Adding Intelligence to Media

XMP SPECIFICATION PART 1

DATA MODEL, SERIALIZATION, AND CORE PROPERTIES

April, 2012

Adobe

Copyright © 2012 Adobe Systems Incorporated. All rights reserved.

Extensible Metadata Platform (XMP) Specification: Part 1, Data Model, Serialization, and Core Properties.

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, ActionScript, Creative Suite, Photoshop, and the XMP logo 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, Mac OS, 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

Page

©Adobe Systems Incorporated, 2012

©Adobe Systems Incorporated, 2012

Preface

This document set provides a complete specification for the Extensible Metadata Platform (XMP), which provides a standard format for the creation, processing, and interchange of metadata, for a wide variety of resources.

The *Specification* has three parts:

- *Part 1, Data Model, Serialization, and Core Properties,* covers the basic metadata representation model that is the foundation of the XMP standard format. The data model prescribes how XMP metadata can be organized; it is independent of file format or specific usage. The serialization information prescribes how the data model is represented in XML, specifically RDF/XML. Core properties are those XMP properties that have general applicability across a broad range of resources; these include general-purpose namespaces such as Dublin Core. This document also provides details needed to implement a metadata manipulation system such as the XMP Toolkit (which is available from Adobe®).
- *Part 2, Additional Properties*, provides detailed property lists and descriptions for standard XMP metadata namespaces beyond the core properties; these include special-purpose namespaces for Adobe applications such as Photoshop®. It also provides information on extending existing namespaces and creating new namespaces.
- *Part 3, Storage in Files*, provides information about how serialized XMP metadata is packaged into XMP packets and embedded in different file formats. It includes information about how XMP relates to and incorporates other metadata formats, and how to reconcile values that are represented in multiple metadata formats.

About this document

This document, *XMP Specification Part 1, Data Model, Serialization, and Core Properties*, provides a thorough understanding of the XMP data model. It is useful for anyone who wishes to use XMP metadata, including both developers and end-users of applications that handle metadata for resources of any kind.

The serialization information is vital for developers of applications that will generate, process, or manage files containing XMP metadata. Such developers may use either the XMP Toolkit provided by Adobe, or independent implementations. The serialization information will also interest application developers wishing to understand file content.

This document also provides guidelines and important information for programmers who will implement XMP metadata manipulation systems of their own.

Conventions used in this document

This document uses normative language that follows ISO practice as defined in Annex H of *ISO/IEC Directives Part 2*. The following table lists the most common verbal forms and gives the definitions from Annex H:

Table 1 — Normative language

Verbal form	Meaning
shall, shall not	Requirements to be strictly followed in order to conform to this document and from which no deviation is permitted.
should, should not	Among several possibilities, one is recommended as particularly suitable without excluding others, or a certain course of action is preferred but not necessarily required, or (in the negative form) a certain possibility or course of action is deprecated but not prohibited.
may, need not	A course of action that is permissible within the limits of the document.
can, cannot	Statements of possibility and capability.

©Adobe Systems Incorporated, 2012

©Adobe Systems Incorporated, 2012

Introduction

This document specifies a standard for the definition, creation, and processing of metadata that can be applied to a broad range of resource types. The Extensible Metadata Platform (XMP) was introduced by Adobe Systems Incorporated in 2001 and has since established itself as a critical technology for improving business efficiency in many industries

Metadata is data that describes the characteristics or *properties* of a resource. It can be distinguished from the main content of a resource. For example, for a word processing document, the *content* includes the actual text data and formatting information, while the *metadata* might include properties such as author, modification date, or copyright status.

Some information could be treated as either content or metadata, depending on context. In general, metadata is useful without regard for a resource's content. For example, a list of all fonts used in a document could be useful metadata, while information about the specific font used for a specific paragraph on a page would be logically treated as content.

Metadata allows users and applications to work more effectively with resources. Applications can make use of metadata, even if they cannot understand the native format of the resource's content. Metadata can greatly increase the utility of resources in collaborative production workflows. For example, an image file might contain metadata such as its working title, description, and intellectual property rights. Accessing the metadata makes it easier to perform such tasks as searching for images, locating image captions, or determining the copyright clearance to use an image.

File systems have typically provided metadata such as file modification dates and sizes. Other metadata can be provided by other applications, or by users. Metadata might or might not be stored as part of the resource with which it is associated.

©Adobe Systems Incorporated, 2012

©Adobe Systems Incorporated, 2012

Extensible metadata platform (XMP) — Part 1: Data model, serialization, and core properties

1 Scope

This document defines two essential components of XMP metadata:

- *Data model*: The data model is the most fundamental aspect. This is an abstract model that defines the forms of XMP metadata items, essentially the structure of statements that XMP can make about resources.
- *Serialization*: The serialization of XMP defines how any instance of the XMP data model can be recorded as XML.

In addition, this document defines a collection of *core properties*, which are XMP metadata items that can be applied across a broad range of file formats and domains of usage.

The embedding of XMP packets in specific file formats and domain-specific XMP properties are beyond the scope of this document.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEEE 754, *Standard for Binary Floating-Point Arithmetic*
http://grouper.ieee.org/groups/754/

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, November 1996
http://www.ietf.org/rfc/rfc2046.txt

IETF RFC 3066, *Tags for the Identification of Languages*, January 2001
http://www.ietf.org/rfc/rfc3066.txt

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005
http://www.ietf.org/rfc/rfc3986.txt

Date and Time Formats, W3C submission, September 1997
http://www.w3.org/TR/NOTE-datetime

Dublin Core Metadata Element Set, Version 1.1, October 2010
http://dublincore.org/documents/dces/

Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008
http://www.w3.org/TR/2008/REC-xml-20081126/

Namespaces in XML 1.0 (Second Edition), August 2006
http://www.w3.org/TR/2006/REC-xml-names-20060816/

RDF/XML Syntax Specification (Revised), W3C Recommendation 10 February 2004
http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

The Unicode Standard
http://www.unicode.org/standard/standard.html

©Adobe Systems Incorporated, 2012

URIs, URLs, and URNs: Clarifications and Recommendations 1.0, W3C Note 21 September 2001
http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
character data
XML text that is not markup

[Extensible Markup Language specification, Section 2.4]

3.2
element content
XML text between the start-tag and end-tag of an element

[Extensible Markup Language specification, Section 3.1, syntax production 43]

3.3
empty-element tag
XML tag identifying an element with no content

[Extensible Markup Language specification, Section 3.1]

3.4
NCName
XML name that does not contain a colon (':', U+003A)

[Namespaces in XML, Section 3, syntax production 4]

3.5
property
named container for a metadata value at the top level of an XMP packet

NOTE Lower-level components of an XMP packet are structure fields, array items, and qualifiers.

3.6
RDF
Resource Description Framework, an XML syntax for describing metadata

[RDF/XML Syntax Specification]

3.7
rendition (of a resource)
resource that is a rendering of some other resource in a particular form

NOTE Various renditions of a resource have the same content in differing forms. For example, a digital image could have high resolution, low resolution, or thumbnail renditions. A text document could be in a word processor format for editing or rendered as a PDF for sharing. See also version (of a resource).

3.8
URI
Uniform Resource Identifier, a compact sequence of characters that identifies an abstract or physical resource

[IETF RFC 3986]

©Adobe Systems Incorporated, 2012

3.9
version (of a resource)
resource that is the result of editing some other resource

NOTE Different versions of a resource typically have differing content in the same form. See also rendition (of a resource).

3.10
XML element
primary component of XML syntax

[Extensible Markup Language specification, Section 3, syntax production 39]

3.11
XML expanded name
pair of strings consisting of a namespace URI and a local name

[Namespaces in XML, Section 2.1]

3.12
XMP processor
hardware or software component that is responsible for reading, modifying, or writing XMP

3.13
white space
XML text consisting of one or more space characters, carriage returns, line feeds, or tabs

[Extensible Markup Language specification, Section 2.3]

4 Notations

Table 1 shows the type styles used for specific types of text:

Table 1 — Conventions for type styles

Typeface style	Used for
Bold	XMP property names. For example, **xmp:CreateDate**
Italic	Terms when defined in text, document titles, or emphasis.

The following names are used for important Unicode characters:

- SPACE - U+0020
- QUOTE - U+0022 (")
- APOSTROPHE - U+0027 (')

5 Conformance

5.1 General

Conforming XMP packets shall adhere to all requirements of this document and conforming XMP packets are not required to use any feature other than those explicitly required by this document.

NOTE The proper mechanism by which XML can presumptively identify itself as being an XMP packet is described in 7.3, “Optional outer XML”, and 7.4, “rdf:RDF and rdf:Description elements”.

©Adobe Systems Incorporated, 2012

5.2 Conforming readers

A conforming reader shall comply with all requirements regarding reader functional behaviour specified in this document. The requirements of this document with respect to reader behaviour are stated in terms of general functional requirements applicable to all conforming readers. A conforming reader shall accept all output from conforming writers, including optional output that conforming writers may produce. This document does not prescribe any specific technical design, user interface, or implementation details for conforming readers.

5.3 Conforming writers

A conforming writer shall comply with all requirements regarding writer functional behaviour specified in this document. The requirements of this document with respect to writer behaviour are stated in terms of general functional requirements applicable to all conforming writers and focus on the creation of conforming XMP packets. This document does not prescribe any specific technical design, user interface, or implementation details for conforming writers.

5.4 Conforming products

A conforming product shall comply with all requirements regarding reader and writer functional behaviour as specified in this document.

6 Data model

6.1 XMP packets

An instance of the XMP data model is called an *XMP packet*. An XMP packet is a set of XMP metadata properties. Each property has a name and a value. Each property name in an XMP packet shall be unique within that packet.

NOTE 1 The restriction for unique names means that it is invalid to have multiple occurrences of the same property name in an XMP packet. Multiple values are represented using an XMP array value (see 6.3.4, “Array values”). Instead of having three **dc:subject** properties that each hold one keyword, there would be one **dc:subject** property that is an array with three items.

All properties in a single XMP packet shall describe a single resource. Separate XMP packets may describe the same resource. Conflict resolution for separate packets that describe the same resource is beyond the scope of this document.

Lower-level components of an XMP packet (structure fields or array items) may describe one or more other resources.

NOTE 2 The provision for lower-level components about some other resource is not an addition to the data model, in that this is not a formal feature of the data model and is not reflected in written XMP in any specific manner. Rather, it is a clarification to the “one packet about one resource” rule, to avoid disallowing certain data models. The XMP about a compound resource might have a list of constituent resources and even copies of XMP about those constituents. This would all be modelled using the defined XMP value forms.

The composition of a resource and the precise association of an XMP packet with a resource is beyond the scope of this document. Where feasible, an XMP packet should be physically associated with the resource that it describes.

NOTE 3 A common resource is a complete digital file, or an identifiable part of a digital file such as an embedded image in PDF. The structure of a PDF file and the manner of associating XMP with any particular component of a PDF file is beyond the scope of this document.

©Adobe Systems Incorporated, 2012

The XMP packet that describes a digital file or part of a digital file should be embedded in the file using standard features of the file format to provide the association between the XMP packet and the resource. The embedding mechanisms for specific file formats are beyond the scope of this document.

An XMP packet may contain a URI, called the *AboutURI*, that identifies the resource that the packet describes. The URI scheme, detailed URI syntax, and association of the URI with any target entity is beyond the scope of this document.

NOTE 4 It is possible for an XMP packet to not contain an AboutURI and not have a physical association with the resource. Instead, there can be an external means of association.

EXAMPLE Consider the statement, “The author of *Moby Dick* is Herman Melville”. This statement is represented by metadata in which the resource is the book “Moby Dick”, the property name is “author”, and the property value is “Herman Melville”, as in Figure 1. (This is only a diagram, not an example of well-formed XMP.)

