolean	<i>True</i> or <i>False</i>	The strings should be spelled as shown.						
Choice	<p>A value chosen from a single list, and represented by a string. There may also be a vocabulary qualifier indicating the vocabulary from which the value was chosen. Note that the added vocabulary values apply to the property, not to the base vocabulary. Thus, the set of values for the property is extended; the vocabularies are not. The vocabulary qualifiers for <i>Choice</i> are:</p> <ul style="list-style-type: none"> <li>● Open: Single list of preferred values, but new values may be added to the list</li> <li>● Closed: Single fixed list of values for the vocabulary.</li> </ul> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>vQual:vocabulary</td> <td>URI</td> <td>Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.</td> </tr> </tbody> </table> <p><i>Field namespace xmlns:vQual=http://ns.adobe.com/xap/1.0/ValueQualifier#.</i></p>	Name	Type	Comments	vQual:vocabulary	URI	Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.	
Name	Type	Comments						
vQual:vocabulary	URI	Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.						
Date	YYYY-MM-DDThh:mm:ss.sTZD	A date (ISO 8601) <a href="http://www.w3.org/TR/NOTE-datetime">http://www.w3.org/TR/NOTE-datetime</a>						
Integer	Signed or unsigned numeric string	Integer number representation. The string consists of an arbitrary length decimal numeric string with an option leading “+” or “-” sign.						
Locale	Choice (closed)	Identifies a language. Values from <a href="http://www.ietf.org/rfc/rfc1766.txt">http://www.ietf.org/rfc/rfc1766.txt</a>						
MIMETYPE	XChoice (closed)	image/gif, image/jpeg, etc. See: <a href="http://www.isi.edu/in-notes/iana/assignments/media-types/media-types">http://www.isi.edu/in-notes/iana/assignments/media-types/media-types</a>						
ProperName	Text	A name of a person or organization.						
Text	Unicode	A string that can contain characters from most locales. Can be marked up with additional XML tags.						
URL	URI	RFC 1630, RFC 1738, and RFC 2396 (see: <a href="http://www.w3.org/Addressing/">http://www.w3.org/Addressing/</a> ).						

Type	Representation	Notes						
XChoice	<p>A value chosen from one of several possible lists, and represented by a string. XChoice indicates that the list of choices may be extended in the schema by listing additional vocabularies. If the value is not from a core XMP schema, it's name should be specified using the vQual qualifier. Note that the added vocabulary values apply to the property, not to the base vocabulary. Thus, the set of values for the property is extended; the vocabularies are not. The vocabulary qualifiers for XChoice are:</p> <ul style="list-style-type: none"> <li>● Open: Several lists of preferred values, any value can be used.</li> <li>● Closed: Fixed list of values, but new lists can be added.</li> </ul> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>vQual:vocabulary</td> <td>URI</td> <td>Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.</td> </tr> </tbody> </table> <p><i>Field namespace xmlns:vQual=http://ns.adobe.com/xap/1.0/ValueQualifier#.</i></p>	Name	Type	Comments	vQual:vocabulary	URI	Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.	
Name	Type	Comments						
vQual:vocabulary	URI	Optional. The vocabulary name. The vocabulary implies a set of values, conventions, and semantics for values. The list of legal vocabulary items may be present in the schema.						
XPath	XPath	XML Path Language (XPath), for addressing parts of an XML document; see <a href="http://www.w3.org/Tr/xpath">http://www.w3.org/Tr/xpath</a> . XPath is a bag of lists; each list item must contain a single namespace and XPath element. See the example of how to set xap:Advisory values in Section 7.1.1.2, "Property Categories."						

TABLE 4.7 Media Management Value Types

Type	Representation	Notes
AgentName	Text	Name of a program. Format convention: "Vendor App Version for Platform," for example: "Adobe Acrobat Distiller 5.0 for Windows."
RenditionClass	XChoice (open)	The type of rendition, from a controlled vocabulary of standard names. The convention used is to have a series of ":" separated tokens and parameters, the first of which names the basic concept of the rendition. Additional tokens are optional and provide additional characteristics of the rendition. See Section 4.4, "XMP Vocabularies" for defined values.
ResourceEvent	<p><b>Name</b></p> <p>action</p>	<p><b>Type</b></p> <p>XChoice (open)</p> <p><b>Comments</b></p> <p>Conventions to use: See Section 4.4, "XMP Vocabularies" for defined values, and to see if they apply. If new strings are used, they should be verbs in past tense.</p>



Type	Representation		Notes															
ResourceEvent (cont'd)	<table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>instanceID</td> <td>URI</td> <td>The value of the rdf:Description element about attributes in the referenced resource's metadata.</td> </tr> <tr> <td>parameters</td> <td>Text</td> <td>Additional information.</td> </tr> <tr> <td>softwareAgent</td> <td>AgentName</td> <td>Name of the software agent that was used. For example, this resource could be Edited by "Microsoft Word 7.0 for Windows," and Produced by "Adobe Acrobat Distiller 3.01 for UNIX."</td> </tr> <tr> <td>when</td> <td>Date</td> <td>Optional timestamp for when this event occurred.</td> </tr> </tbody> </table> <p><i>Field namespace: xmlns:stEvt=http://ns.adobe.com/xap/1.0/sType/ResourceEvent#</i></p>	Name	Type	Comments	instanceID	URI	The value of the rdf:Description element about attributes in the referenced resource's metadata.	parameters	Text	Additional information.	softwareAgent	AgentName	Name of the software agent that was used. For example, this resource could be Edited by "Microsoft Word 7.0 for Windows," and Produced by "Adobe Acrobat Distiller 3.01 for UNIX."	when	Date	Optional timestamp for when this event occurred.		
Name	Type	Comments																
instanceID	URI	The value of the rdf:Description element about attributes in the referenced resource's metadata.																
parameters	Text	Additional information.																
softwareAgent	AgentName	Name of the software agent that was used. For example, this resource could be Edited by "Microsoft Word 7.0 for Windows," and Produced by "Adobe Acrobat Distiller 3.01 for UNIX."																
when	Date	Optional timestamp for when this event occurred.																
ResourceRef	An identifier for a resource; used to uniquely identify a resource.																	
	<table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>documentID</td> <td>URI</td> <td>See comments in xapMM Media Management schema</td> </tr> <tr> <td>versionID</td> <td>Text</td> <td></td> </tr> <tr> <td>instanceID</td> <td>URI</td> <td>The value of the rdf:Description element's about attribute, in the referenced resource's metadata.</td> </tr> <tr> <td>RenditionClass</td> <td>Text</td> <td></td> </tr> </tbody> </table> <p><i>Field namespace: xmlns:stRef=http://ns.adobe.com/xap/1.0/sType/ResourceRef#</i></p>	Name	Type	Comments	documentID	URI	See comments in xapMM Media Management schema	versionID	Text		instanceID	URI	The value of the rdf:Description element's about attribute, in the referenced resource's metadata.	RenditionClass	Text			
Name	Type	Comments																
documentID	URI	See comments in xapMM Media Management schema																
versionID	Text																	
instanceID	URI	The value of the rdf:Description element's about attribute, in the referenced resource's metadata.																
RenditionClass	Text																	

