XMP Specification

January 2004

Adding Intelligence to Media

ADOBE SYSTEMS INCORPORATED
Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://www.adobe.com

Table of Contents

Table of Contents

Preface

About This Document

The XMP (Extensible Metadata Platform) provides a standard format for the creation, processing, and interchange of metadata, for a wide variety of applications.

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

Audience

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

How This Document Is Organized

This document has the following sections:

- Chapter 1, "Introduction", explains what metadata is, and gives a brief overview of the XMP model.
- Chapter 2, "XMP Data Model", gives a conceptual overview of the data that XMP supports. It describes how metadata is organized into schemas containing a number of properties.
- Chapter 3, "XMP Storage Model", shows the overall structure of XMP data in files.
- Chapter 4, "XMP Schemas", lists common schemas that are used for XMP metadata, as well as the value types used for properties. It also describes how new schemas can be defined to meet needs beyond what is supported by the existing model.
- Chapter 5, "Embedding XMP Metadata in Application Files", describes how XMP metadata is embedded in a variety of specific application files.

Conventions used in this Document

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

Typeface Style	Used for:
Sans serif regular	XMP property names. For example, xmp:CreationDate
`Monospaced Regular`	All XML code

Where to Go for More Information

See these sites for information on the Internet standards and recommendations on which XMP Metadata is based:

Dublin Core Metadata Initiative	http://purl.org/DC/
Extensible Markup Language (XML)	http://www.w3.org/XML/
IETF Standard for Language element values (RFC 1766)	http://www.ietf.org/rfc/rfc1766.txt?number=1766
ISO 639 Standard for Language Codes	http://www.loc.gov/standards/iso639-2/
ISO 3166 Standard for Country Codes	http://www.iso.ch/iso/en/prods-services/iso3166ma/index.html
Naming and Addressing: URIs, URLs, and so on	http://www.w3.org/Addressing/
Resource Description Framework (RDF):	http://www.w3.org/RDF/
Resource Description Framework (RDF) Model and Syntax Specification	http://www.w3.org/TR/REC-rdf-syntax/
Unicode	http://www.unicode.org
XML Namespaces	http://www.w3.org/TR/REC-xml-names/

1 Introduction

What is Metadata?

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

There can be gray areas where the same information could be treated as content or metadata, depending on the workflow. In general, metadata should have value on its own without regard for the 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 documents. Applications can do many useful things with metadata in files, even if they are not able to understand the native file format of the document. Metadata can greatly increase the utility of managed assets in collaborative production workflows. For example, an image file might contain metadata such as its working title, description, thumbnail image, and intellectual property rights data. Accessing the metadata makes it easier to perform such tasks as associating images with file names, locating image captions, or determining 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 file it is associated with.

What is XMP?

In order for multiple applications to be able to work effectively with metadata, there must be a common standard that they understand. XMP—the Extensible Metadata Platform—is designed to provide such a standard.

XMP standardizes the definition, creation, and processing of metadata by providing the following:

- A *data model*: A useful and flexible way of describing metadata in documents: see Chapter 2, “XMP Data Model”.
- A *storage model*: The implementation of the data model: see Chapter 3, “XMP Storage Model”. This includes the serialization of the metadata as a stream of XML; and *XMP Packets*, a means of packaging the data in files. Chapter 5, “Embedding XMP Metadata in Application Files”, describes how XMP Packets are embedded in various file formats.

- *Schemas*: Predefined sets of metadata property definitions that are relevant for a wide range of applications, including all of Adobe's editing and publishing products, as well as for applications from a wide variety of vendors. See Chapter 4, "XMP Schemas". XMP also provides guidelines for the extension and addition of schemas.

The following XMP features are described in separate documents:

- *The Adobe XMP Toolkit* describes Adobe's open source toolkit API for developers.
- *XMP Custom Panels* describes how to create a Custom Panel Description file, which gives developers the ability to define, create, and manage custom metadata properties by customizing the standard **File Info** dialog in Adobe applications that support XMP.

XMP is designed to accommodate a wide variety of workflows and tool environments. It allows localization and supports Unicode.