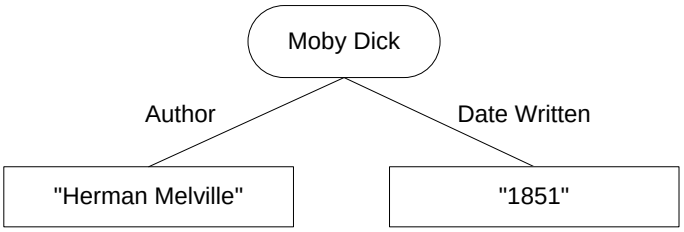

Figure 1 — Simple properties example diagram

NOTE 5 Notation such as that in Figure 1 is used in this document to illustrate the XMP data model.

An XMP processor should accept all well-formed XMP as input, regardless of the data model expressed, and should by default preserve all unanticipated XMP when modifying a resource.

NOTE 6 The intent of these rules is that XMP is generally open to arbitrary extension of properties. Users of XMP are allowed to freely invent custom metadata and to expect XMP-aware applications to support the creation, modification, and viewing of that metadata. Therefore, this is expressed as a recommendation instead of as a requirement because any particular environment could have local policies about XMP usage.

6.2 XMP names

Properties (6.1, “XMP packets”) have names, as do fields of structure values (6.3.3, “Structure values”) and qualifiers (6.4, “Qualifiers”). All names in XMP shall be XML expanded names, consisting of a namespace URI and a local name. The namespace URI for an XMP name shall not be empty. Two XMP names shall be equivalent if their namespace URIs are identical and their local names are identical. This comparison shall be physical, byte-for-byte equality using the same Unicode encoding. Other processing, including but not limited to Unicode character normalizations, shall not be applied.

NOTE 1 XML namespace URIs are generally best viewed as string literals. There is no commitment that the URI is resolvable to a Web resource. Although many XML namespace URIs begin with "http://", there might be no HTTP page at that address.

The namespace prefix used in written XML—and, as a consequence, in XMP—serves only as a key to look up the appropriate URI. For convenience in this document, XMP names are commonly written in a **prefix:local** style, for example, **dc:title**. The relevant URI for the prefix used in this document is either explicit, clear from local context, or irrelevant (as in the generic value-form diagrams where the specific URI does not matter).

NOTE 2 The specific convenience is that **dc:title** is more concise and readable than something like ("http://purl.org/dc/elements/1.1/", creator) in the cases where the namespace URI is known and meaningful. This is especially so when the precise URI is not relevant, as in an artificial example.

©Adobe Systems Incorporated, 2012

A namespace URI used in XMP should end in a character that is not allowed in an XML NCName (the local name). Recommended characters are the slash ("/", U+002F) or the number sign ("#", U+0023). This can improve compatibility with applications that concatenate the namespace URI and local name, avoiding potential collisions.

NOTE 3 The textual concatenation of a namespace URI and local name is seen in generic RDF processors that utilize the RDF triple notation. See B.3, “Namespace URI termination”, for details.

Other than **xml:** and **rdf:**, all namespaces used in 6, “Data model”, and Figure 5, “Qualifiers example”, are illustrative. In particular, the "http://ns.adobe.com/xmp-example/" URI is fictional. The use of specific XMP names in the illustrations does not imply that they are defined in this document. The namespaces defined in 8, “Core properties”, are normative.

Following typical XML and World Wide Web practice, the creation of XMP names should use a namespace URI that incorporates a domain name owned by the creator. This diminishes the chance of namespace collisions and identifies the origin of the namespace.

In this document, the **xml:** prefix is bound to the URI "http://www.w3.org/XML/1998/" that is defined in the Extensible Markup Language specification. The **rdf:** prefix is bound to the URI "http://www.w3.org/1999/02/22-rdf-syntax-ns#" that is defined in the RDF/XML Syntax Specification. The Extensible Markup Language specification and the RDF/XML Syntax Specification heavily restrict the use of these namespaces. Except for **rdf:type**, these namespaces shall not be used for any XMP property or structure field. Except for **rdf:type** and **xml:lang**, these namespaces shall not be used for any XMP qualifier. See also 7.9.2.5, “RDF Typed Nodes”.

6.3 XMP value forms

6.3.1 General

Values in the XMP data model have one of three forms: simple, structure, or array. There are two variants of simple values: normal and URI. There are three variants of the array form: unordered array, ordered array, and alternative array. The fields in structures and the items in arrays may have any value form. There is no fixed bound on the complexity of XMP data modelling.

These forms are the primitive values of XMP. Higher-level data types may be defined that combine these primitive forms with additional constraints, such as those defined in 8, “Core properties”.

6.3.2 Simple values

A simple value is a string of Unicode text as defined in The Unicode Standard. The string may be empty.

There are two variants of simple values: normal and URI. The URI variant of a simple value should be used for values that represent URIs; the normal variant should be used for all other simple values.

NOTE The distinction between normal and URI simple values is not critical to the organization of the abstract XMP data model. The distinction does have an effect on the RDF serialization, as seen in 7.5, “Simple valued XMP properties”. This allows XMP data modelling to more closely align with general RDF data modelling.

EXAMPLE In Figure 2, the document XMP_Specification.pdf is shown with two properties, each with a simple value:

The value of the property **dc:format** is "application/pdf".

The value of the property **xmp:CreateDate** is "2002-08-15T17:10:04-06:00".

©Adobe Systems Incorporated, 2012

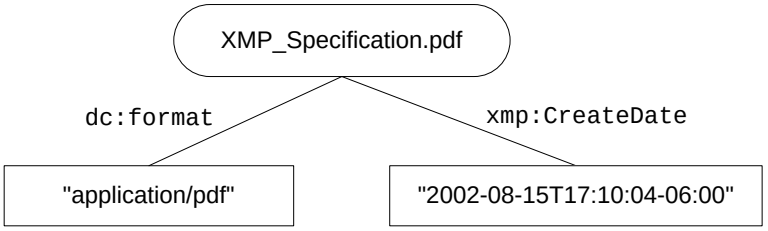

Figure 2 — Simple values example

6.3.3 Structure values

A structure is a container for zero or more named fields. The order of fields in a structure shall not be significant. Fields may be optional or required.

Each field in a structure shall have a unique name within that structure. Field names shall be XML expanded names. Fields need not be in the same namespace as their parent structure nor in the same namespace as other fields in the structure.

Each field in a structure may have any value form. The usage and consistency of fields in a given structure type is beyond the scope of this document.

EXAMPLE Figure 3 shows a single structured property with three fields: **stDim:w** (width), **stDim:h** (height) and **stDim:unit** (units), whose values are "8.5", "11.0", and "inch".

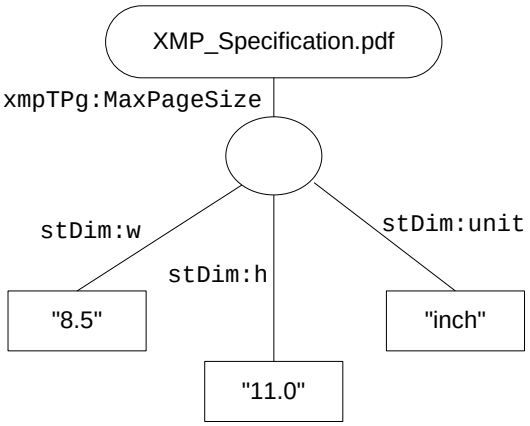

Figure 3 — Example of structure values

6.3.4 Array values

An array is a container for zero or more items indexed by ordinal position, starting from 1. The form of the array items may be any XMP value form. All items in an array shall have the same data type.

There are three variants of array: ordered, unordered, and alternative. The variant indicates the anticipated use of the array and constrains what XMP processors may do with it:

- An *unordered array* shall have no meaning or constraints on the order of items within it. The items in an unordered array may be reordered at any time.
- The items in an *ordered array* are ordered by their indices. The items in an ordered array shall not be arbitrarily reordered. The meaning of the order may be defined by data type or by application. Except for the data types defined in 8, “Core properties”, this document does not specify any assumed or default

©Adobe Systems Incorporated, 2012

meaning to the order of items in an ordered array.

- The items in an *alternative array* are ordered and shall not be arbitrarily reordered. The meaning of the order may be defined by data type or by application. Except for the data types defined in 8, “Core properties”, this document does not specify any assumed or default ordering. If any item is a preferred default, it should be the first item in the array. The first item in the array should be chosen when no other criteria apply. An alternative array need not have an explicitly designed default item.

NOTE 1 The intent is that a reader who has no idea how to choose an item from the alternative array is encouraged to pick the first item.

NOTE 2 The anticipated usage of an unordered or ordered array is to consider all items together, such as an unordered list of keywords or an ordered list of authors. The anticipated usage of an alternative array is to select one item based on some criteria, for example, having multiple descriptions of a resource in various languages, then selecting one based on the user’s preferred language. Both ordered and alternative arrays have ordered items; the anticipated usage determines which array variant to use.

EXAMPLE Figure 4 shows an example of the Dublin Core property **dc:subject** (see 8.3, “Dublin Core namespace”), which is an unordered array. In this example, it contains three items.

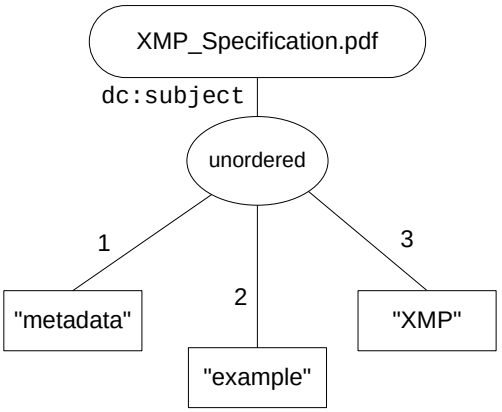

Figure 4 — Array values example

6.4 Qualifiers

XMP qualifiers may be used to attach annotations to any XMP value, without changing the form of that value. For example, a simple value remains a simple value even when some XMP processor attaches arbitrary qualifiers to it. Qualifiers are metadata about the value to which they are attached. Each qualifier has a name and a value. The names shall be XML expanded names. The names of all qualifiers attached to a particular value shall be unique in that value. The value of a qualifier may be any XMP value form. A qualifier value may have qualifiers.

The **xml:lang** qualifier shall have a simple non-URI value and shall not have qualifiers on its value. An **xml:lang** qualifier on a structure or array should be considered a default language for the structure fields or array items. In accordance with IETF RFC 3066, the value of the **xml:lang** qualifier shall be a language code and all comparisons of **xml:lang** values shall be case-insensitive.

EXAMPLE Figure 5 shows an example of qualifiers.

©Adobe Systems Incorporated, 2012

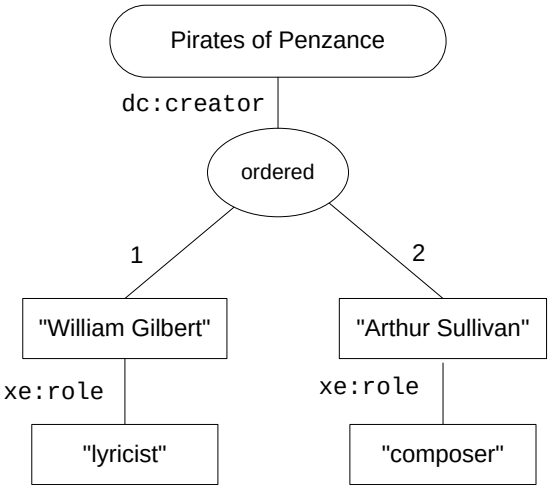

Figure 5 — Qualifiers example

7 Serialization

7.1 General

The abstract XMP data model needs a concrete representation when given a physical representation such as a digital file or a printed document. This document defines a canonical serialization of XMP metadata using a subset of the RDF metadata syntax. The RDF serialization shall use Unicode text as defined in The Unicode Standard. The choice of Unicode encoding (UTF-8, UTF-16, or UTF-32) is beyond the scope of this document. Other file embedding or usage standards may specify the Unicode encoding.

For this serialization, a single XMP packet shall be serialized using a single **rdf:RDF** XML element.

Serialized XMP shall be well-formed XML and well-formed RDF. An XMP reader shall conform to the rules of XML and RDF given in their respective specifications.