Type	Representation	Notes																		
URI	Text	<p>The purpose of this URI is to provide a unique identifier in the widest possible scope (like all time and space). Any valid URI can be used. For DocumentID URIs we recommend the following convention:</p> <p>vendor:docid:{manager}:{uri_safe_text}</p> <p>where:</p> <p><i>vendor</i> is replaced by a short representation of the company name implementing the tool that is assigning the name. For example, “adobe.”</p> <p><i>{manager}</i> is replaced by a short and unique application identifier (most Adobe applications have three-letter codes through Adobe Online, or you can use Apple Creator codes), and <i>{uri_safe_text}</i> is replaced by additional id information, as long as it is encoded in URI safe text. For example, it may contain a 128-bit UUID encoded as hex.</p> <p>Example (Adobe Acrobat):</p> <p>adobe:docid:acro:9705C6F1-81BD-11d3-9CCC-006097CEC6D4</p> <p>(See <a href="http://www.w3.org/Addressing/">http://www.w3.org/Addressing/</a> for information on URIs)</p>																		
Version	<table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>comments</td> <td>Text</td> <td>Comments concerning what was changed</td> </tr> <tr> <td>event</td> <td>ResourceEvent</td> <td>High level, formal description of what operation the user performed</td> </tr> <tr> <td>modifyDate</td> <td>Date</td> <td>Date this version checked in</td> </tr> <tr> <td>modifier</td> <td>ProperName</td> <td>Person who modified this version</td> </tr> <tr> <td>version</td> <td>Text</td> <td>Version name, such as “1.2”</td> </tr> </tbody> </table> <p><i>Field namespace: xmlns:stVer=http://ns.adobe.com/xap/1.0/sType/Version#</i></p>	Name	Type	Comments	comments	Text	Comments concerning what was changed	event	ResourceEvent	High level, formal description of what operation the user performed	modifyDate	Date	Date this version checked in	modifier	ProperName	Person who modified this version	version	Text	Version name, such as “1.2”	
Name	Type	Comments																		
comments	Text	Comments concerning what was changed																		
event	ResourceEvent	High level, formal description of what operation the user performed																		
modifyDate	Date	Date this version checked in																		
modifier	ProperName	Person who modified this version																		
version	Text	Version name, such as “1.2”																		

TABLE 4.8 Adobe Support Metadata Value Types

Type	Representation		Notes
FileDisposition	<b>Name</b>	<b>Type</b>	<b>Comments</b>
	filename	Text	Simple local filename in local os form
	OS	Text	Examples: UNIX, MacOS, Windows
	directoryPath	Text	Fully qualified pathname to directory
<i>Field namespace: xmlns:stDsp=http://ns.adobe.com/xap/1.0/sType/FileDisposition#</i>			

TABLE 4.9 Basic Job/Workflow Value Types

Type	Representation		
Job	<b>Name</b>	<b>Type</b>	<b>Comments</b>
	name	Text	Informal name of job. This name is for user display and informal systems.
	id	Text	Unique Id for the job. This field is a reference into some external job management system.
<i>Field namespace xmlns:stJob=http://ns.adobe.com/xap/1.0/sType/Job#</i>			

## 4.4 XMP Vocabularies

Vocabularies provide a means of specifying a limited but extensible set of values for a property. The metadata schema can indicate whether the set of legal values is fixed or can be extended. The flexibility can be limited in the following ways:

- Fixed list of values (closed vocabulary, choice type)
- Fixed list of values, but new lists can be added (closed vocabulary, xchoice type)
- List of preferred values, but any value can be used (open vocabulary, choice type)
- Several lists of preferred values, but any value can be used (open vocabulary, xchoice type)

If a property value is to have a very definite meaning and all users of that property must know the exact meaning, the first, most restrictive form should be used. If there are well defined sets

of values whose meanings are known, but additional value might be used without causing problems, then the less restrictive forms may should be used.

Unless otherwise specified, the default namespace for all vocabulary elements is:

<http://ns.adobe.com/xap/1.0/corevocabulary>

Table 4.10 lists the XMP vocabulary elements.

**TABLE 4.10 XMP Vocabularies**

Vocabulary for	Vocabulary elements	
xap:Locale dc:language	Values used in this vocabulary are from <a href="http://www.ietf.org/rfc/rfc1766.txt">Locale</a> : <a href="http://www.ietf.org/rfc/rfc1766.txt">http://www.ietf.org/rfc/rfc1766.txt</a>	
xap:Format dc:Format	Values used in this vocabulary are those from MIME Type. See the relevant standard for the current values. <a href="http://www.isi.edu/in-notes/iana/assignments/media-types/media-types">http://www.isi.edu/in-notes/iana/assignments/media-types/media-types</a>	
xapMM:RenditionClass	<b>Value</b>	<b>Comments</b>
	default	Indicates the master document; no additional tokens allowed
	thumbnail	For a simplified and/or reduced preview of a version. Additional tokens, if any, provide more characteristics of the thumbnail. The colon character “:” is used as a delimiter. The recommended order is: thumbnail:format:size:colorspace. Examples: thumbnail:jpeg thumbnail:16x16 thumbnail:gif:8x8:bw
	screen	For a screen resolution/Web rendition
	proof	For a review proof
xapMM:RenditionClass (cont'd)	<b>Value</b>	<b>Comments</b>
	draft	For a review rendition
	low-res	For a low resolution, full size stand-in

Vocabulary for	Vocabulary elements	
xapMM:History/*/stEvt:action	<p><b>Value</b></p> <p>converted</p> <p>copied</p> <p>created</p> <p>cropped</p> <p>edited</p> <p>filtered</p> <p>formatted</p> <p>version_updated</p> <p>printed</p> <p>published</p> <p>managed</p> <p>produced</p> <p>resized</p>	<p><b>Comments</b></p> <p>Format changed in some way</p> <p>Covers all kinds of editing, generally of the content</p> <p>Content filtering algorithm applied</p> <p>Placed under media management</p>



# 5

## XMP Property Commentary

### 5.1 XMP Properties

Table 5.1 through Table 5.4 list the properties of the XMP metadata schemas, and explain the use of, and rationale for, each property.

**TABLE 5.1 XMP Core Schema Metadata Properties**

Property	Description, Notes, Rationale
xap:Advisory	If the value of any external or relational properties were changed outside the authoring application, the value of xap:Advisory is a <i>bag</i> that points to those properties. Each item in the bag should contain a single namespace and XPath. See the example of how to set xap:Advisory values in Section 7.1.1.3, “XAP:Advisory Example.”
xap:Author	Aliased to the first item of xap:Authors.
xap:Authors	People for whom the credit of authoring this resource is attributed. The authors are listed in order of precedence, should that be significant. Each Authors item is a string representing an author. Although recommended conventions exist for the structure of the name, programs should not make any assumptions about the structure or format of the ProperName. Searches should do unanchored matches against the name strings. The string can include XML/HTML formatting and other information. Authoring applications should set the value of the xap:Author property as directed by the user or other conventions used to establish authorship. The user should be allowed to edit this property. Aggregating applications: Insertion or deletion of a document should not affect author information for either the inserted or containing document.
xap:BaseURL	If this document contains Internet links, and those links are relative, they are relative to this base URL. The purpose of this property is to provide a standard way for embedded relative URLs to be interpreted by tools. Web authoring tools will want to set the value based on their notion of where URLs will be interpreted. Should be the same as the value of the path property of the FileDisposition if the value of the OS attribute tag is “URL.”