XMP metadata is encoded as XML-formatted text, using the W3C standard Resource Description Framework (RDF), described in Chapter 3, "XMP Storage Model".

NOTE: The string "XAP" or "xap" appears in some namespaces, keywords, and related names in this document and in stored XMP data. It reflects an early internal code name for XMP; the names have been preserved for compatibility purposes.

What XMP Does Not Cover

Applications can support XMP by providing the ability to preserve and generate XMP metadata, giving users access to the metadata, and supporting extension capabilities.

A number of related areas are outside the scope of XMP itself, and should be under the control of the applications and tools that support XMP metadata, although this document may make some recommendations. These areas include the following:

- The specific metadata set by each application.
- The operation of media management systems.
- The user interface to metadata.
- The definition of schemas beyond those defined by XMP.
- Validity and consistency checking on metadata properties.
- The requirement that users set or edit metadata.

Following the XMP schemas and guidelines presented in this document cannot guarantee the integrity of metadata or metadata flow. That integrity must be accomplished and maintained by a specific set of applications and tools.

2 XMP Data Model

This chapter describes the kinds of data that XMP supports.

- Metadata Properties" describes how metadata items are associated with a document in the form of *properties*.
- "Schemas and Namespaces" on page 12 discusses how properties are named and organized into groups called *schemas*.
- "Property Values" on page 13 describes the data types that can be used for XMP properties.

Metadata Properties

In XMP, metadata consists of a set of properties. Properties are always associated with a particular entity referred to as a *resource*; that is, the properties are "about" the resource. A resource may be:

- A file. This includes simple files such as JPEG images, or more complex files such as entire PDF documents.
- A meaningful portion of a file, as determined by the file structure and the applications that process it. For example, an image imported into a PDF file is a meaningful entity that could have associated metadata. However, a range of pages is not meaningful, because there is no specific PDF structure that corresponds to it. In general, XMP is not designed to be used with very fine-grained subcomponents, such as words or characters.

Any given property has a *name* and a *value*. Conceptually, each property makes a statement about a resource of the form

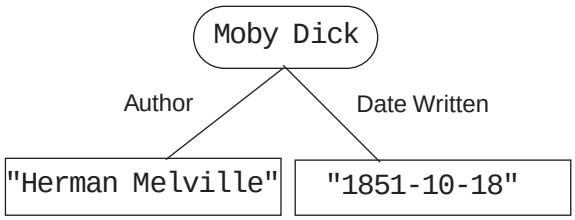
> "The *property_name* of *resource* is *property_value*."

For example:

> The author of *Moby Dick* is "Herman Melville"

This chapter uses a number of diagrams to illustrate the data model. The top or root of the metadata tree is shown as the document or component the metadata applies to, as in the following figure.

NOTE: All property names must be legal XML names.



Schemas and Namespaces

A schema is a set of properties. Typically, schemas may consist of properties that are relevant only for particular types of documents or for certain stages of a workflow. Chapter 4, "XMP Schemas", defines a set of standard metadata schemas and explains how to define new schemas.

Each schema is identified by means of a *namespace* (which follows the usage of XML namespaces). The use of namespaces avoids conflict between properties in different schemas that have the same name but different meanings. For example, two independently designed schemas might have a Creator property: in one, it might mean the person who created a resource; in another, the application used to create the resource. Name conflicts are avoided by qualifying property names with a schema-specific namespace prefix (see below).

Each schema consists of

- A *schema name*, which is a URI that serves to uniquely identify the schema. It is simply a unique string. (Although it often looks like a URL, there might or might not be an actual Web page at the URI. In the case of Adobe namespaces, currently there is no corresponding Web page.) The URI must obey XML namespace rules, and to be RDF-compliant it should end in "/" or "#".

  NOTE: The schema URI is a unique string, whose components have no significance. For example, `foo:/schema/1.0/` and `foo:/schema/2.0/` are completely different schemas with no necessary relationship between them.

- A *schema namespace prefix,* which is a short abbreviation for the full schema name. The schema namespace prefixes used here are not formal. Following the rules of XML namespaces, the schema namespace prefix is simply shorthand for the schema URI and is local to the scope of the `xmlns` attribute that declares it.