An XMP reader shall recognize and honour a leading Unicode U+FEFF character as a byte-order marker. An XMP writer using UTF-16 or UTF-32 should include a leading Unicode U+FEFF character. An XMP writer using UTF-8 may include a leading Unicode U+FEFF character, although it is not recommended.

NOTE 1 One reason to avoid the U+FEFF with UTF-8 is that devices might exist that read only UTF-8 and are not prepared for a leading U+FEFF. The only rationale for using a leading U+FEFF with UTF-8 is as a clear encoding marker for when a reader might get either UTF-8 or UTF-16/32.

NOTE 2 The XMP serialization is intentionally presented as a fragment of an XML document, not as a fully formed XML document. That is, it is presented as a single outer XML element and element content, with no mention of the XML document prolog. This is done to allow the inclusion of multiple XMP packets in larger XML documents.

7.2 Equivalent RDF and XML

The normative statements in 7.4 to 7.8 define a canonical usage of the RDF syntax. Equivalent forms of RDF syntax that are allowed or prohibited are defined in 7.9, “Equivalent forms of RDF”. Serialization of XMP uses only a subset of the RDF syntax. Parts of the RDF syntax not presented in this document shall not be used in serialized XMP.

Except where explicitly noted, XML white space, comments, and processing instructions may be written anywhere allowed by the RDF/XML Syntax Specification. Non-white character data is heavily constrained by RDF and XMP. It shall be used only in the element content of leaf elements that represent simple XMP values.

©Adobe Systems Incorporated, 2012

Comments and processing instructions may be ignored when reading and need not be preserved when updating an XMP packet.

NOTE 1 Phrases in this document such as "… element content shall consist of only …" do not constitute an explicit prohibition of white space, comments, or processing instructions. Such phrases restrict only the use of XML elements, attributes, and non-white character data.

NOTE 2 The purpose of the RDF serialization of XMP is to carry an instance of the XMP data model. XML comments and processing instructions have no effect on the XMP data model, no matter where they appear. White space outside of the **rdf:RDF** element has no effect on the XMP data model. Allowed white space inside the **rdf:RDF** element has no effect on the XMP data model except when it is part of the element content for a simple value (7.5, "Simple valued XMP properties"), in which case it is part of the value.

All equivalent forms of XML text may be written. This includes but is not limited to:

- Use of either an empty-element tag (of the form <ns:name/>) or an element with empty element content (of the form <ns:name></ns:name>).
- Use of either QUOTEs or APOSTROPHEs for attribute values.
- Order of attributes within an element.
- Distribution of xmlns attributes.
- The specific prefix associated with an XML namespace URI.

NOTE 3 When XMP is embedded within digital files, including white-space padding is sometimes helpful. Doing so facilitates modification of the XMP packet in-place. The rest of the file is unaffected, which could eliminate a need to rewrite the entire file if the XMP changes in size. Appropriate padding is SPACE characters placed anywhere white space is allowed by the general XML syntax and XMP serialization rules, with a linefeed (U+000A) every 100 characters or so to improve human display. The amount of padding is workflow-dependent; around 2000 bytes is often a reasonable amount.

7.3 Optional outer XML

7.3.1 General

Other XML elements may appear around the **rdf:RDF** element. The XML processing instructions and elements defined in this clause can assist in locating the XMP packet in some use cases.

7.3.2 XMP packet wrapper

A wrapper consisting of a pair of XML processing instructions (PIs) may be placed around the **rdf:RDF** element. If used, the wrapped packet layout shall consist of the following, in order: a header PI, the serialized XMP data model (the XMP packet) with optional white-space padding, and a trailer PI.

NOTE 1 File formats or use cases defined elsewhere can forbid the packet wrapper, for instance where minimal size is paramount or where the stored XMP is not contiguous. Discontiguous XMP can occur in a file format that models paged virtual memory.

NOTE 2 The packet wrapper, if used, facilitates primitive byte-oriented XMP packet scanning and in-place editing in data streams of unknown format. The packet wrapper has no purpose other than to provide markers allowing the packet scanner to locate the bounds of the XMP packet. The packet wrapper has no meaning once the XMP has been located, whether by scanning or format knowledge. The recommended practice is to use format knowledge and locate the XMP by other means whenever possible. Byte-oriented packet scanning is fragile and is appropriate only as a means of last resort.

EXAMPLE This outline of a wrapped XMP packet shows the text of the header and trailer:

```
<?xpacket begin="❑" id="W5M0MpCehiHzreSzNTczkc9d"?>
   <rdf:RDF xmlns:rdf= ...>
      ...
   </rdf:RDF>
 ... XML white space as padding ...
```


©Adobe Systems Incorporated, 2012

```
<?xpacket end="w"?>
```

The header PI shall be an XML processing instruction of exactly the form shown in Figure 6. The text of the header PI contains a GUID, making it unlikely to appear by accident in the data stream. In Figure 6, the character ❑ represents the Unicode character U+FEFF used as a byte-order marker. The U+FEFF may be omitted from the begin="".

```
<?xpacket begin="❑" id="W5M0MpCehiHzreSzNTczkc9d"?>
```

Figure 6 — Header PI form

QUOTEs should be used around the values for "begin" and "id". APOSTROPHEs may be used instead. A single SPACE shall be used before "begin" and before "id". Other text may appear immediately before the closing "?>"; it shall be ignored.

The trailer PI shall be an XML processing instruction of exactly one of the two forms shown in Figure 7. QUOTEs should be used around the value assigned to "end". APOSTROPHEs may be used instead. No white space shall be used within the trailer PI, except for a single SPACE before the "end".

```
<?xpacket end="w"?>
<?xpacket end="r"?>
```

Figure 7 — Allowed forms of the trailer PI

The end="w" or end="r" portion shall be used by packet scanning processors to determine whether the XMP may be modified in-place. The end="w" form indicates writable and the end="r" form indicates read-only. The writable or read-only indication should be ignored by all “smart” (not packet scanning) processors.

NOTE 3 A smart processor has implicit permission to use more appropriate means to determine whether it is OK to modify the XMP and to modify a file appropriately. Examples include using file system permissions or updating a separate checksum that is part of some file formats. A packet scanner has no context other than the packet itself.

7.3.3 x:xmpmeta element

An optional **x:xmpmeta** element may be placed around the **rdf:RDF** element. The element’s namespace URI shall be "adobe:ns:meta/".

The purpose of this element is to identify XMP metadata within general XML text that might contain other non-XMP uses of RDF. While this element might be used in any XMP, it has no meaning in other contexts. An XMP processor should tolerate an **x:xmpmeta** element in any input and look within it for the **rdf:RDF** element.

If both a packet wrapper and an **x:xmpmeta** element are present, the **x:xmpmeta** element may be inside or outside of the packet wrapper. While there are no standard attributes for the **x:xmpmeta** element, XMP processors may write custom attributes. Unknown attributes shall be ignored when reading.

EXAMPLE An example of **x:xmpmeta**:

```
<x:xmpmeta xmlns:x="adobe:ns:meta/">
   <rdf:RDF xmlns:rdf= ...>
      ...
   </rdf:RDF>
</x:xmpmeta>
```

7.4 rdf:RDF and rdf:Description elements

A single XMP packet shall be serialized using a single **rdf:RDF** XML element. The **rdf:RDF** element content shall consist of only zero or more **rdf:Description** elements.

©Adobe Systems Incorporated, 2012

The element content of top-level **rdf:Description** elements shall consist of zero or more XML elements for XMP properties. XMP properties may be arbitrarily apportioned among the **rdf:Description** elements.

The recommended approach is to have either a single **rdf:Description** element containing all XMP properties or a separate **rdf:Description** element for each XMP property namespace.

If the XMP data model has an AboutURI (6.1, “XMP packets”), that same URI shall be the value of an **rdf:about** attribute in each top-level **rdf:Description** element. Otherwise, the **rdf:about** attributes for all top-level **rdf:Description** elements shall be present with an empty value. The **rdf:about** attribute shall not be used in more deeply nested rdf:Description elements.

For compatibility with very early XMP usage, it is recommended that XMP readers tolerate a missing **rdf:about** attribute and treat it as present with an empty value. It is also recommended that XMP readers tolerate a mix of empty and non-empty **rdf:about** values, as long as all non-empty values are identical.

EXAMPLE An **rdf:RDF** element containing one **rdf:Description** element:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/">
   <rdf:Description rdf:about="">
      <xmp:Rating>3</xmp:Rating>
   </rdf:Description>
</rdf:RDF>
```

The RDF serialization of an XMP property shall be an XML element whose name is the name of the XMP property. The element content shall be determined by the form of the XMP value being serialized (simple, structure, or array), and whether the XMP value has qualifiers.

7.5 Simple valued XMP properties

The element content for an unqualified XMP property with a normal (non-URI) simple value (6.3.2, “Simple values”) shall be text that provides the value. The text shall contain only character data, entity references, character references, and CDATA sections. The element shall not contain nested XML elements.

EXAMPLE 1 See **xmp:Rating** in the example in 7.4, “rdf:RDF and rdf:Description elements”.

The element content for an XMP property with a URI simple value shall be empty. The value shall be provided as the value of an **rdf:resource** attribute attached to the XML element.

EXAMPLE 2 A URI simple value:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/">
   <rdf:Description rdf:about="">
      <xmp:BaseURL rdf:resource="http://www.adobe.com/"/>
   </rdf:Description>
</rdf:RDF>
```

General RDF allows an **rdf:parseType="Literal"** attribute to be placed in certain XML elements. This attribute specifies that the entire XML content of that element be treated as a single literal string. The **rdf:parseType="Literal"** attribute shall not be used in XMP.

When an XMP simple value contains XML markup characters, the value shall be written using XML entities or CDATA sections.

NOTE The use of CDATA sections is discouraged. They are not necessary and there is no way to escape the presence of "]]>" in a value. XML entities are sufficient for all cases.

EXAMPLE 3 Examples of simple values containing XML markup:

©Adobe Systems Incorporated, 2012

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">
   <rdf:Description rdf:about="">
      <xe:Entity>Embedded &lt;bold&gt;XML&lt;/bold&gt; markup</xe:Entity>
      <xe:CDATA><![CDATA[Embedded <bold>XML</bold> markup]]></xe:CDATA>
   </rdf:Description>
</rdf:RDF>
```

7.6 Structure valued XMP properties

The element content for an unqualified XMP property with a structure value (6.3.3, “Structure values”) shall be a nested **rdf:Description** element. The element content of the nested **rdf:Description** element shall consist of zero or more XML elements whose names are the names of the fields of the XMP structure.

The element content for each field in the structure shall follow the rules for properties, varying according to the form of the XMP value being serialized (simple, structure, or array), and whether the XMP value has qualifiers.

EXAMPLE Serialized XMP property with a structure value:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"
          xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#">

   <rdf:Description rdf:about="" >

      <xmpTPg:MaxPageSize>
         <rdf:Description>
            <stDim:h>11.0</stDim:h>
            <stDim:w>8.5</stDim:w>
            <stDim:unit>inch</stDim:unit>
         </rdf:Description>
      </xmpTPg:MaxPageSize>

   </rdf:Description>

</rdf:RDF>
```

NOTE Many XMP processors use a more concise notation for structure values as described in 7.9.2.3, “rdf:parseType="Resource" attribute”.

7.7 Array valued XMP properties

The element content for an unqualified XMP property with an array value (6.3.4, “Array values”) shall consist of exactly one nested element. The nested element shall be one of the following:

- An **rdf:Bag** element for an unordered array.
- An **rdf:Seq** element for an ordered array.
- An **rdf:Alt** element for an alternative array.

The nested element’s element content shall consist of zero or more **rdf:li** elements, one for each item in the array.

The element content of the **rdf:li** element for each array item shall follow the rules for properties, varying according to the form of the XMP value being serialized (simple, structure, or array), and whether the XMP value has qualifiers.