Property	Description, Notes, Rationale
xap:CreateDate	<p>The date and time the document was originally created.</p> <p>One meaning of the date is when the first <i>Save</i> or <i>New</i> was executed for a new document. Another meaning is the date that the content was created (for example, an illustration that has been scanned). The user would need to set the <code>CreateDate</code> explicitly in that case.</p> <p>Once set, tools should not change the value, unless explicitly directed to do so by the user.</p>
xap:CreatorTool	<p>Original tool that was used to create the resource (at least the first known tool).</p> <p>If history is present in the metadata, this value should be equivalent to that of <code>xapMM:History</code>'s <code>softwareAgent</code> property.</p>
xap:Description	<p>An account of the content of the resource.</p> <p>A textual description of the resource content. The alternation is for different languages. Description may include but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.</p> <p>The description text is entirely under user control. Tools can set the default to some summary if they wish but should not overwrite user-entered information. The XML strings may contain HTML formatting tags and other information.</p> <p>The <code>xml:lang</code> attribute on the property alternative, if present, is used to identify the language on each alternative.</p>
xap:Format	<p>Defines the resource format (e.g. video, audio, etc.)</p> <p>Tools and Application should set this resource to the save format of the data.</p> <p>Recommended practice is to select a value from a controlled vocabulary (specifically, the list of Internet Media Types [MIME] defining computer media formats).</p>
xap:Keywords	<p>Can consist of an unordered list of descriptive phrases, or specify the topic of the content of the resource.</p> <p>Recommended practice is to select a value from a controlled vocabulary or formal classification scheme.</p> <p>This property is user editable. It is unlikely that the applications would change the value of <code>Keywords</code> without explicit user direction.</p> <p>Keywords are not merged when another document is embedded by an aggregating application.</p>
xap:Locale	<p>The set of languages used in the document content. This property is set and managed by applications. This property should be updated whenever a document is saved with content in a language not yet in the set.</p> <p>Recommended practice for the values of the <code>Locale</code> property is defined by <a href="#">RFC 1766</a> which includes a two-letter Language Code (taken from the <a href="#">ISO 639</a> standard), followed optionally, by a two-letter Country Code (taken from the <a href="#">ISO 3166</a> standard).</p> <p>The following are examples of language codes: 'en' for English, 'fr' for French, or 'en-uk' for English used in the United Kingdom.</p>



Property	Description, Notes, Rationale
xap:MetadataDate	<p>Date and time that any metadata for this resource was last changed.</p> <p>This property may be used to optimize searches by excluding resources that might be out-of-date with respect to externally stored metadata. When a reconciliation is executed between an external metadata store and the XMP metadata, the modify time on the XMP metadata can be compared with the modify time bounds on the external metadata, and, if the external metadata is clearly newer, its value can be merged into the XMP metadata. If the external metadata has been updated but it cannot be established that it is newer than the MetadataDate, the property-by-property timestamp comparisons may be necessary.</p>
xap:ModifyDate	<p>The date and time the resource was last modified.</p> <p>Media management systems receiving resources via check-in that have no value for ModifyDate may set the value to the check-in timestamp.</p> <p>It is not necessary for applications to update this property each time an opened resource is modified (that is, for individual edits). An acceptable interpretation of this property would be to update it at the last resource <i>write</i> (or <i>save</i>) time.</p>
xap:Nickname	<p>Short informal name for resource, e.g. “Adobe logo,” or “Feb issue,” – for UI purposes.</p> <p>A user specified value should not be overwritten by an application wishing to set the value. The value can be used for window/pane naming or other similar purposes.</p> <p>Applications may set the value if there is some convention for this type of information in place and no user-supplied value is present.</p>
xap:Title	<p>A name given to the resource. The alternation is for different languages.</p> <p>Typically, a Title will be a name by which the resource is formally known.</p> <p>The Title values are arbitrarily formatted strings. Programs should not make any assumptions about the structure or format of the Title value. The strings can include XML/HTML formatting and other information. The language can be determined by looking at an xml:lang property in the string, if present. For example:</p> <pre data-bbox="491 1270 1126 1356"> &lt;rdf:li xml:lang="fr"&gt;   XMP – Une Plateforme Extensible pour les Métadonnées &lt;/rdf:li&gt; </pre> <p>Authoring applications: should set the value of the xap:Title resource as directed by the user or other conventions used to establish the title. The user should be allowed to edit this property.</p>

TABLE 5.2 Adobe Media Management Metadata Properties

Property	Description
xapMM:ContainedResources	<p>References to resources that are directly or indirectly contained in this compound document. Contained resources that do not have any of the information needed to compose a ResourceRef value are not included.</p> <p>The resources are likely contained in this document but may have been deleted. Thus, this list is likely to be a superset. The intent is to make it easy to locate references to documents that are embedded in other documents without placing a large management burden on applications and tools.</p> <p>Since ResourceRefs are provided, it should be possible to locate the inserted documents if they exist by using a media management system or module.</p>
xapMM:ContributorResources	<p>References to resources that in some way contributed to this document, other than being contained as part of a compound document. Contributor resources that do not have any of the information needed to compose a ResourceRef value are not included.</p> <p>The resources listed in ContributorResources are intended to be dependencies for the listing resource. Although the resources are not embedded in their entirety, some component of them may be embedded or there may be some dependence such that changing one of the resources listed in ContributorResources would imply a possible change to this resource.</p> <p>Since ResourceRefs are provided, it should be possible to locate the referenced documents if they exist by using a media management system or module.</p> <p>Tools or users would need to have some way of indicating such a dependency. Applications may be able to set this resource depending on specific cases of user actions.</p>
xapMM:DocumentID	<p>The common identifier for all versions and renditions of a document. The DocumentID should be assigned when a new document is created, typically when File:New (or equivalent) is executed. Subsequent edits should not affect this value. See <a href="#">Section 3.5.2, “Creation of Instance IDs”</a> for more information.</p> <p>Media management systems and tools must not change the DocumentID. The only exception is that media management systems, tools, and aggregating applications may assign a document ID if there is none.</p> <p>Do not copy the xapMM:DocumentID value of an imported document into current document.</p>

Property	Description
xapMM:History	<p>A chronological sequence of high-level user actions which resulted in this resource.</p> <p>This is intended to give human readers a general indication of the steps taken to make the changes from the previous version to this one. The list should be at an abstract level. It is not intended to be an exhaustive keystroke or other detailed history.</p> <p>Example:</p> <p style="padding-left: 40px;">Action: Created Parameters: SoftwareAgent: Adobe Photoshop 5.5 for Windows When: ...</p> <p style="padding-left: 40px;">Action: Cropped Parameters: old size 640x480 SoftwareAgent: Adobe Photoshop 5.5 for Windows When: ...</p> <p style="padding-left: 40px;">Action: Resized Parameters: SoftwareAgent: Adobe Photoshop 5.5 for Windows When: ...</p> <p style="padding-left: 40px;">Action: Edited Parameters: SoftwareAgent: Adobe Photoshop 5.5 for Windows When: ...</p>
xapMM:LastURL	<p>Last place this resource was written.</p> <p>Each time the resource is written to a file or web server, the name of the file or server location should be stored in this property. The URL syntax handles both file and web locations.</p> <p>The purpose of this property is to enable some amount of tracing of the history of where the document has been. Copies of the document retain the LastURL value from when the tool wrote it. If the file is moved, there is some means to trace back to where it was last written.</p>
xapMM:Manager	<p>Name of the software agent that manages this resource.</p> <p>Although AgentName has a defined format, this property is intended for human use. It tells the user where to go look to find things, but does not really enable automated retrieval or check-in of documents.</p> <p>This property is set by media management tools and may be set at check-out time or earlier.</p>
xapMM:ManageTo	<p>Specifies the location of managed rendition of asset.</p>
xapMM:RenditionClass	<p>Rendition class name for this resource. This property should be absent or equal to “default” for a document version that is not a derived rendition.</p> <p>Each version of a document may have several renditions. Each rendition is distinguished by a rendition class name. This property lists the rendition class name for a particular rendition.</p> <p>If the application UI is capable of determining that a rendition is being created, then the type of rendition should be set by setting a value for xapMM:RenditionClass in the new file.</p>