For example, in the following code, the namespace prefix for the XMP Basic Schema is defined to be `xmp`:

```
xmlns:xmp="http://ns.adobe.com/xap/1.0/"
```

Following XML qualified name conventions, properties in a schema are written as

prefix:*name*

where *prefix* is a schema namespace prefix and *name* is a valid simple XML name; for example, xmp:CreateDate.

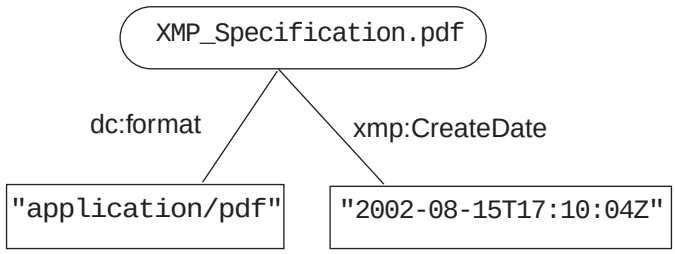
Property Values

The data types that can represent the values of XMP properties are in three basic categories, described here: *simple types*, *structures*, and *arrays*. Since XMP metadata is stored as XML, values of all types are written as Unicode strings.

This section shows conceptual examples of XMP data types. "Serializing XMP" on page 19 shows how these examples are represented in XML. Definitions of all predefined properties and value types can be found in Chapter 4, "XMP Schemas".

Simple Types

A simple type has a single literal value. Simple types include familiar ones such as strings, booleans, integers and real numbers, as well as others such as Choice.

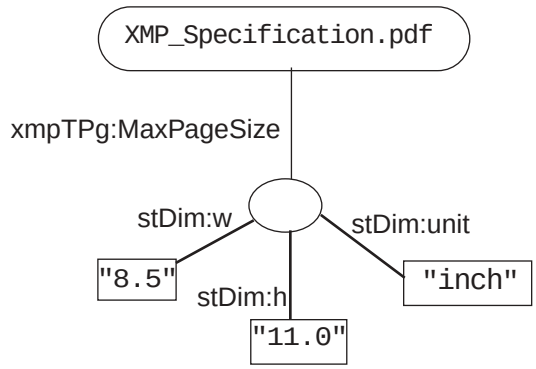


In this figure, the document `XMP_Specification.pdf` is shown with 2 simple properties:

- The value of the property dc:format is the MIMEType value `"application/pdf"`.
- The value of the property xmp:CreateDate is the Date value `"2002-08-15T17:10:04Z"`.

Structures

A structured property consists of one or more named fields.

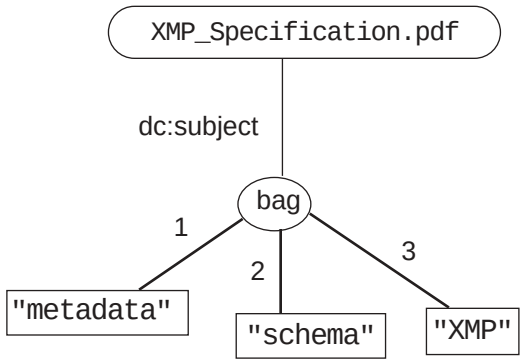


This example shows a single structured property whose type is Dimensions. The structure has its own XML namespace prefix (`stDim`), although this is not required of structure fields in general. There are three fields: stDim:w (width), stDim:h (height) and stDim:unit (units), whose values are `"8.5"`, `"11.0"` and `"inch"`.

A field in a structure can itself be a structure or an array.

Arrays

An array consists of a set of values. You can think of an array as a structure whose field names are ordinal numbers, as shown in this figure.



The individual elements of an array are strongly recommended to be of the same type. (In the example, the elements are of type Text.) In addition to simple types, array elements may be structures or arrays.

XMP supports three types of arrays: *unordered*, *ordered*, and *alternative*, described in the following sections.

Unordered Arrays

An *unordered* array is a list of values whose order does not have significance. For example, the order of keywords associated with a document does not generally matter, so the dc:subject property is defined as an unordered array.