EXAMPLE Serialized XMP property with an unordered array value containing three items:

©Adobe Systems Incorporated, 2012

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="">

      <dc:subject>
         <rdf:Bag>
            <rdf:li>XMP</rdf:li>
            <rdf:li>metadata</rdf:li>
            <rdf:li>ISO standard</rdf:li>
         </rdf:Bag>
      </dc:subject>

   </rdf:Description>

</rdf:RDF>
```

7.8 Qualifiers

Except for the **xml:lang** qualifier, the presence of qualifiers significantly modifies the RDF serialization of an XMP value.

NOTE 1 The use of the xml:lang attribute is not a part of the RDF formal grammar. The RDF/XML Syntax Specification states, “The **xml:lang** attribute can be used on any node element or property element to indicate that the included content is in the given language.”

An **xml:lang** qualifier shall be serialized as an **xml:lang** attribute attached to the named XML element for any property, structure field, array item (**rdf:li**), or qualifier which has the qualified value. The **xml:lang** attribute may be used on any of these elements regardless of the form of the value (it is not restricted to simple values). The **xml:lang** attribute shall not be used on an **rdf:Description**, **rdf:Bag**, **rdf:Seq**, **rdf:Alt**, or **rdf:value** element.

NOTE 2 Clause 6.4, “Qualifiers”, states that the **xml:lang** qualifier shall have a simple non-URI value and shall not have qualifiers on its value. The reason for this restriction is the serialization of an **xml:lang** qualifier as an XML attribute. This serialization of **xml:lang** follows standard RDF practice.

EXAMPLE 1 Serialized XMP with **xml:lang** qualifiers:

```
<!-- These examples illustrate the syntax for xml:lang qualifiers. -->
<!-- They do not imply particularly appropriate use of xml:lang. -->

<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/">

   <rdf:Description rdf:about="">

      <dc:source xml:lang="en-us">Adobe XMP Specification, April 2010</dc:source>

      <xmp:BaseURL rdf:resource="http://www.adobe.com/" xml:lang="en"/>

      <dc:subject xml:lang="en">
         <rdf:Bag>
            <rdf:li>XMP</rdf:li>
            <rdf:li>metadata</rdf:li>
            <rdf:li>ISO standard</rdf:li>
            <rdf:li xml:lang="fr">Norme internationale de l’ISO</rdf:li>
         </rdf:Bag>
      </dc:subject>
```


©Adobe Systems Incorporated, 2012

```
   </rdf:Description>
</rdf:RDF>
```

If a value has any qualifier other than **xml:lang**, the value shall be serialized as a nested **rdf:Description** element. The element content of that **rdf:Description** element shall consist of exactly one **rdf:value** element and one or more XML elements whose names are the names of the qualifiers. This form should not be used when there are no qualifiers other than **xml:lang**.

The element content of the **rdf:value** element shall be the original XMP value being serialized, the one which is qualified. The element content of the **rdf:value** element and the qualifier elements shall follow the rules for properties, varying according to the form of the respective XMP values being serialized (simple, structure, or array), and whether the qualifier values are themselves further qualified.

The **rdf:value** element shall not contain an **xml:lang** attribute and shall not contain nested general qualifiers.

NOTE 3 Although the **rdf:value** element looks a lot like a structure field, it is not a structure field and is not allowed to have qualifiers in the start tag (**xml:lang**) or content (nested general qualifiers). Example 3, “Prohibited nesting of general qualifiers:”, illustrates this restriction.

EXAMPLE 2 Serialized XMP with general qualifiers:

```
<!-- These examples illustrate the syntax for general qualifiers. -->
<!-- They do not imply particularly appropriate use of general qualifiers. -->

<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/"
          xmlns:xe="http://ns.adobe.com/xmp-example/">

   <rdf:Description rdf:about="">

      <dc:source>
         <rdf:Description>
            <rdf:value>Adobe XMP Specification, April 2010</rdf:value>
            <xe:qualifier>artificial example</xe:qualifier>
         </rdf:Description>
      </dc:source>

      <xmp:BaseURL>
         <rdf:Description>
            <rdf:value rdf:resource="http://www.adobe.com/"/>
            <xe:qualifier>artificial example</xe:qualifier>
         </rdf:Description>
      </xmp:BaseURL>

      <dc:subject>
         <rdf:Bag>
            <rdf:li>XMP</rdf:li>
            <rdf:li>
               <rdf:Description>
                  <rdf:value>metadata</rdf:value>
                  <xe:qualifier>artificial example</xe:qualifier>
               </rdf:Description>
            </rdf:li>
            <rdf:li>
               <rdf:Description>  <!-- Discouraged without qualifiers. ->
                  <rdf:value>ISO standard</rdf:value>
               </rdf:Description>
            </rdf:li>
```

©Adobe Systems Incorporated, 2012

```
         </rdf:Bag>
      </dc:subject>

   </rdf:Description>

</rdf:RDF>
```

EXAMPLE 3 Prohibited nesting of general qualifiers:

```
<!-- This example illustrates prohibited nesting of general qualifiers. -->

<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">

   <rdf:Description rdf:about="">

      <!-- This usage is permitted. -->
      <xe:source-a>
         <rdf:Description>
            <xe:qual1>one</xe:qual1>
            <xe:qual2>two</xe:qual2>
            <rdf:value>Adobe XMP Specification, April 2010</rdf:value>
         </rdf:Description>
      </xe:source-a>

      <!-- This usage is prohibited. -->
      <xe:source-b>
         <rdf:Description>
            <xe:qual1>one</xe:qual1>
            <rdf:value>
               <rdf:Description>
                  <xe:qual2>two</xe:qual2>
                  <rdf:value>Adobe XMP Specification, April 2010</rdf:value>
               </rdf:Description>
            </rdf:value>
         </rdf:Description>
      </xe:source-b>

   </rdf:Description>

</rdf:RDF>
```

7.9 Equivalent forms of RDF

7.9.1 General

The RDF presented in 7.4 to 7.8 defines a canonical form for XMP serialization. The RDF/XML Syntax Specification defines a number of equivalent forms, each presenting distinct XML usage that conveys an equivalent RDF data model and hence an equivalent XMP data model. Some equivalent forms of RDF are allowed in XMP; some are prohibited.

7.9.2 Allowed equivalent RDF

7.9.2.1 Summary

The following equivalent forms of RDF may be used when serializing XMP.

©Adobe Systems Incorporated, 2012

7.9.2.2 rdf:Description with property attributes

Property and structure field elements that have normal (non-URI) simple, unqualified values may be replaced with attributes in the **rdf:Description** element. This also applies to the pseudo-structure for general qualifiers, including the **rdf:value** element. The element and attribute forms may be mixed.

EXAMPLE Simple valued elements shortened to attributes:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/"
          xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"
          xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">

   <rdf:Description rdf:about="" xmp:Rating="3">

      <xmpTPg:MaxPageSize>
         <rdf:Description stDim:h="11.0" stDim:w="8.5">
            <!-- Best to use attributes for all, illustrates allowed mixing. -->
            <stDim:unit>inch</stDim:unit>
         </rdf:Description>
      </xmpTPg:MaxPageSize>

      <xmp:BaseURL>
         <rdf:Description xe:qualifier="artificial example">
            <rdf:value rdf:resource="http://www.adobe.com/"/>
         </rdf:Description>
      </xmp:BaseURL>

   </rdf:Description>

</rdf:RDF>
```

7.9.2.3 rdf:parseType="Resource" attribute

The **rdf:Description** element within a structure element may be replaced with an **rdf:parseType="Resource"** attribute in the structure element. This also applies to the pseudo-structure for general qualifiers.

EXAMPLE Structure replacing **rdf:Description** with **rdf:resource** attribute:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"
          xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
          xmlns:xmp="http://ns.adobe.com/xap/1.0/"
          xmlns:xe="http://ns.adobe.com/xmp-example/">

   <rdf:Description rdf:about="">

      <xmpTPg:MaxPageSize rdf:parseType="Resource">
         <stDim:h>11.0</stDim:h>
         <stDim:w>8.5</stDim:w>
         <stDim:unit>inch</stDim:unit>
      </xmpTPg:MaxPageSize>

      <xmp:BaseURL rdf:parseType="Resource">
         <rdf:value rdf:resource="http://www.adobe.com/"/>
         <xe:qualifier>artificial example</xe:qualifier>
      </xmp:BaseURL>
```

©Adobe Systems Incorporated, 2012

```
   </rdf:Description>

</rdf:RDF>
```

7.9.2.4 Structure element with field attributes

If all fields of a structure have normal (non-URI) simple, unqualified values, the fields may be written as attributes of the structure element instead of using a nested **rdf:Description** element and field elements within that. The structure element in this case shall have empty element content. All fields of a structure shall be written in the same manner, either as nested elements or as attributes. This shorthand may also be applied to the pseudo-structure for general qualifiers. This use of field attributes shall not be mixed with the **rdf:parseType="Resource"** attribute.

EXAMPLE Structure with all simple fields as attributes:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"
          xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#">

   <rdf:Description rdf:about="">
      <xmpTPg:MaxPageSize stDim:h="11.0" stDim:w="8.5" stDim:unit="inch"/>
   </rdf:Description>

</rdf:RDF>
```

7.9.2.5 RDF Typed Nodes

RDF has a notation called Typed Nodes that allows an arbitrarily named element to appear where an **rdf:Description** element is expected. This is equivalent to having an **rdf:Description** element instead with a nested **rdf:type** element. All other attributes and nested element content are retained in the replacement **rdf:Description** element. The added **rdf:type** element has empty element content and one attribute. That attribute is an **rdf:resource** attribute whose value is the concatenation of the original element's namespace URI and local name.

EXAMPLE 1 The following two snippets are equivalent in general RDF:

```
<!-- Assume rdf: namespace and xmlns:xe="http://ns.adobe.com/xmp-example/". -->

<xe:myType>
   <!-- Arbitrary other RDF. -->
</xe:myType>

<rdf:Description>
   <rdf:type rdf:resource="http://ns.adobe.com/xmp-example/myType"/>
   <!-- Same other RDF as above. -->
</rdf:Description>
```

NOTE 1 This conversion of the XML qualified name to the **rdf:type** value is another motivation for the previous recommendation (6.2, "XMP names") to terminate namespace URIs with a character that is not part of an XML NCName. Conversion in the other direction is problematic. Because the **rdf:type** value is a literal, the associated namespace URI might not even be defined. A possibly unfounded presumption would be made about where to separate the namespace URI and the local name.

The use of Typed Nodes is restricted in XMP and carries different semantics from the RDF textual equivalence. The **rdf:Bag**, **rdf:Seq**, and **rdf:Alt** elements used for XMP arrays are in fact uses of RDF Typed Nodes; their use in XMP is defined in 7.9.3.2, "Arrays not using Typed Node form".

NOTE 2 The **rdf:Description** element is used as a container in three ways in XMP: for properties, for structure fields, and for general qualifiers. These are the places where the use of Typed Nodes requires definition.

©Adobe Systems Incorporated, 2012

Top-level typed nodes, immediately within the **rdf:RDF** element, shall not be used in XMP. An **rdf:type** property may be explicitly used in XMP.

EXAMPLE 2 Top-level Typed Node and **rdf:type** property:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">

   <!-- A Typed Node immediately within rdf:RDF is not allowed in XMP. -->
   <xe:myType>
      <!-- Arbitrary other XMP. -->
   </xe:myType>

   <!-- A top-level rdf:type property is allowed in XMP. -->
   <rdf:Description>
      <rdf:type rdf:resource="http://ns.adobe.com/xmp-example/myType"/>
      <!-- Same other XMP as above. -->
   </rdf:Description>

</rdf:RDF>
```

The use of an inner Typed Node in XMP shall attach an **rdf:type** qualifier to the containing element. The value of the **rdf:type** qualifier shall be a URI consisting of the Typed Node element's namespace URI concatenated with the local name. The remaining interpretation shall be as though an **rdf:Description** element were used instead of the original Typed Node element.

NOTE 3 The addition of the **rdf:type** qualifier will significantly alter the serialization of the equivalent XMP interpretation of the Typed Node. The XMP processing is not a simple textual replacement like RDF.

EXAMPLE 3 Inner Typed Node interpretation in XMP:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">
   <rdf:Description>

      <!-- A Typed Node used for a structure value. -->
      <xe:Prop1>
         <xe:myType>
            <xe:Field>value</xe:Field>
         </xe:myType>
      </xe:Prop1>

      <!-- This is equivalent XMP to the above Typed Node. -->
      <xe:Prop2>
         <rdf:Description>
            <rdf:value rdf:parseType="Resource">
               <xe:Field>value</xe:Field>
            </rdf:value>
            <rdf:type rdf:resource="http://ns.adobe.com/xmp-example/myType"/>
         </rdf:Description>
      </xe:Prop2>

   </rdf:Description>
</rdf:RDF>
```

An explicit structure field named **rdf:type** may be used in XMP.