Property	Description
xapMM:RenditionOf	<p>For a rendition of something, indicates the resource that this resource is a rendition of.</p> <p>This property indicates what resource is the source of this rendition.</p> <p>When a rendition is created, the creating application should set RenditionOf to indicate the resource from which this rendition is derived. The ResourceRef type allows any resource to be specified as the source for this rendition, although the DocumentID and VersionID would normally be the same in the source and the rendition.</p>
xapMM:SaveID	<p>An id number which is incremented each time the resource is written to a particular URL.</p> <p>This number is used to help track the movement of a resource through media management and file systems. Each time an application (or other tool) writes the resource out to the same location, this property value should be incremented.</p> <p>If it is being written to a different location than the LastURL property indicates, the SaveID should be reset to 1 and the LastURL property updated to reflect the new location. A History entry should also be added indicating this. (See XapMM:History)</p>
xapMM:VersionID	<p>The document version identifier for this resource.</p> <p>Each version of a document gets a new identifier. Usually these values are simply incrementing integers 1, 2, 3 . . . etc. Media management systems may have other conventions or support branching which requires a more complex scheme. The Version identifier should be kept short.</p> <p>This property should be used primarily by a media management system. Applications with sufficient interfaces to detect user intent of creating new versions (for example, Microsoft Word<sup>®</sup>), can assign new version identifiers at appropriate times, but should be careful to avoid conflicts with media management systems.</p>
xapMM:Versions	<p>The version history associated with this resource. Entry [1] is the oldest known version for this document, entry [LAST] is the current version.</p> <p>Typically, a media management system would fill in the version information in the metadata on check-in. Individual applications can also place version information into metadata, but need to be careful to avoid conflicts with media management systems. An application should avoid adding version information if a media management system is in use.</p> <p>It is not guaranteed that complete history of version from the first to this one will be present in the xapMM:Versions property. Interior version information may be compressed or eliminated and the version history may be truncated at some point.</p>

TABLE 5.3 Adobe Support Schema Metadata Properties

Property	Description
xapS:EntityTag	Supports cache validation when comparing the <i>content value</i> of two blocks of metadata. Follows a convention based on the <i>entity tag</i> concept defined in HTTP/1.1. See <a href="http://www.ietf.org/rfc/rfc2616.txt">http://www.ietf.org/rfc/rfc2616.txt</a>
xapS:FileDisposition	<p>Preferred file name to save on disk, or preferred URL for publishing on a web server, by OS.</p> <p>If absent, this property can be set to the <i>Save As</i> filename. If already set, it should be left alone. The user can explicitly edit it if desired. It can be used, at a tool's discretion, as a default for <i>Save As</i>, <i>Export</i>, or check-in operations.</p> <p>Attribute tags can be used to specify different values for different systems and environments. The following conventions should be used: <i>Windows</i>, <i>MacOS</i>, <i>Unix</i>, or <i>URL</i></p> <p>New tags can be used if the above are not adequate.</p>
xapS:ResourceID	<p>The unique identifier for this particular resource. Used for storage systems which provide a unique identifier for a stored resource which is incompatible with the format defined in xapMM.</p> <p>This property is for use by media management systems. When used, the values should meet the following requirements:</p> <p>This identifier needs to be unique in time and space. It can be a random number or can include some information uniquely associated with the document (such as creation timestamp).</p> <p>ResourceID should be set when a new resource is being created. This would typically be done by a media management system on check-in.</p> <p>In absence of a media management system this property should not be used. Authoring application should not set or use it.</p> <p>A media management system is allowed to change the ResourceID value for a document when the document is checked-in. Thus, a media management system may implement its own notion of unique identifiers for resources and override the value that was placed by a different media management system.</p>
xapS:Size	<p>File size in bytes.</p> <p>This value should be the actual file size and include any embedded metadata.</p> <p>This property would typically only be set by media management systems and used by clients of media management systems. It should be deleted by applications when a file is open to avoid confusion of the "current size" versus the last saved size.</p>

**TABLE 5.4** *XMP Basic Job Properties*

<b>Property</b>	<b>Description</b>
xapBJ:JobRef	<p>Name of the job(s) that this document is part of. Use of job names is under user control. Typical use would be to identify all documents that are part of a particular job or contract.</p> <p>There are multiple values since there may be more than one job using a particular document at any time and it may be desired to keep historical information about what jobs a document was part of.</p>

# 6

## XMP Extensibility

### 6.1 Making Custom Schema

The schemas defined by this document are *core schemas* that all implementations of XMP must support. If your metadata needs are not already covered by the core schemas, you may add your own schemas as extensions.

XMP was designed to be easily extensible, particularly for the addition of custom schemas. All software systems that support XMP must handle extension metadata. A conforming software system must be able to parse extension metadata, read/modify/write extension metadata, and serialize extension metadata back into XMP format.

To define a new schema, you must write a human-readable schema specification document. Make this specification document available to any developers who need to write code that understands your metadata.

**NOTE:** Future versions of XMP may include support for machine-readable schema specifications, but such support will always be in addition to the requirement for human readable schema specification documents.

Your specification document must include at least two items: 1) a unique name for your schema, in the form of a URI, and 2) a table which represents the name of each property, the value type, the description of the property, and its usage type.

**NOTE:** The XMP 1.0 implementation does not support aliasing for custom schemas. Usage of aliasing for custom schemas has the potential to result in data loss or software failure.

If you define properties that have structured value types, you may also need additional URI names to identify the components of a structured property value (for a core schema example, see [xapG:NaturalDimensions](#) in Table B.1, “XMP Graphics Schema” on page 71). This is only necessary if the value type is used in more than one property definition. If not, the URI you choose for the schema will be used for all component property values as well.

For example, if you are working on the JAKES project for Billingsgate.net, you might select for your schema name the URI <http://ns.billingsgate.net/JAKES/>. This does not have to be an actual URL that people can connect to on the Internet.

Once you have selected a unique name, you should also pick a short prefix. This will be used to qualify your property names (see XML namespace specification). Remember that XML implementations are free to replace your prefix name with another, but when there is no other choice, you should have a default name selected. For example, again referring to the JAKES project, you might pick “jks:” as the prefix.

Now define all desired properties, using this reference document as a guideline for the types of structures and values you can define. For example, if JAKES needs a Quay property, which has a simple text value, you would define:

**Table X.X JAKES Schema** (example of schema extension)

The namespace prefix is *jks*. The namespace is *http://ns.billingsgate.net/JAKES/*.

Property	Value Type	Description	Category
jks:Quay	Text	Place to dock	External

You can then add more properties as needed, following the RDF and XMP syntax requirements described in this document to create compatible RDF metadata.

## 6.2 New Versions of Existing Schemas

We have already seen how versions of schemas are introduced; they are part of the URL which defines a namespace. The following convention for schema versions is recommended for conforming applications.

Convention: If and only if a client application implements schema versions as fully backward compatible, we recommend that Properties be set *once* in the group associated with the earliest schema version in which they are defined. Full backwards compatibility means that schema N+1 contains all of schema N, with no changes other than new *additional* Properties. This case is exemplified by the example above: our hypothetical Dublin Core 9.0 contains all of the previous version of the Dublin Core schema, without change, except that we add the new element CRYPTOKEY. A conforming application may also define a new Property which is intended to supersede an old one. For example, suppose schema N defines a PageNumber property, whose value is a number. Later, we discover that page numbers can be arbitrary strings, like “xix” or “A-3”. For schema N+1, we define a new property ExtendedPageNumber, whose value can be a String. We continue to set the old property when appropriate, but when the page number needs to be a string, we omit the old one and only set the ExtendedPageNumber.