In the schema definitions, an unordered array is referred to as a *bag*. For example, dc:subject is defined as “bag Text”.

Ordered Arrays

An *ordered* array is a list whose order is significant. For example, the order of authors of a document might matter (such as in academic journals), so the dc:creator property is defined as an ordered array.

In the schema definitions, an ordered array is referred to as a *seq*. For example, dc:creator is defined as “seq ProperName”.

Alternative Arrays

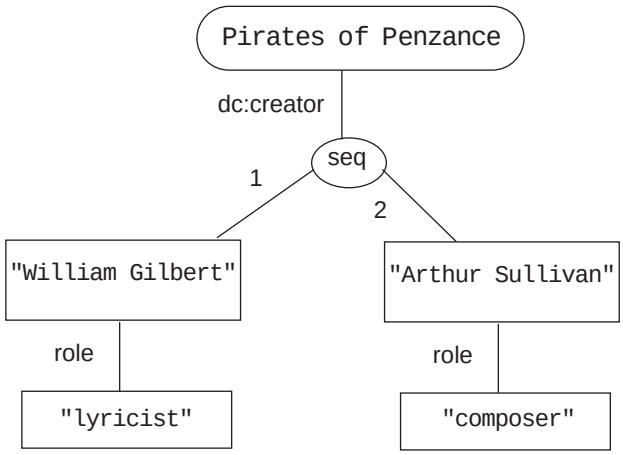
An *alternative* array is a set of one or more values, one of which should be chosen. In the schema definitions, an alternative array is referred to as an *alt*. For example, xmp:Thumbnails is defined as “alt Thumbnail”. There are no specific rules for selection of alternatives: in some situations, an application may make a choice; in others, a user may make a choice. The first item in the array is considered by RDF to be the default value.

A common example is an array that contains the same logical text (such as a title or copyright) in multiple languages. This is known as a *language alternative*; it is described further in “Language Alternatives” on page 16.

Property Qualifiers

Any individual property value may have other properties attached to it; these attached properties are called *property qualifiers*. They are in effect “properties of properties”; they can provide additional information about the property value. For example, a digital resource representing a musical production might have one or more authors, specified using the dc:creator property, which is an array (see the figure below). Each array value might have a property qualifier called role, which could take a value of `"composer"` or `"lyricist"` or possibly other values.

NOTE: At this time, only simple properties may have qualifiers, and the qualifiers themselves must be simple values (not structures or arrays). This is because of limitations in early versions of the Adobe XMP Toolkit.



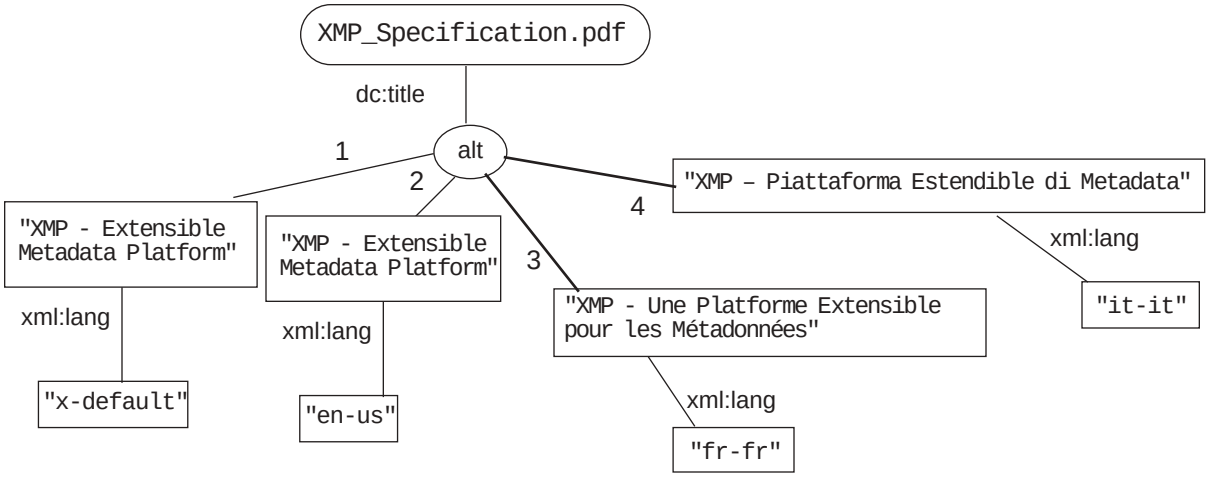
Property qualifiers allow values to be extended without breaking existing usage. For example, the role qualifier in the diagram does not interfere with readers who simply want the dc:creator names. An alternative would be to change dc:creator values to structures with name and role fields, but that would confuse old software that expected to find a simple value.