EXAMPLE 4 An explicit **rdf:type** field in an XMP structure:

©Adobe Systems Incorporated, 2012

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:xe="http://ns.adobe.com/xmp-example/">
   <rdf:Description>

      <!-- In XMP this is not related to the Typed Node usage. -->
      <xe:Prop3>
         <rdf:Description>
            <xe:Field>value</xe:Field>
            <rdf:type rdf:resource="http://ns.adobe.com/xmp-example/myType"/>
         </rdf:Description>
      </xe:Prop3>

   </rdf:Description>
</rdf:RDF>
```

7.9.3 Prohibited equivalent RDF

7.9.3.1 Summary

The following equivalent forms of RDF shall not be used when serializing XMP.

7.9.3.2 Arrays not using Typed Node form

As noted in 7.9.2.5, “RDF Typed Nodes”, the canonical RDF for an XMP array uses RDF Typed Node notation. The **rdf:Bag**, **rdf:Seq**, and **rdf:Alt** elements are in fact an example of RDF Typed Node syntax. These three elements shall be used for XMP arrays. The equivalent RDF using **rdf:Description** and **rdf:type** elements shall not be used in XMP.

7.9.3.3 rdf:_n elements and attributes

RDF allows a sequence of **rdf:li** elements to be replaced with elements of the form **rdf:_1**, **rdf:_2**, and so on. Because these are unique names, element-to-attribute substitution similar to 7.9.2.2, “rdf:Description with property attributes”, is also allowed in general RDF. The **rdf:li** element shall be used in XMP and the **rdf:_n** form shall not be used in XMP.

8 Core properties

8.1 Overview

This clause defines a collection of XMP properties that have broad applicability across domains of usage and digital file formats, along with data types that are used to represent values of these properties. The XMP names listed here originated in other standards and specifications, as is reflected in the namespace URIs using domains owned by those originators. This document does not claim ownership over those namespaces. Their respective owners may define additional XMP names in those namespaces, whether properties, structure fields, or qualifiers.

NOTE Unless local conditions dictate otherwise, XMP processors are encouraged to support the unrestricted use of XMP properties in these and other namespaces. There is no intent that the properties defined here be the only allowed ones.

©Adobe Systems Incorporated, 2012

8.2 Core value types

8.2.1 Basic value types

8.2.1.1 Boolean

Boolean values shall be "True" or "False".

8.2.1.2 Date

A date-time value is represented using a subset of the formats as defined in Date and Time Formats:

```
YYYY
YYYY-MM
YYYY-MM-DD
YYYY-MM-DDThh:mmTZD
YYYY-MM-DDThh:mm:ssTZD
YYYY-MM-DDThh:mm:ss.sTZD
```

In which:

- `YYYY` = four-digit year
- `MM` = two-digit month (01=January)
- `DD` = two-digit day of month (01 to 31)
- `hh` = two digits of hour (00 to 23)
- `mm` = two digits of minute (00 to 59)
- `ss` = two digits of second (00 to 59)
- `s` = one or more digits representing a decimal fraction of a second
- `TZD` = time zone designator (Z or +hh:mm or -hh:mm)

The time zone designator need not be present in XMP. When not present, the time zone is unknown, and an XMP processor should not assume anything about the missing time zone.

Local time-zone designators `+hh:mm` or `-hh:mm` should be used when possible instead of converting to UTC.

NOTE If a file was saved at noon on October 23, a timestamp of `2004-10-23T12:00:00-06:00` conveys more information than `2004-10-23T18:00:00Z`.

8.2.1.3 Integer

A signed or unsigned numeric string used as an integer number representation. The string consists of an arbitrary-length decimal numeric string with an optional leading “+” or “–” sign.

8.2.1.4 Real

A simple text value denoting a floating-point numeric value, written using decimal notation of an optional sign followed by an integer part and a fraction part. Either the integer part or the fraction part, but not both, may be omitted. The sign, if present, is "+" (U+002B) or "-" (U+002D). The integer part, if present, is a sequence of one or more decimal digits (U+0030 to U+0039). The fraction, if present, is a decimal point (".", U+002E) followed by a sequence of one or more decimal digits.

The precise range and precision for the general type are not specified by this document. If converted to a binary value, an XMP processor shall support at least the 32-bit IEEE 754 range and precision, and it should

©Adobe Systems Incorporated, 2012

support at least the 64-bit IEEE 754 range and precision. A particular use of the Real type may specify a required range or precision, such as nonnegative or microsecond resolution (for a duration in seconds).

8.2.1.5 Text

A possibly empty Unicode string.

8.2.2 Derived value types

8.2.2.1 AgentName

The name of an XMP processor, a Text value.

It is recommended that the value use this format convention:

```
Organization Software_name Version (token;token;...)
```

- *Organization*: The name of the company or organization providing the software, no SPACEs.
- *Software_name*: The full name of the software, SPACEs allowed.
- *version*: The version of the software, no SPACEs.
- *tokens*: Can be used to identify an operating system, plug-in, or more detailed version information.

EXAMPLE "Adobe Acrobat 9.0 (Mac OS X 10.5)"

8.2.2.2 Choice

A value chosen from a *vocabulary* of values. Vocabularies provide a means of specifying a limited and possibly extensible set of values for a property.

A choice can be *open* or *closed*:

- An open choice has one or more lists of preferred values, but other values can be used freely.
- A closed choice has one or more lists of allowed values, other values shall not be used.

NOTE An XMP reader would be more robust if it tolerated unexpected values for closed choice types when the set of allowed values can be expected to grow over time.

8.2.2.3 GUID

A string representing a “**g**lobally **u**nique **id**entifier”. A GUID shall be a normal (non-URI) simple value, even though it might appear similar to a URI string. This document does not require any particular methodology for creating a GUID, nor does it require any specific means of formatting the GUID as a simple XMP value. The only valid operations on GUIDs are to create them, to assign one to another, and to compare two of them for equality. This comparison shall use the Unicode string value as-is, using a direct byte-for-byte check for equality.

8.2.2.4 Language Alternative

An alternative array of simple text items. Language alternatives facilitate the selection of a simple text item based on a desired language. Each array item shall have an **xml:lang** qualifier. Each **xml:lang** value shall be unique among the items. As defined in IETF RFC 3066, the **xml:lang** value is composed of one or more parts: a primary language subtag and a (possibly empty) series of subsequent subtags. The same primary subtag may be used alone and in conjunction with one or more lower-level subtags. A default value, if known, should be the first array item. The order of other array items is not specified by this document.

©Adobe Systems Incorporated, 2012

An **xml:lang** value of "x-default" may be used to explicitly denote a default item. If used, the "x-default" item shall be first in the array and its simple text value should be repeated in another item in which **xml:lang** specifies its actual language. However, an "x-default" item may be the only item, in which case there is only a default value in no defined language.

EXAMPLE 1 Language alternative with an "x-default" item:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="">

<!-- Line wrapping of rdf:li elements is for presentation in this example. -->
<!-- Leading and trailing white space is part of the array item values. -->

       <dc:title>
          <rdf:Alt>
             <rdf:li xml:lang="x-default">
                XMP - Extensible Metadata Platform
             </rdf:li>
             <rdf:li xml:lang="en-us">
                XMP - Extensible Metadata Platform
             </rdf:li>
             <rdf:li xml:lang="fr">
                XMP - Une Platforme Extensible pour les Metadonneés
             </rdf:li>
          </rdf:Alt>
       </dc:title>

   </rdf:Description>

</rdf:RDF>
```

8.2.2.5 Locale

A simple text value denoting a language code as defined in IETF RFC 3066.

8.2.2.6 MIMEType

A simple text value denoting a digital file format as defined in IETF RFC 2046.

8.2.2.7 ProperName

A simple text value denoting the name of a person or organization.

8.2.2.8 RenditionClass

A simple text Open Choice value denoting the form or intended usage of a resource. A series of colon-separated (":", U+003A) tokens and parameters, the first of which names the basic usage of the rendition. Additional tokens need not be present; they provide specific characteristics of the rendition. Table 2 lists defined values.

NOTE See definitions of *rendition* (3.7) and *version* (3.9).

©Adobe Systems Incorporated, 2012

.

Table 2 — Defined values for rendition tokens

Token	Defined value
`default`	The master resource; no additional tokens allowed.
`draft`	A review rendition.
`low-res`	A low-resolution, full-size stand-in.
`proof`	A review proof.
`screen`	Screen resolution or Web rendition.
`thumbnail`	A simplified or reduced preview. Additional tokens can provide characteristics. The recommended order is: `thumbnail`:*format*:*size*:*colorspace*. EXAMPLE `thumbnail:jpeg`, `thumbnail:16x16`, `thumbnail:gif:8x8:bw`.

8.2.2.9 ResourceRef

A structure denoting a multiple-component reference to a resource. The field values are taken from various properties in the referenced resource.

- The field namespace URI shall be "http://ns.adobe.com/xap/1.0/sType/ResourceRef#".
- The preferred field namespace prefix is **stRef**.

Table 3 lists the fields available in ResourceRef. Fields need not be present. The fields, if used, shall be of the specified types. The field content should be as described.

Table 3 — ResourceRef fields

Name	Type	Field content
stRef:documentID	GUID	The value of the **xmpMM:DocumentID** property from the referenced resource.
stRef:filePath	URI	The referenced resource's file path or URL.
stRef:instanceID	GUID	The value of the **xmpMM:InstanceID** property from the referenced resource. NOTE The difference in capitalization between **stRef:documentID** and **xmpMM:DocumentID** is real, the result of historical accident. This is also true of the other ResourceRef fields.
stRef:renditionClass	RenditionClass	The value of the **xmpMM:RenditionClass** property from the referenced resource.
stRef:renditionParams	Text	The value of the **xmpMM:RenditionParams** property from the referenced resource.

8.2.2.10 URI

Text denoting an Internet Uniform Resource Identifier as defined in IETF RFC 3986.

8.2.2.11 URL

Text denoting an Internet Uniform Resource Locator as defined in URIs, URLs, and URNs: Clarifications and Recommendations.

©Adobe Systems Incorporated, 2012

8.3 Dublin Core namespace

The Dublin Core namespace provides a set of commonly used properties. The names and usage shall be as defined in the Dublin Core Metadata Element Set, created by the Dublin Core Metadata Initiative (DCMI).

- The namespace URI shall be "http://purl.org/dc/elements/1.1/".
- The preferred namespace prefix is **dc**.

NOTE 1 The Dublin Core elements as defined by DCMI all have URIs of the form "http://purl.org/dc/elements/1.1/<*name*>" where the <*name*> part differs.

The Dublin Core elements are defined in XMP as properties using the namespace URI "http://purl.org/dc/elements/1.1/"; the local names are the leaf part of the DCMI URI.

The XMP data modelling of these is consistent with the apparent Dublin Core intent, but specific to XMP.

As a corollary of the data modelling, the RDF serialization of Dublin Core in XMP might not exactly match other RDF usage of the Dublin Core element set.

XMP does not “include Dublin Core” in any fuller sense.

Table 4 lists properties in the Dublin Core namespace. The properties, if used, shall be of the specified types. The property content should be as described.

NOTE 2 In Table 4 the property content has subsections for the DCMI definition and comment, plus an XMP addition. The DCMI definition and comment text come directly from the Dublin Core Metadata Element Set. The XMP addition is specific to this document.