To add elements in a later version of a schema, just do the same thing as you would for adding elements in a different schema. Consider the hypothetical Dublin Core version 9.0, which contains the element CRYPTOKEY:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/1.1/"
    dc:creator="John Doe"
    dc:title="XMP - Extensible Metadata Platform"/>

  <rdf:Description about="" xmlns:xap="http://ns.adobe.com/xap/1.0/"
    xap:Format="text/html" xap:Locale="en"/>

  <rdf:Description about="" xmlns:dc="http://purl.org/dc/elements/9.0/"
    dc:CRYPTOKEY="29J39LJKS9JL388X,39872KD987KJ30987S08NV83LU784"/>
</rdf:RDF>
```



Note that we are allowed to reuse the *dc:* prefix if we want, since we are in a different *rdf:Description*, and thus a different *xmlns* scope.



# 7

## Application Integration Guidelines

### 7.1 Supporting XMP Metadata

This section describes what applications must do to properly support XMP Metadata. It explains what information should be considered metadata; what is needed for a User Interface; and how to store data. It also describes what metadata actions need to be performed for specific application operations like *Open*, *Save As*, and embedding an image in a document.

#### 7.1.1 Requirements

To properly support XMP metadata, the following three rules must be followed.

##### **Preserve Metadata**

Any metadata that is read as part of loading application data should by default be preserved and written back out when the application data is rewritten. This behavior may be overridden by explicit user action. The metadata must be preserved regardless of whether the application understands its semantics.

When application data is loaded from one format and written out as a different format, metadata should be preserved by including it in the file in the saved format, or writing it to a separate, parallel file if the new format does not accommodate metadata.

##### **Set Meaningful and Useful Information in Metadata**

Each application will have a set of metadata properties that are relevant to that application and useful to customers who use metadata. The application should set metadata values for those properties. This would be done at least at save-time and, if the application provides an interface to access or edit metadata, at points where those interfaces could be used. This might mean that the metadata would need to be updated each time the underlying application data, to which the metadata is related, is changed.

One important rule is that you should not define a property in the metadata if you can't find a value for it that makes sense. An example of this would be the properties for the XMP Rights Management Schema; if you don't have valid values for those properties, those properties should not be included.

There is a fine line between metadata and application data. Ultimately, it is a judgment call as to what to include in metadata. Here are some useful guidelines for making this judgment:

- Information that is only useful to the internals of the application is not a good candidate for public metadata.
- Information that can only be used by a tool that is deeply involved with application internals is not a good candidate for metadata.
- Information whose size or structure has a linear relationship with the size of the application data is not a good candidate for metadata. It will be too large and impractical to maintain.
- Information that is only useful in conjunction with other application data that is not available in metadata is not a good candidate for metadata.
- Information that users or tools may want to browse, select, or otherwise use to characterize an asset is a good candidate.
- Explicit user information that the application currently supports is a good candidate for metadata.
- Information that the application happens to have that is present in some industry metadata standard is a good candidate.

### **Allow display/edit**

If the application has metadata or it is a user expectation to be able to look at or edit or enter metadata, then the application should provide a metadata display and editing interface.

A key requirement for such an interface is that it accommodate the extensibility model of XMP, namely that metadata defined and introduced after the product ships receive the same status as metadata defined by Adobe or before the product has shipped. Metadata properties defined later may be as or more important to the customer than metadata defined prior to shipment.

Fully general display and editing of metadata is complicated by the existence of sequence and structured values, and, particularly, sequences of structured values. The user interface may have to support more than simple (name, value) pairs.

#### **7.1.1.1 Aliasing**

The definition of aliases should specify which of the alias forms are normally written in the serialized RDF. Multiple forms may be written; by default, only the base form should be written. This primarily applies to the creation of new metadata.

When existing XMP metadata is parsed and normalized, multiple forms of aliased properties should be allowed. The values should be compared for equality, with any differences reported as errors. The same forms should be output when re-serializing the metadata.

#### **7.1.1.2 Property Categories**

XMP metadata may be read by an application when it opens a resource. For the results of modifying metadata associated with a resource to give predictable behavior, it is important that authoring applications follow some simple guidelines. These are usage guidelines and are not enforced by software; in situations where they do not make sense, they may be ignored. The

three categories of XMP properties are defined on [page 10](#), in [Section 2.4](#), “[Model and Terminology](#).”

### 7.1.1.3 XAP:Advisory Example

If External or Relational properties are modified outside of the authoring application, they should be mentioned in the `xap:Advisory` property. The value of that property is a *bag*, and each item contains a single namespace and XPath element.

For example, let's assume that a resource has had its `dc:format`, `xapG:NumberOfColors`, and `xapG:img:Resolution/stRes:unit` properties modified externally, so those properties need to be marked as *Advisory*: The following example illustrates how that would be done in RDF:

```
<xap:Advisory>
  <rdf:Bag>
    <rdf:li>
      http://purl.org/dc/elements/1.1/ format
    </rdf:li>
    <rdf:li>
      http://ns.adobe.com/xap/1.0/xap/g/ NumberOfColors
    </rdf:li>
    <rdf:li>
      http://ns.adobe.com/xap/1.0/xap/g/img/ Resolution/stRes:units
    </rdf:li>
  </rdf:Bag>
</xap:Advisory>
```

Notice that the two parts, the namespace and the property's XPath, must be separated by at least one character of whitespace.

## 7.2 Metadata Actions For Specific Application Operations

The following sections describe which metadata actions should be performed for specific application operations such as *Open*, *Save* and *Save-As*, creating new versions and renditions, placing images, and embedding unaltered copy.

### 7.2.1 General Comment on Schemas

Applications should not assume that the only properties that they will encounter in a XMP metadata object are those defined in the current schema or that all properties defined in the schema are present.

If schema information for a specific property is not available (not in the schema, or there is no schema), then primitive values are considered simple text. Structures and containers such as sequences and bags still function without any problems because their representation is self-describing.

### 7.2.1.1 New Document

When a new document is created, the following metadata properties should be set by the creating application:

- xap:CreateDate
- xap:Locale
- xapMM:DocumentID
- xapMM:VersionID (value = 1)
- xapMM:RenditionClass (value set to “default”)
- xapMM:History (add first entry)

### 7.2.1.2 Save and Save-As

When a new document is first saved, the following additional metadata properties should be set by the creating application:

- xap:Author (to default, if available and not already set)
- xap:ModifyDate
- xap:MetadataDate
- xap:Format
- xap:Title (to default, if available and not already set)
- media-specific metadata properties as applicable
- xapMM>LastURL
- xapMM:SaveID
- xapMM:Manager
- xapBJ:JobName
- xap:Locale

If there are other pieces of metadata redundant with XMP in the application data (typically for legacy reasons), it must be made consistent with XMP at save time.

Individual applications should set the following metadata properties:

- xap:CreateDate
- xap:Format
- xap:Locale
- any media-specific properties
- general metadata such as xap:Author, xap:Description, and xap:Title
- xapMM:ContainedResources to track embedded documents
- xapMM:History (for major impact changes in the document)

### 7.2.1.3 Versions

Creation of a new version usually requires some explicit action on the part of the user and consequently there must be some user interface that allows the application to deduce that a new version is to be created. This might commonly be the check-in operation for an authoring application or a media management system.

From the point of view of XMP metadata, all that is required is to assign a new `xapMM:VersionID` typically by incrementing it. The application creating the new version is responsible for assigning the new `VersionID`.

### 7.2.1.4 Renditions

**Renditions** of a version of a document are created directly by some explicit user action or indirectly by some tool that is carrying out some user action. Renditions may be stored as normal resource or files, or may be embedded inside another resource by an aggregating application.

Creation of a rendition involves changing the `xapMM:RenditionClass` to a value that identifies the kind of rendition being created, and changing the `xapMM:RenditionOf` property to indicate the resource from which the new rendition is derived.

If an aggregating application creates a new rendition of a resource as part of embedding a resource, the metadata for the modified resource should be changed as indicated in the previous paragraph and the `xapMM:ContainedResources` property of the aggregate resource should be modified to include the new, embedded rendition.

### 7.2.1.5 Document *Open* Time

When documents are opened, there is some processing that is required to reconcile metadata values that may have been edited in various locations. At *open* time, applications need to analyze and merge metadata as detailed below.

Metadata sources include the following:

- XMP metadata stored with the application document representation
- XMP metadata stored in a separate metadata repository
- non-XMP metadata stored with the application document representation
- non-XMP metadata stored in a separate metadata repository

At *open* time, any internal metadata that has been edited outside the application should be reset to values consistent with the application data. This is effectively recovering from the editing of what should have been read-only metadata.

Metadata conflicts should be resolved using timestamps to pick the latest updates which may have been made from different sources.

Thus, the following steps should occur when the document is opened:

1. XMP metadata is extracted from the file and processed.
2. Metadata is obtained from the media management system, if available.
3. Media management system metadata is merged into the application file metadata using property timestamps to identify metadata items that have been modified, and if necessary, choosing the latest value according to an application-defined algorithm when conflicting changes have been made.

When and if the file is re-saved by the user, the merged metadata is written out.

### 7.2.2 Media Management System Actions

The media management system should set the following metadata properties:

- xapMM:Manager
- xapMM:Versions (the media management system may truncate the length of the version history that is embedded in the document metadata to keep it from growing without bounds)
- xapMM:VersionID
- xapS(all) (as needed)
- xapMM:ManageTo

Media management systems likely have their own database for storing metadata about managed documents. It is the responsibility of the media management system to extract XMP metadata and merge it with its internal database to maintain data integrity. Similarly, it is the media management system's responsibility to supply updated XMP metadata to applications so that embedded metadata can be properly maintained and reflect changes made to metadata by other applications and tools while the document is not being actively edited.

### 7.2.3 Document Embedding and Metadata Preservation

This section contains guidelines on how applications should handle the embedding of one document, or image, in another document.

**NOTE:** The guidelines given here about how to handle different kinds of embedding are based on a model that reflects the conceptual document model on which XMP is based. Implementors who follow these guidelines must consider how the specific operational embedding model, inherent in their application, aligns with the guidelines described here to determine which embedding procedures to utilize. In general, the preference should always be to preserve as much of the original metadata as practical; implementors are encouraged to consider whether they can enhance their implementation's embedding model so as to make this possible.



### 7.2.3.1 Placed-Image Metadata

A placed image refers to the embedding of all or a part of one document (the *contributor document*) into another document (the *compound document*). The term *placed image* is used because the contributor document often contributes an image, but the concept can apply to any content that is contributed from one document and embedded in another. The application that does this is called the aggregating application. Examples of placed images include placement of a page or image into a PDF file by Acrobat<sup>®</sup>, or placement of an article, page, or image on a page by Adobe InDesign<sup>™</sup> or Framemaker<sup>®</sup>.

When a placement of content from a contributor document into an compound document occurs, one of four things should happen with metadata, according to the guidelines in the following sections. This does not apply when a reference to another document is placed into an compound document. When just a reference is added:

- The contributor document's metadata is not brought into the compound document.
- The `xapMM:ContributorResources` property in the compound document should have a `ResourceRef` to the contributor document appended to it. If there is `xapMM:DocumentID` metadata present in the contributor document, it should be used to form the `ResourceRef`. If not, a `ResourceRef` should be constructed using a URL to the file containing the contributor document.

### 7.2.3.2 Full Unaltered Copy Embedding

When the contributor document is embedded in the compound document unaltered and in its entirety, all metadata should be copied with the contributor document. Examples of this form would include placement of an image from a contributor document that contains only that image into an compound document without alteration. A more subtle example would be the placement of an image that is part of an compound document but was at some point in the past a complete document into another compound document.