Table 4 — Dublin Core properties

Name	Type	Property content
dc:contributor	Unordered array of ProperName	**DCMI definition:** An entity responsible for making contributions to the resource. **DCMI comment:** Examples of a contributor include a person, an organization, or a service. Typically, the name of a contributor should be used to indicate the entity. **XMP addition:** XMP usage is a list of contributors. These contributors should not include those listed in **dc:creator**.
dc:coverage	Text	**DCMI definition:** The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant. **XMP addition:** XMP usage is the extent or scope of the resource.
dc:creator	Ordered array of ProperName	**DCMI definition:** An entity primarily responsible for making the resource. **DCMI comment:** Examples of a creator include a person, an organization, or a service. Typically, the name of a creator should be used to indicate the entity. **XMP addition:** XMP usage is a list of creators. Entities should be listed in order of decreasing precedence, if such order is significant.
dc:date	Ordered array of Date	**DCMI definition:** A point or period of time associated with an event in the life cycle of the resource.
dc:description	Language Alternative	**DCMI definition:** An account of the resource. **XMP addition:** XMP usage is a list of textual descriptions of the content of the resource, given in various languages.

©Adobe Systems Incorporated, 2012

Table 4 — Dublin Core properties *(continued)*

Name	Type	Property content
dc:format	MIMEType	**DCMI definition:** The file format, physical medium, or dimensions of the resource. **DCMI comment:** Examples of dimensions include size and duration. Recommended best practice is to use a controlled vocabulary such as the list of Internet Media Types [MIME]. **XMP addition:** XMP usage is a MIME type. Dimensions would be stored using a media-specific property, beyond the scope of this document.
dc:identifier	Text	**DCMI definition:** An unambiguous reference to the resource within a given context. **DCMI comment:** Recommended best practice is to identify the resource by means of a string conforming to a formal identification system.
dc:language	Unordered array of Locale	**DCMI definition:** A language of the resource. **XMP addition:** XMP usage is a list of languages used in the content of the resource.
dc:publisher	Unordered array of ProperName	**DCMI definition:** An entity responsible for making the resource available. **DCMI comment:** Examples of a publisher include a person, an organization, or a service. Typically, the name of a publisher should be used to indicate the entity. **XMP addition:** XMP usage is a list of publishers.
dc:relation	Unordered array of Text	**DCMI definition:** A related resource. **DCMI comment:** Recommended best practice is to identify the related resource by means of a string conforming to a formal identification system. **XMP addition:** XMP usage is a list of related resources.
dc:rights	Language Alternative	**DCMI definition:** Information about rights held in and over the resource. **DCMI comment:** Typically, rights information includes a statement about various property rights associated with the resource, including intellectual property rights. **XMP addition:** XMP usage is a list of informal rights statements, given in various languages.
dc:source	Text	**DCMI definition:** A related resource from which the described resource is derived. **DCMI comment:** The described resource may be derived from the related resource in whole or in part. Recommended best practice is to identify the related resource by means of a string conforming to a formal identification system.
dc:subject	Unordered array of Text	**DCMI definition:** The topic of the resource. **DCMI comment:** Typically, the subject will be represented using keywords, key phrases, or classification codes. Recommended best practice is to use a controlled vocabulary. To describe the spatial or temporal topic of the resource, use the **dc:coverage** element. **XMP addition:** XMP usage is a list of descriptive phrases or keywords that specify the content of the resource.

©Adobe Systems Incorporated, 2012

Table 4 — Dublin Core properties *(continued)*

Name	Type	Property content
dc:title	Language Alternative	**DCMI definition:** A name given to the resource. **DCMI comment:** Typically, a title will be a name by which the resource is formally known. **XMP addition:** XMP usage is a title or name, given in various languages.
dc:type	Unordered array of Text	**DCMI definition:** The nature or genre of the resource. **DCMI comment:** Recommended best practice is to use a controlled vocabulary such as the DCMI Type Vocabulary [DCMITYPE]. To describe the file format, physical medium, or dimensions of the resource, use the **dc:format** element. **XMP addition:** See the **dc:format** entry for clarification of the XMP usage of that element.

8.4 XMP namespace

The XMP basic namespace contains properties that provide basic descriptive information.

- The namespace URI shall be "http://ns.adobe.com/xap/1.0/".
- The preferred namespace prefix is **xmp**.

Table 5 lists properties in the XMP namespace. The properties, if used, shall be of the specified types. The property content should be as described in Table 5.

Table 5 — Properties in the XMP namespace

Name	Type	Property content
xmp:CreateDate	Date	The date and time the resource was created. For a digital file, this need not match a file-system creation time. For a freshly created resource, it should be close to that time, modulo the time taken to write the file. Later file transfer, copying, and so on, can make the file-system time arbitrarily different.
xmp:CreatorTool	AgentName	The name of the first known tool used to create the resource.
xmp:Identifier	Unordered array of Text	An unordered array of text strings that unambiguously identify the resource within a given context. An array item may be qualified with **xmpidq:Scheme** (see 8.7, “xmpidq namespace”) to denote the formal identification system to which that identifier conforms. NOTE The **xmp:Identifier** property was added because **dc:identifier** has been defined in the original XMP specification as a single identifier instead of as an array, and changing **dc:identifier** to an array would break compatibility with existing XMP processors.
xmp:Label	Text	A word or short phrase that identifies a resource as a member of a user-defined collection. NOTE One anticipated usage is to organize resources in a file browser.
xmp:MetadataDate	Date	The date and time that any metadata for this resource was last changed. It should be the same as or more recent than **xmp:ModifyDate**.

©Adobe Systems Incorporated, 2012

Table 5 — Properties in the XMP namespace *(continued)*

Name	Type	Property content
xmp:ModifyDate	Date	The date and time the resource was last modified. NOTE The value of this property is not necessarily the same as the file's system modification date because it is typically set before the file is saved.
xmp:Rating	Closed Choice of Real	A user-assigned rating for this file. The value shall be -1 or in the range [0..5], where -1 indicates "rejected" and 0 indicates "unrated". If **xmp:Rating** is not present, a value of 0 should be assumed. NOTE Anticipated usage is for a typical "star rating" UI, with the addition of a notion of rejection.

8.5 XMP Rights Management namespace

The XMP Rights Management namespace contains properties that provide information regarding the legal restrictions associated with a resource.

- The namespace URI shall be "http://ns.adobe.com/xap/1.0/rights/".
- The preferred namespace prefix is **xmpRights**.

NOTE These XMP properties are intended to provide a means of rights expression. They are not intended to provide digital rights management (DRM) controls.

Table 6 lists XMP Rights Management properties. The properties, if used, shall be of the specified types. The property content should be as described.

Table 6 — Properties in the XMP Rights Management namespace

Name	Type	Property content
xmpRights:Certificate	Text	A Web URL for a rights management certificate. NOTE This is a normal (non-URI) simple value because of historical usage.
xmpRights:Marked	Boolean	When true, indicates that this is a rights-managed resource. When false, indicates that this is a public-domain resource. Omit if the state is unknown.
xmpRights:Owner	Unordered array of ProperName	A list of legal owners of the resource.
xmpRights:UsageTerms	Language Alternative	A collection of text instructions on how a resource can be legally used, given in a variety of languages.
xmpRights:WebStatement	Text	A Web URL for a statement of the ownership and usage rights for this resource. NOTE This is a normal (non-URI) simple value because of historical usage.

8.6 XMP Media Management namespace

The XMP Media Management namespace contains properties that provide information regarding the identification, composition, and history of a resource.

- The namespace URI shall be "http://ns.adobe.com/xap/1.0/mm/".
- The preferred namespace prefix is **xmpMM**.

©Adobe Systems Incorporated, 2012

Table 7 lists XMP Media Management properties. The properties, if used, shall be of the specified types. The property content should be as described.

Table 7 — XMP Media Management properties

Name	Type	Property content
xmpMM:DerivedFrom	ResourceRef	A reference to the resource from which this one is derived. This should be a minimal reference, in which missing components can be assumed to be unchanged. See definitions of *rendition* (3.7) and *version* (3.9). NOTE A rendition might need to specify only the **xmpMM:InstanceID** and **xmpMM:RenditionClass** of the original.
xmpMM:DocumentID	GUID	The common identifier for all versions and renditions of a resource. See Annex A, "(informative) Document and instance IDs" and definitions of *rendition* (3.7) and *version* (3.9).
xmpMM:InstanceID	GUID	An identifier for a specific incarnation of a resource, updated each time a file is saved. See Annex A, "(informative) Document and instance IDs".
xmpMM:OriginalDocumentID	GUID	The common identifier for the original resource from which the current resource is derived. For example, if you save a resource to a different format, then save that one to another format, each save operation should generate a new **xmpMM:DocumentID** that uniquely identifies the resource in that format, but should retain the ID of the source file here. See Annex A, "(informative) Document and instance IDs".
xmpMM:RenditionClass	RenditionClass	The rendition class name for this resource. This property should be absent or set to `default` for a resource that is not a derived rendition. See definitions of *rendition* (3.7) and *version* (3.9).
xmpMM:RenditionParams	Text	Can be used to provide additional rendition parameters that are too complex or verbose to encode in **xmpMM:RenditionClass**.

8.7 xmpidq namespace

The **xmpidq** namespace contains a single qualifier that defines the scheme used in the **xmp:Identifier** array.

- The namespace URI shall be "http://ns.adobe.com/xmp/Identifier/qual/1.0/".
- The preferred namespace prefix is **xmpidq**.

©Adobe Systems Incorporated, 2012

Table 8 lists the single **xmpidq** qualifier. The qualifier shall be of the specified type. The qualifier content should be as described.

Table 8 — XMP xmpidq qualifier

Name	Type	Qualifier content
xmpidq:Scheme	Text	A qualifier providing the name of the formal identification scheme used for an item in the **xmp:Identifier** array.

©Adobe Systems Incorporated, 2012

Annex A
(informative)

Document and instance IDs

There can often be ambiguity when referring to resources. The contents of a resource can change over time. Depending on the situation, it might be desirable to refer to either:

- a specific state of the resource as it exists at a point in time, or
- the resource in general, as a persistent container whose content can change.

Some characteristics of a resource (such as the application that created it) are normally expected to be persistent over its life. Other characteristics (such as word count) are expected to change as the content of the resource changes. Some characteristics (such as copyright information or authors' names) might or might not change.

In the same way, XMP properties that represent such characteristics of a resource are inherently ambiguous as to whether they refer to the current content of a resource or to the resource in general. XMP itself provides no mechanisms for distinguishing these.

This document defines three GUIDs that are intended to help manage copies of a resource, to identify a specific state when desired, and to associate related copies of the same conceptual resource. These are not the only items in XMP that are intended to help resource management, but they are important ones. The XMP Media Management namespace (8.6, "XMP Media Management namespace") defines these properties:

- **xmpMM:DocumentID**: Created once for new resources. Different renditions are expected to have different values for **xmpMM:DocumentID**.
- **xmpMM:InstanceID**: Changes with each save operation.
- **xmpMM:OriginalDocumentID**: Links a resource to its original source. For example, when you save a PSD document as a JPEG, then convert the JPEG to GIF format, the immediate source of the GIF is the JPEG, and the original source is the PSD. The value of **xmpMM:OriginalDocumentID** in both the JPEG and GIF files is the value of **xmpMM:DocumentID** from the original PSD file.

In addition, the **xmpMM:DerivedFrom** property is defined to store linkage information from one resource to its ancestors, possibly including these GUIDs.

The use of robust GUIDs is encouraged; having globally unique values is important. In practical terms, this means that the probability of a collision is so remote as to be effectively impossible. Typically, 128-bit or 144-bit numbers are used, encoded as hexadecimal strings.

This document does not require any particular methodology for creating a GUID, nor does it require any specific means of formatting the GUID as a simple XMP value.

The only valid operations on XMP IDs are to create them, to assign one to another, and to compare two of them for equality. Comparisons use the Unicode string value as-is, using a direct byte-for-byte check for equality.