The underlying principle at work in this case is that the content and metadata of the placed document are not being altered; only the storage location changes and this does not affect either content or metadata.

Typically, an application would not need to take any action to implement this case because the contributor file contents that are embedded in the compound document already include the metadata.

In addition, in the document level metadata for the compound document, `xapMM:ContainedResources` should include a reference to each embedded resource. Specifically, the `xapMM:ContainedResources` property in the compound document should have a `ResourceRef` to the contributor document appended to it. If there is `xapMM:DocumentID` metadata present in the contributor document, it should be used to form the `ResourceRef`. If not, a `ResourceRef` should be constructed using a URL to the file containing the contributor document.

### 7.2.3.3 Subset or New Rendition Embedding

When a subset of the content of the contributor document is embedded structurally in an compound document or when the form or rendition of the content from the contributor document is changed from the original, some metadata should be included with the placed content. Examples of this would include placement of a page from a multi-page contributor document into an compound document or the alteration of the form of an image (such as the size, colorspace, or resolution) as it is embedded into an compound document.

When a component (or possibly all) of one document is embedded in another document, the following metadata properties should be set. This list represents a minimum. Ideally, all of the metadata should be preserved, but this will not always be practical because of size and granularity constraints.

When a component of a document is placed into another document, the component maintains its own metadata. The compound document also maintains its own metadata. The philosophy is that there should be at least minimal identifying information on the placed component so that its origin and characteristics could be traced, if necessary. It is not required that all of the metadata be maintained in this instance because of the possible size and management implications.

The metadata associated with the embedded contributor document content should include at a minimum:

- **xapMM:DocumentID**: set to the value of xapMM:DocumentID from the contributor document if one was present, otherwise set to a value based on the URL of the contributor document filename.
- **xapMM:VersionID** if there was a xapMM:VersionID present in the original.
- **xapMM:RenditionClass** should be set to a new value by the aggregating application. The value should be consistent with either the component's previous location or the component's new context. Examples:
  - “Figure 15” for the placement of Figure 15 of a document into the new one (Placed component was “Figure 15” of the source document).
  - “Image from page 15” for the placement of an image that was on page 15 into a new document (assuming the image didn't have its own document id)
- **xapMM:History**: The value from the contributor document, if any, can be ignored and the value set to a new single entry of the form: (action = “RenditionCreated,” softwareAgent = “the aggregating app,” when = “current date/time,” parameters = “additional information the aggregating app chooses to include”).
- **xapMM>LastURL** should be set to the value of xapMM>LastURL from the contributor document if one was present, or else set to the contributor document filename.
- **xapMM:RenditionOf** should be set to reference the contributor document. The reference should be to the document, version, and rendition class of the contributor. If those properties are not set in the contributor document, a URL for the contributor filename should be used as the document id in the reference.

In addition, in the metadata for the compound document, `xapMM:ContainedResources` should include a reference to each embedded resource.

If possible, the following additional metadata should be set in the metadata associated with the embedded content from the contributor document:

- `xap:Author`
- Applicable properties from the `xapRights` schema: set values for properties for which you have appropriate data

#### 7.2.3.4 Small Subset Embedding

When a small subset of the contributor document content is embedded in an undistinguished stream in the compound document, then no metadata is associated with the placed content.

Examples of this form would include embedding a copy of a few words or paragraphs of text or a few frames of video content from a contributor document into an compound document.

#### 7.2.3.5 Embedding of a Document Which Had No Metadata

When a contributor document has no metadata the application may include no metadata with the embedded placed content in the compound document. The application may set some metadata consistent with the previous sections if it determines that this would be in the user's best interest.





# PDF and Dublin Core Schema

This appendix contains the schemas for the PDF and Dublin Core schemas. Aliases to XMP core metadata properties are specified where they apply.

## A.1 Adobe PDF Schema

**TABLE A.1** *Adobe PDF Schema*

*The namespace prefix is pdf. The namespace is <http://ns.adobe.com/pdf/1.3/>.*

Property	Value Type	Description	Alias Of	Category
pdf:Author	ProperName	Document author	dc:creator/*[1]	External
pdf:BaseURL	URL	base URL for relative URLs	xap:BaseURL	Relational
pdf:CreationDate	Date	Document creation time	xap:CreateDate	Internal
pdf:Creator	AgentName	Tool that created the resource	xap:CreatorTool	Internal
pdf:Keywords	Text	Keywords		External
pdf:ModDate	Date	Last modify date	xap:ModifyDate	Internal
pdf:PDFVersion	Text	PDF file version (for example: 1.0, 1.3, etc.)		Internal
pdf:Producer	AgentName	Name of tool that created PDF document		Internal
pdf:Subject	Text	Document subject text	dc:description/*[@xml:lang='x-default']	External
pdf>Title	Text	Document title text	dc:title/*[@xml:lang='x-default']	External

## A.2 Dublin Core Schema

**TABLE A.2** *Dublin Core Schema*

The namespace prefix is *dc*. The namespace is <http://purl.org/dc/elements/1.1/>.

Property	Value Type	Description	Aliased by	Category
dc:contributor	bag <a href="#">ProperName</a>	Other contributors to document		External
dc:coverage	<a href="#">Text</a>	The extent or scope of the resource		External
dc:creator	seq <a href="#">ProperName</a>	Authors who created document	<a href="#">xap:Authors</a> <sup>1</sup>	External
dc:date	seq <a href="#">Date</a>	Date(s) that something interesting happened to the resource		External
dc:description	alt <a href="#">Text</a>	A textual description of the resource, selected by language	<a href="#">xap:Description</a>	External
dc:format	<a href="#">MIMETYPE</a>	Data format	<a href="#">xap:Format</a>	Internal
dc:identifier	<a href="#">Text</a>	Unique identifier of the resource		External
dc:language	bag <a href="#">Locale</a>	Languages used in the content of the document	<a href="#">xap:Locale</a>	Internal
dc:publisher	bag <a href="#">ProperName</a>	Publishers		External
dc:relation	bag <a href="#">Text</a>	Relationships to other documents		
dc:rights	alt <a href="#">Text</a>	Informal rights statement, selected by language	<a href="#">xapRights:Copyright</a>	External
dc:source	<a href="#">Text</a>	Unique identifier of the work from which this resource was derived		External
dc:subject	bag <a href="#">Text</a>	The topic of the resource	<a href="#">xap:Keywords</a>	External
dc:title	alt <a href="#">Text</a>	Document title	<a href="#">xap:Title</a>	External
dc:type	bag <a href="#">XChoice</a> (open)	novel, poem, working paper, etc.		External

1. [xap:Author](#) aliases [dc:creator](#)/\*[1]



# Proposed Media-Type Schemas

The schemas in this appendix are proposed for basic media-type (content-specific) metadata. They are not intended to be thorough or all encompassing schemas. They are presented for review, especially by subject matter experts.

This section also contains the Value Types for the Media-Type Schemas – in section B.2, “Property Value Types,” and the associated Vocabulary elements are in section B.3, “Vocabulary for Media-Specific Schema.”

## B.1 XMP Media-Type Schemas

**TABLE B.1 XMP Graphics Schema**

*The namespace prefix is xapG. The namespace is <http://ns.adobe.com/xap/1.0/g/>.*

Property	Value Type	Description	Category
xapG:ColorSpace	ColorMode	The name of the group of colors from which colors in the document are taken	Internal
xapG:Compression	XChoice (open)	The name of the compression algorithm used to compress all or part of the document.	Internal
xapG:GraphicsType	XChoice (closed)	Raster, Vector, Dynamic	Internal
xapG:NaturalDimensions	Dimensions	Presentation dimensions	Relational
xapG:NumberOfColors	Integer	Number of colors in color space (256, 65536, etc.)	Internal
xapG:NumberOfInks	Integer	Number of process and spot colors needed to print entire document including any contained documents	Internal

**TABLE B.2 XMP Graphics: Image Schema**

*The namespace prefix is xapGimg. The namespace is <http://ns.adobe.com/xap/1.0/g/img/>.*