IETF RFC 4122 (http://www.ietf.org/rfc/rfc4122.txt) describes ways to create and format GUID strings. For privacy, the use of a MAC address is not recommended. See section 4.1.6 of RC 4122 for details and alternatives.

©Adobe Systems Incorporated, 2012

©Adobe Systems Incorporated, 2012

Annex B
(informative)

Implementation guidance

B.1 General

This annex contains informative text about a variety of implementation issues facing XMP processors. It provides informal guidance on a variety of separate topics. These generally clarify some smaller aspects of the mappings between the XMP data model and RDF.

B.2 Escaping XML markup in values

The following sections of the Extensible Markup Language specification discuss the treatment of special characters used in element character data content or attribute values:

- Section 2.1, “Well-Formed XML Documents”
- Section 2.2, “Characters”
- Section 2.4, “Character Data and Markup”
- Section 2.11, “End-of-Line Handling”
- Section 3.3.3, “Attribute-Value Normalization”
- Section 4, “Physical Structures”
- Appendix D, “Expansion of Entity and Character References (Non-Normative)”

These rules require that certain characters in XMP values be escaped on output.

The rules from section 2.4 reduce to escaping of "&", "<", ">", and the other characters in the RestrictedChar set. Use of CDATA sections is discouraged in XMP; there is no way to escape the presence of "]]>" in a value.

The rules from section 2.4 prohibit all ASCII controls (U+000..U+001F) except for tab (U+0009), linefeed (U+000A), and carriage return (U+000D). The prohibited controls cannot even appear as character entities.

B.3 Namespace URI termination

This section expands on the namespace URI termination issues mentioned in 6.2, “XMP names”.

The formal definition of RDF transforms the XML representation into “triples” in a manner that concatenates XML namespace URI strings with the local part of XML element and attribute names. This can lead to ambiguities if the URI does not end in a separating character that cannot appear in the local name. This is not a problem for an XMP processor that avoids use of the RDF triple representation. But it could be a problem in other implementations of XMP, or if the RDF form of XMP were fed to a traditional RDF processor.

EXAMPLE Here is an artificial example of RDF that produces ambiguities in the triples:

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:ns1="http://ns.adobe.com/xmp-example/namespace"
          xmlns:ns2="http://ns.adobe.com/xmp-example/name">
   <rdf:Description
          rdf:about="http://ns.adobe.com/xmp-example/RDF-predicate-collision">
      <ns1:ship>value of ns1:ship</ns1:ship>
```

©Adobe Systems Incorporated, 2012

```
        <ns2:spaceship>value of ns2:spaceship</ns2:spaceship>
    </rdf:Description>
</rdf:RDF>
```

Here are the ambiguous RDF triples from the RDF Validator (http://www.w3.org/RDF/Validator/). Notice that the two predicates are the same:

```
Subject: http://ns.adobe.com/xmp-example/RDF-predicate-collision
Predicate: http://ns.adobe.com/xmp-example/namespaceship
Object: "value of ns1:ship"

Subject: http://ns.adobe.com/xmp-example/RDF-predicate-collision
Predicate: http://ns.adobe.com/xmp-example/namespaceship
Object: "value of ns2:spaceship"
```

B.4 Case-neutral xml:lang values

The values of **xml:lang** qualifiers, and some standard XMP properties, obey the rules for language identifiers given in IETF RFC 3066. Implementers are encouraged to pay particular attention to these aspects of IETF RFC 3066:

- Both two-letter and three-letter primary subtags as defined by ISO 639-1 and ISO 639-2 are supported.
- When a language has both an ISO 639-1 two-character code and an ISO 639-2 three-character code, the tag derived from the ISO 639-1 two-character code is used.
- All tags are treated as case-insensitive; there exist conventions for capitalization of some of them, but case is not allowed to carry meaning. For instance, ISO 3166 recommends that country codes be capitalized (MN Mongolia), while ISO 639 recommends that language codes be written in lower case (mn Mongolian).

Since the values are required to be treated as case-insensitive, XMP processors are allowed to normalize them on input and to output the normalized values. The recommendations of ISO 639 and ISO 3166 are preferred. This document does not define or require a normalization policy. Since comparisons are case-insensitive, differences in policy can have no substantive effect.

©Adobe Systems Incorporated, 2012

Annex C
(informative)

RDF parsing information

C.1 General

This annex presents the formal grammar for RDF and a walkthrough for implementing a top-down parser that recognizes the XMP subset of RDF. The parsing description is based on the grammar in section 7, “RDF/XML Grammar”, of the RDF/XML Syntax Specification.

```
7.2.2 coreSyntaxTerms
   rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
      rdf:resource | rdf:nodeID | rdf:datatype

7.2.3 syntaxTerms
   coreSyntaxTerms | rdf:Description | rdf:li

7.2.4 oldTerms
   rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID

7.2.5 nodeElementURIs
   anyURI - ( coreSyntaxTerms | rdf:li | oldTerms )

7.2.6 propertyElementURIs
   anyURI - ( coreSyntaxTerms | rdf:Description | oldTerms )

7.2.7 propertyAttributeURIs
   anyURI - ( coreSyntaxTerms | rdf:Description | rdf:li | oldTerms )

7.2.8 doc
   root ( document-element == RDF, children == list ( RDF ) )

7.2.9 RDF
   start-element ( URI == rdf:RDF, attributes == set() )
      nodeElementList
   end-element()

7.2.10 nodeElementList
   ws* ( nodeElement ws* )*

7.2.11 nodeElement
   start-element ( URI == nodeElementURIs,
       attributes == set ( ( idAttr | nodeIdAttr | aboutAttr )?,
                  propertyAttr* ) )
      propertyEltList
   end-element()

7.2.12 ws
   A text event matching white space defined by [XML] definition
   White Space Rule [3] S in section Common Syntactic Constructs.

7.2.13 propertyEltList
   ws* ( propertyElt ws* )*
```

©Adobe Systems Incorporated, 2012

```
7.2.14 propertyElt
   resourcePropertyElt | literalPropertyElt |
   parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt |
   parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt |
   emptyPropertyElt

7.2.15 resourcePropertyElt
   start-element ( URI == propertyElementURIs, attributes == set ( idAttr? ) )
      ws* nodeElement ws*
   end-element()

7.2.16 literalPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, datatypeAttr?) )
      text()
   end-element()

7.2.17 parseTypeLiteralPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseLiteral ) )
   literal
   end-element()

7.2.18 parseTypeResourcePropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseResource ) )
      propertyEltList
   end-element()

7.2.19 parseTypeCollectionPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseCollection ) )
      nodeElementList
   end-element()

7.2.20 parseTypeOtherPropertyElt
   start-element ( URI == propertyElementURIs, attributes
                   == set ( idAttr?, parseOther ) )
      propertyEltList
   end-element()

7.2.21 emptyPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, ( resourceAttr | nodeIdAttr )?,
                   propertyAttr* ) )
   end-element()

7.2.22 idAttr
   attribute ( URI == rdf:ID, string-value == rdf-id )

7.2.23 nodeIdAttr
   attribute ( URI == rdf:nodeID, string-value == rdf-id )

7.2.24 aboutAttr
   attribute ( URI == rdf:about, string-value == URI-reference )

7.2.25 propertyAttr
   attribute ( URI == propertyAttributeURIs, string-value == anyString )
```


©Adobe Systems Incorporated, 2012

```
7.2.26 resourceAttr
   attribute ( URI == rdf:resource, string-value == URI-reference )

7.2.27 datatypeAttr
   attribute ( URI == rdf:datatype, string-value == URI-reference )

7.2.28 parseLiteral
   attribute ( URI == rdf:parseType, string-value == "Literal")

7.2.29 parseResource
   attribute ( URI == rdf:parseType, string-value == "Resource")

7.2.30 parseCollection
   attribute ( URI == rdf:parseType, string-value == "Collection")

7.2.31 parseOther
   attribute ( URI == rdf:parseType,
               string-value == anyString - ("Resource" | "Literal" |
               "Collection") )

7.2.32 URI-reference
   An RDF URI Reference.

7.2.33 literal
   Any XML element content that is allowed according to [XML] definition
   Content of Elements Rule [43] content in section 3.1 Start-Tags, End-Tags, and
   Empty-Element Tags.

7.2.34 rdf-id
   An attribute string-value matching any legal [XML-NS] token NCName.
```

C.2 Top-down parsing of RDF

C.2.1 Overview

Here is a sample description of the desired RDF parsing support. This covers all forms for the RDF, the XMP canonical form, and all alternatives. The description presumes an initial raw XML parse that creates a runtime data structure of the XML. As a simplification, **xmlns** attributes are presumed to have been removed; the run-time data structure for the raw XML parse has propagated the namespace URIs.

The syntax and descriptions presented here are appropriate for construction of a top-down parser. They are not appropriate for a bottom-up parser. For example, there are significant ambiguities in the use of XML elements that are not RDF terms: they could be the Typed Node form of a nodeElement or one of the seven propertyElt forms.

The **xml:lang**, **rdf:about**, **rdf:ID**, **rdf:nodeID**, and **rdf:datatype** attributes are special in RDF. The use of **xml:lang** is not shown in the syntax productions. As mentioned in 7.8, “Qualifiers”, it is mapped to an XMP qualifier. The other attributes are specifically represented in the RDF syntax. Other than **rdf:about** for a top-level **rdf:Description** element, they are not allowed in XMP, as specified in 7.1, “General”.

C.2.2 Outermost element, rdf:RDF

```
7.2.9 RDF
   start-element ( URI == rdf:RDF, attributes == set() )
      nodeElementList
   end-element()
```

©Adobe Systems Incorporated, 2012

```
7.2.10 nodeElementList
   ws* ( nodeElement ws* )*

<rdf:RDF>
   ...
</rdf:RDF>
```

The outermost RDF element is **rdf:RDF**. Although optional in general RDF, the **rdf:RDF** element is required in XMP. The content of the **rdf:RDF** element represents a single XMP packet. No attributes are allowed on the **rdf:RDF** element. Inside **rdf:RDF** is a top-level nodeElementList, a sequence of zero or more white-space separated nodeElements with “top-level” restrictions.

C.2.3 Top-level and inner nodeElements

```
7.2.5 nodeElementURIs
   anyURI - ( coreSyntaxTerms | rdf:li | oldTerms )

7.2.11 nodeElement
   start-element ( URI == nodeElementURIs,
                   attributes == set ( ( idAttr | nodeIdAttr | aboutAttr )?,
                   propertyAttr* ) )
      propertyEltList
   end-element()

<rdf:RDF>
   <rdf:Description rdf:about="">  <!-- Top level rdf:Description nodeElement -->

      <ns:Struct1>
         <rdf:Description>  <!-- Inner rdf:Description nodeElement -->
            ...
         </rdf:Description>
      </ns:Struct1>

      <ns:Struct2>
         <ns:MyType>  <!-- Inner Typed Node form of nodeElement -->
            ...
         </ns:MyType>
      </ns:Struct2>

   </rdf:Description>
</rdf:RDF>
```

In the RDF syntax, a nodeElement (top-level or inner) can be an **rdf:Description** element, or any other element that is not an RDF term. Use of an RDF term element is an error. In XMP, a top-level nodeElement can only be **rdf:Description**. A nodeElement that is **rdf:Bag**, **rdf:Seq**, or **rdf:Alt** represents an XMP array value. Any other nodeElement that is not **rdf:Description** is a Typed Node, which is processed according to the canonical Typed Node expansion using **rdf:Description** and **rdf:type** shown in 7.9.2.5, “RDF Typed Nodes”.

C.2.4 Attributes of a nodeElement

```
7.2.7 propertyAttributeURIs
   anyURI - ( coreSyntaxTerms | rdf:Description | rdf:li | oldTerms )

7.2.11 nodeElement
   start-element ( URI == nodeElementURIs,
                   attributes == set ( ( idAttr | nodeIdAttr | aboutAttr )?,
                   propertyAttr* ) )
      propertyEltList
   end-element()
```


©Adobe Systems Incorporated, 2012

The attributes of a nodeElement can be **rdf:about**, **rdf:ID**, **rdf:nodeID**, or anything else that is not an RDF term. As specified in 7.4, “rdf:RDF and rdf:Description elements”, a top-level nodeElement in XMP is required to have an **rdf:about** attribute; the values of **rdf:about** attributes are all required to match. An XMP processor should allow a top-level nodeElement element to have no **rdf:about** attribute and treat this as identical to an **rdf:about** attribute with an empty value. For maximum compatibility with older files, an XMP processor might choose to allow a mix of empty and non-empty **rdf:about** values, but still require that all non-empty values match. The **rdf:about**, **rdf:ID**, and **rdf:nodeID** attributes are mutually exclusive in the RDF syntax. The **rdf:ID** and **rdf:nodeID** attributes are not allowed in XMP.

XMP does not allow an **xml:lang** attribute on a nodeElement.

Other attributes (propertyAttr) of a top-level nodeElement become simple unqualified properties in the XMP packet. Other attributes of an inner nodeElement become simple unqualified fields of the XMP struct value represented by the nodeElement, or become qualifiers if the nodeElement represents a value with general qualifiers, or a simple qualified value if the attribute is **rdf:value**.

C.2.5 Content of a nodeElement

```
7.2.11 nodeElement
   start-element ( URI == nodeElementURIs,
                   attributes == set ( ( idAttr | nodeIdAttr | aboutAttr )?,
                   propertyAttr* ) )
      propertyEltList
   end-element()

7.2.13 propertyEltList
   ws* ( propertyElt ws* )*

7.2.14 propertyElt
   resourcePropertyElt | literalPropertyElt |
   parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt |
   parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt |
   emptyPropertyElt
```

The contained elements of a nodeElement are a propertyEltList (property element list), a sequence of zero or more white-space separated propertyElts (property elements). The contained elements of a top-level nodeElement become properties in the XMP packet. The contained elements of an inner nodeElement become fields of the XMP struct value represented by the nodeElement, qualifiers or the qualified value if the nodeElement represents an XMP value with general qualifiers, or items in the XMP array value represented by the nodeElement. This pertains to the “valid” contained elements as defined in the rules for the various forms of a propertyElt.

The syntax of a propertyElt is somewhat complex. The various forms are not generally distinguished by their XML element name, but by attributes. Exceptions are resourcePropertyElt, literalPropertyElt, and emptyPropertyElt, which are distinguished by a combination of attributes and XML content. For those distinguished by attribute, **xml:lang** attributes cause some small complication. The use of **xml:lang** in RDF is special; it is not part of the syntax productions. An **xml:lang** attribute in the RDF always maps to an **xml:lang** qualifier in XMP.

The rules for distinguishing the propertyElt forms are:

- If there are more than three attributes (counting **xml:lang**), this is an emptyPropertyElt.
- Look for an attribute that is not **xml:lang** or **rdf:ID**.
- If none is found, look at the XML content of the propertyElt.
  - — If there is no content, this is an emptyPropertyElt.
  - — If the only content is character data, this is a literalPropertyElt.

©Adobe Systems Incorporated, 2012

  - Otherwise this is a resourcePropertyElt.
- Otherwise (if an attribute is found that is not **xml:lang** or **rdf:ID**):
  - If the attribute name is **rdf:datatype**, this is a literalPropertyElt.
  - If the attribute name is not **rdf:parseType**, this is an emptyPropertyElt.
  - If the attribute value is Literal, this is a parseTypeLiteralPropertyElt.
  - If the attribute value is Resource, this is a parseTypeResourcePropertyElt.
  - If the attribute value is Collection, this is a parseTypeCollectionPropertyElt.
- Otherwise, this is a parseTypeOtherPropertyElt.

The use of phrases such as “this is an emptyPropertyElt” in the preceding list means that the only applicable RDF syntax production has been identified. Further processing performs additional error checking to verify the specific syntax.

C.2.6 The resourcePropertyElt

```
7.2.15 resourcePropertyElt
   start-element ( URI == propertyElementURIs, attributes == set ( idAttr? ) )
      ws* nodeElement ws*
   end-element()

<ns:Struct>  <!-- resourcePropertyElt -->
   <rdf:Description>  <!-- nodeElement -->
      <ns:Field> ... </ns:Field>
      ...
   </rdf:Description>
</ns:Struct>

<ns:Array>  <!-- resourcePropertyElt -->
   <rdf:Bag>  <!-- nodeElement -->
      <rdf:li> ... </rdf:li>
      ...
   </rdf:Bag>
</ns:Array>

<ns:Prop>  <!-- resourcePropertyElt -->
   <rdf:Description>  <!-- nodeElement -->
      <rdf:value> ... </rdf:value>
      <ns:Qual> ... </ns:Qual>
      ...
   </rdf:Description>
</ns:Prop>
```

A resourcePropertyElt most commonly represents an XMP struct or array property. It can also represent a property with general qualifiers (other than **xml:lang** as an attribute). These are expressed in RDF as pseudo-structs with a special **rdf:value** “field”.

The **rdf:ID** attribute is not allowed in XMP.

A resourcePropertyElt can have an **xml:lang** attribute; it becomes an **xml:lang** qualifier on the XMP value represented by the resourcePropertyElt.

```
<ns:Prop>  <!-- resourcePropertyElt -->
   <ns:Type>  <!-- Typed Node form of a nodeElement -->
      ...
   </ns:Type>
</ns:Prop>
```


©Adobe Systems Incorporated, 2012

A resourcePropertyElt can contain an RDF Typed Node, a form of shorthand that elevates an **rdf:type** qualifier to a more visible position in the XML.

Note that the canonical array form used by XMP is in fact a Typed Node. Because of their common usage and known semantics, the array forms are more easily dealt with as direct special cases. The XMP treatment of Typed Nodes is discussed in 7.9.2.5, “RDF Typed Nodes”.

C.2.7 The literalPropertyElt

```
7.2.16 literalPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, datatypeAttr?) )
      text()
   end-element()

<ns:Prop>value</ns:Prop>  <!-- literalPropertyElt -->
```

A literalPropertyElt is the typical element form of a simple property. The text content is the property value. Attributes of the element become qualifiers in the XMP data model.

The **rdf:ID** and **rdf:datatype** attributes are not allowed in XMP.

A literalPropertyElt can have an **xml:lang** attribute. It becomes an **xml:lang** qualifier on the XMP value represented by the literalPropertyElt.

C.2.8 The parseTypeLiteralPropertyElt

```
7.2.17 parseTypeLiteralPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseLiteral ) )
      literal
   end-element()

<ns:Prop rdf:parseType="Literal">  <!-- parseTypeLiteralPropertyElt -->
   ...
</ns:Prop>
```

The parseTypeLiteralPropertyElt is not allowed by XMP. It is a controversial component of RDF that requires the entire XML content of the outer element to be preserved and reconstituted as a textual literal. That is, in essence, the original XML input text of all contained elements, processing instructions, comments, and character data comprise the “literal”. In XMP, this would be a simple property whose value happened to contain XML markup. The XMP approach is to serialize this with escaping, not as text that is actual XML markup.

C.2.9 The parseTypeResourcePropertyElt

```
7.2.18 parseTypeResourcePropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseResource ) )
      propertyEltList
   end-element()

<ns:Struct rdf:parseType="Resource">  <!-- parseTypeResourcePropertyElt -->
   <ns:Field> ... </ns:Field>
   ...
</ns:Struct>
```

©Adobe Systems Incorporated, 2012

```
<ns:Struct>  <!-- resourcePropertyElt -->
   <rdf:Description>  <!-- nodeElement -->
      <ns:Field> ... </ns:Field>
      ...
   </rdf:Description>
</ns:Struct>
```

A parseTypeResourcePropertyElt is a form of shorthand that replaces the inner nodeElement of a resourcePropertyElt with an **rdf:parseType="Resource"** attribute on the outer element. This form is commonly used in XMP as a cleaner way to represent a struct.

The **rdf:ID** attribute is not allowed in XMP.

A parseTypeResourcePropertyElt can have an **xml:lang** attribute. It becomes an **xml:lang** qualifier on the XMP value represented by the parseTypeResourcePropertyElt.

C.2.10 The parseTypeCollectionPropertyElt

```
7.2.19 parseTypeCollectionPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseCollection ) )
      nodeElementList
   end-element()

<ns:List rdf:parseType="Collection">  <!-- parseTypeCollectionPropertyElt -->
   ...
</ns:List>
```

A parseTypeCollectionPropertyElt is not allowed by XMP. It appeared as an addition to RDF after XMP was first delivered, and does not map well to the XMP data model. In RDF usage, a collection models a LISP-like sequential list, with additional RDF-specific semantics.

C.2.11 The parseTypeOtherPropertyElt

```
7.2.20 parseTypeOtherPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, parseOther ) )
      propertyEltList
   end-element()

<ns:Prop rdf:parseType="...">  <!-- parseTypeOtherPropertyElt -->
   ...
</ns:Prop>
```

A parseTypeOtherPropertyElt is not allowed by XMP. It is an element containing an **rdf:parseType** attribute whose value is other than Resource, Literal, or Collection. The RDF/XML Syntax Specification says that the content of a parseTypeOtherPropertyElt is to be treated as a literal in the same manner as a parseTypeLiteralPropertyElt.

C.2.12 The emptyPropertyElt

```
7.2.21 emptyPropertyElt
   start-element ( URI == propertyElementURIs,
                   attributes == set ( idAttr?, ( resourceAttr | nodeIdAttr )?,
                   propertyAttr* ) )
   end-element()

<ns:Prop1/>  <!-- a simple property with an empty value -->
<ns:Prop2 rdf:resource="http://www.adobe.com/"/>  <!-- a URI value -->
```


©Adobe Systems Incorporated, 2012

```
<ns:Prop3 rdf:value="..." ns:Qual="..."/>  <!-- a simple qualified property -->
<ns:Prop4 ns:Field1="..." ns:Field2="..."/>  <!-- a struct with simple fields -->
```

An emptyPropertyElt is an element with no contained content, just a possibly empty set of attributes. An emptyPropertyElt can represent three special cases of simple XMP properties: a simple property with an empty value (**ns:Prop1** above); a simple property whose value is a URI (**ns:Prop2** above); or an alternative RDF form for a simple property with simple qualifiers (**ns:Prop3** above). An emptyPropertyElt can also represent an XMP struct whose fields are all simple and unqualified (**ns:Prop4** above).

An emptyPropertyElt can have an **xml:lang** attribute. It becomes an **xml:lang** qualifier on the XMP data model property represented by the emptyPropertyElt.

The **rdf:ID** and **rdf:nodeID** attributes are not allowed in XMP.

The XMP mapping for an emptyPropertyElt is a bit different from generic RDF, partly for design reasons and partly for historical reasons. The XMP mapping rules are:

1 If there is an **rdf:value** attribute, then this is a simple property. All other attributes are qualifiers.

2 If there is an **rdf:resource** attribute, then this is a simple property with a URI value. All other attributes are qualifiers.

3 If there are no attributes other than **xml:lang**, **rdf:ID**, or **rdf:nodeID**, then this is a simple property with an empty value.

4 Finally, this is a struct, and the attributes other than **xml:lang**, **rdf:ID**, or **rdf:nodeID** are the fields.

Proper operation requires that the XMP mapping rules be applied in the order shown. The concurrent use of **rdf:value** and **rdf:resource** is discouraged.

In the form with an **rdf:resource** attribute, the fact that the value is a URI is not a qualifier in XMP; it is part of the value form.

©Adobe Systems Incorporated, 2012

Bibliography

[1] ISO 639-1, *Codes for the representation of names of languages — Part 1: Alpha-2 code*

[2] ISO 639-2, *Codes for the representation of names of languages — Part 2: Alpha-3 code*

[3] ISO 3166-1, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

[4] IETF RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace, July 2005* : http://www.ietf.org/rfc/rfc4122.txt

©Adobe Systems Incorporated, 2012