Property	Value Type	Description	Category
xapGimg:Dimensions	Dimensions	Image dimensions in sampling unit	Internal

Property	Value Type	Description	Category
xapGImg:Resolution	Resolution	Number of pixels per unit measure	Relational

**TABLE B.3 XMP Dynamic Media Schema**

The namespace prefix is *xapDyn*. The namespace is <http://ns.adobe.com/xap/1.0/dyn/>.

Property	Value Type	Description	Category
xapDyn:Duration	Duration	Total duration of resource	Relational
xapDyn:NTracks	Integer	Number of tracks or channels	Internal
xapDyn:Tracks	seq TrackDesc	Array of track descriptions	External

**TABLE B.4 XMP Dynamic Media: Video Schema**

The namespace prefix is *xapDynV*. The namespace is <http://ns.adobe.com/xap/1.0/dyn/v/>.

Property	Value Type	Description	Category
xapDynV:BitRate	Integer	Bits per second	Relational
xapDynV:Dimensions	Dimensions	Of playback view rectangle	Relational
xapDynV:Interleaved	Boolean	If true, NTSC fields, otherwise frames	Internal
xapDynV:NaturalRate	Real	Fields/Frames per second	Relational
xapDynV:Compression	XChoice (open)	Video compression technique	Internal
xapDynV:Encoding	XChoice (open)	Video encoding (for example: NTSC, PAL, SECAM, etc.)	Internal

**TABLE B.5 XMP Dynamic Media: Audio Schema**

The namespace prefix is *xapDynA*. The namespace is <http://ns.adobe.com/xap/1.0/dyn/a/>.

Property	Value Type	Description	Category
xapDynA:ChannelCount	Integer	Number of audio channels	Internal



Property	Value Type	Description	Category
xapDynA:Compression	XChoice (open)	Audio compression technique	Internal
xapDynA:Rate	Real	Samples per second	Relational
xapDynA:SampleSize	Integer	Number of bits per sample	Internal
xapDynA:Volume	Real	0.0 = silence, 1.0 = maximum volume	Relational

**TABLE B.6 XMP Text Schema**

The namespace prefix is *xapT*. The namespace is *http://ns.adobe.com/xap/1.0/t/*.

Property	Value Type	Description	Category
xapT:Encoding	TextEncoding	Specifies the name of the character encoding standard used for text in the document. For example, ISO-8859-1	Internal
xapT:FontList	bag Font	Lists the names of all fonts used in the document.	Internal

**TABLE B.7 XMP Text: Paged-Text Schema**

The namespace prefix is *xapTPg*. The namespace is *http://ns.adobe.com/xap/1.0/t/pg/*.

Property	Value Type	Description	Category
xapTPg:MaxPageSize	Dimensions	Size of the largest page in the document (including any in contained documents)	Internal
xapTPg:NPages	Integer	Number of pages in the document (including any in contained documents)	Internal

## B.2 Property Value Types

The following tables list the value types used in the proposed XMP media-type schemas.

**TABLE B.8 Basic Value Types for Media-Specific Schemas**

Type	Representation	Notes	
ColorMode	XChoice (open)	A string representing the color space in a document. An open choice that includes the following: RGB, CMYK, Indexed, Monotone, Duotone, Tritone, or Quadtone	
Dimensions	<b>Name</b>	<b>Type</b>	<b>Comments</b>
	w	Real	Width
	h	Real	Height
	unit	XChoice (Open)	Examples: inch, mm, pixel, pica, point
<i>Field namespace: xmlns:stDim=http://ns.adobe.com/xap/1.0/sType/Dimensions#</i>			
Duration	<b>Name</b>	<b>Type</b>	<b>Comments</b>
	length	Real	Number of frames, or seconds
	unit	XChoice (Open)	Examples: <i>frame-count</i> ; <i>elapsed-seconds</i>
<i>Field namespace: xmlns:stDuration = http://ns.adobe.com/xap/1.0/sType/Duration#</i>			
Font	<b>Name</b>	<b>Type</b>	<b>Comments</b>
	name	Text	Specifies the PostScript font name of the font. For TrueType fonts, the PostScript font name is obtained in the <i>name</i> table in a well-formed TrueType font.
	embedded	Boolean	Specifies whether the font is embedded in the document.
<i>Field namespace: xmlns:stFnt=http://ns.adobe.com/xap/1.0/sType/Font#</i>			
Real	A numeric value, integer or real. Integer or decimal number of arbitrary precision. Consists of a decimal numeric string with an optional single decimal point and an optional leading “+” or “-” sign. The following qualifier can optionally appear:		

Type	Representation	Notes	
	<b>Qualifier Name</b> vQual:binRep	<b>Type</b> Text	<b>Comments</b> Optional binary representation qualifier. This qualifier provides an alternate, binary representation for the number when an exact value is needed. The text is interpreted as: <std size> , <endian> , <hexadecimal value> where: <ul style="list-style-type: none"> <li>• <i>std</i> is the standard name. Only IEEE754 is currently supported.</li> <li>• <i>size</i> is S for 32-bit and D for 64-bit</li> <li>• <i>endian</i> is L for little-endian order, B for big-endian order</li> <li>• <i>hexadecimal value</i> is the value represented in hexadecimal.</li> </ul> For example, the value might be: IEEE754D,L,3A4901F387D31108 RDF Note: vQual:binRep is stored as a qualifier on the property node. A Real does not actually contain nested properties.  <i>Field namespace: xmlns:vQual=http://ns.adobe.com/xap/1.0/ValueQualifier#.</i>
Resolution	<b>Name</b> x y unit	<b>Type</b> Real Real XChoice (open)	<b>Comments</b> Resolution in the x direction Resolution in the y direction dpi, etc.  <i>Field namespace: xmlns:stRes=http://ns.adobe.com/xap/1.0/sType/Resolution</i>
TextEncoding	XChoice (open)		Specifies the encoding of text in the document. A string from a controlled vocabulary of charset encodings. See vocabulary for <i>xapT:Encoding</i> .  <b>NOTE:</b> If multiple encodings are used in a document, the primary encoding should be specified.
TrackDesc	<b>Name</b> start length bits codec description name rate	<b>Type</b> Duration Duration Integer Text Text Text Real	<b>Comments</b> Start time relative to doc start Run time at normal speed Bits per sample Name of codec Human readable description  samples or fields/frames per second

Type	Representation	Notes										
TrackDesc (cont'd)	<table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> <th>Comments</th> </tr> </thead> <tbody> <tr> <td>type</td> <td>XChoice (open)</td> <td>Media type (for example, audio, video, caption, etc.)</td> </tr> <tr> <td>blending</td> <td>XChoice (open)</td> <td>Name for blending algorithm</td> </tr> </tbody> </table>	Name	Type	Comments	type	XChoice (open)	Media type (for example, audio, video, caption, etc.)	blending	XChoice (open)	Name for blending algorithm	<p>Field namespace: <code>xmlns:stTrk=http://ns.adobe.com/xap/1.0/sType/TrackDesc#</code></p>	
Name	Type	Comments										
type	XChoice (open)	Media type (for example, audio, video, caption, etc.)										
blending	XChoice (open)	Name for blending algorithm										

## B.3 Vocabulary for Media-Specific Schema

TABLE B.9 XMP Vocabularies

Vocabulary for	Vocabulary elements	
xapDyn:Tracks/*/stTrk:blending	<b>Value</b>	<b>Comments</b>
	copy	
	alpha_blend	
	transparent	
	dither	
xapG:ColorSpace	<b>Value</b>	<b>Comments</b>
	RGB	
	CMYK	
	indexed	
	monotone	
	duotone	
	tritone	
	quadtone	

Vocabulary for	Vocabulary elements	
xapG:Compression xapDynA:Compression	<b>Value</b>	<b>Comments</b>
	LZW	
	JPEG	
	Huffman	
xapG:GraphicsType	<b>Value</b>	<b>Comments</b>
	raster	
	vector	
	dynamic	
xapT:Encoding	Specifies the encoding of text in the document.	
	<b>Value</b>	<b>Comments</b>
	ASCII	
	ISO-8859-n	Specifies the ISO encoding for Latin fonts, according to ISO 8859; <i>n</i> specifies the part of ISO 8859. (ISO 8859 has 12 parts; for example, ISO 8859-1 is one of several encodings for Latin character fonts, and ISO 8859-5 is for Cyrillic fonts)
	Mac	Encoding for Macintosh Roman character set
	WinAnsi	Encoding for Windows Roman character set
	UTF-8	Unicode one-byte
	UTF-16	Unicode two-byte encoding. Document text includes a BOM (Byte-Order Marker) to indicate byte order
	UTF-16BE	Unicode two-byte encoding; big-endian byte order; (BOM not in document text)
	UTF-16LE	Unicode two-byte encoding; little-endian byte order; (BOM not in document text)
	UCS-2	see ISO10646
	UCS-4	see ISO10646
	EUC-CN	Simplified Chinese
	GBK	Simplified Chinese

Vocabulary for	Vocabulary elements	
xapT:Encoding (cont'd)	Big_Five EUC-TW Shift-JIS EUC-JP EUC-KR Unified_Hangul _Code custom	Traditional Chinese Traditional Chinese Japanese Japanese Korean Korean Font has a custom encoding that does not conform to standard encodings
xapG:NaturalDimensions/stDim:unit xapGImg:Dimensions/stDim:unit xapDynV:Dimensions /stDim:unit xapTPg:MaxPageSize /stDim:unit xapGImg:Resolution	<b>Value</b>  inch mm pixel pica point	<b>Comments</b>
xapDyn:Tracks.type	<b>Value</b>  audio video image caption sprite text href	<b>Comments</b>

Vocabulary for	Vocabulary elements	
xapMM:RenditionClass	<b>Value</b>	<b>Comments</b>
	default	Indicates the master document. No additional tokens allowed.
	thumbnail	For a simplified and/or reduced preview of a version. Additional tokens, if any, provide more characteristics of the thumbnail. The colon character “:” is used as a delimiter. The recommended order is: thumbnail:format:size:colorspace. Examples: thumbnail:jpeg thumbnail:16x16 thumbnail:gif:8x8:bw
	screen	For a screen resolution/Web rendition
	proof	For a review proof
	draft	For a review rendition
	low-res	For a low resolution, full size stand-in



# Proposed Media-Type Schemas

*Vocabulary for Media-Specific Schema*