The most common specific use of property qualifiers is for language alternative arrays (see next section).

Language Alternatives

Language alternatives allow the text value of a property to be chosen based on a desired language. Each item in a language alternative array is a simple text value, which must have a *language qualifier* associated with it. The language qualifier is a property qualifier, as described in the previous section. Its property name is xml:lang, and its value is a string that conforms to RFC 1766 notation (see http://www.ietf.org/rfc/rfc1766.txt.).

XMP requires the `"x-default"` language code to be supplied as the default. It should be the first item in the array, so that RDF-aware applications that are unaware of XMP will also use it. The figure below shows an example:



3 XMP Storage Model

This chapter describes how XMP metadata that conforms to the data model discussed in the previous chapter is stored (*serialized*) in files.

- XMP properties are serialized as XML, specifically RDF (see "Serializing XMP", below).
- The serialized data is wrapped in packets for embedding in files. "XMP Packets" on page 28 describes the structure and capabilities of these packets.
- Packets are stored in files in a natural manner for each file format; specific file formats are discussed in Chapter 5, "Embedding XMP Metadata in Application Files".
- "External Storage of Metadata" on page 33 describes how to store XMP data in a separate file from the document with which it is associated.

Serializing XMP

In order to represent the metadata properties associated with a document (that is, to serialize it in a file), XMP makes use of the Resource Description Framework (RDF) standard, which is based on XML. By adopting the RDF standard, XMP benefits from the documentation, tools, and shared implementation experience that come with an open W3C standard. RDF is described in the W3C document *Resource Description Framework (RDF) Model and Syntax Specification* at http://www.w3.org/TR/REC-rdf-syntax/.

The sections below describe the high-level structure of XMP data in an XMP Packet:

- The outermost element is optionally an x:xmpmeta element
- It contains a single rdf:RDF element
- which in turn contains one or more rdf:Description elements
- each of which contains one or more XMP Properties.

The examples in this document are shown in RDF syntax. RDF has multiple ways to serialize the same data model: a "typical" or verbose way, and several forms of shorthand. The examples shown here use the typical way plus a few forms of shorthand used by the Adobe XMP Toolkit; they are designed to assist human readers of stored XMP. Any valid shorthand may be used.

XMP supports a subset of RDF; see "RDF Issues" on page 27 for further information.

NOTE: The default encoding for text is UTF-8 with support for all Unicode characters. Multibyte Unicode encodings can also be used.

x:xmpmeta element

It is recommended that an `x:xmpmeta` element be the outermost XML element in the serialized XMP data, to simplify locating XMP metadata in general XML streams. The format is:

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

The `xmpmeta` element can have any number of attributes, in any order. All unrecognized attributes are ignored, and there are no required attributes. The only defined attribute at present is `x:xmptk`, written by the Adobe XMP Toolkit; its value is the version of the toolkit.

NOTE: Earlier versions of XMP suggested use of the `x:xapmeta` element. Applications filtering input should recognize both.

rdf:RDF element

Immediately within the `x:xmpmeta` element should be a single `rdf:RDF` element.

```
<x:xmpmeta xmlns:x='adobe:ns:meta/'>
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      ...
   </rdf:RDF>
</x:xmpmeta>
```

rdf:Description elements

The `rdf:RDF` element can contain one or more `rdf:Description` elements. The following example shows a single `rdf:Description` element:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <rdf:Description rdf:about=""
       xmlns:dc="http://purl.org/dc/elements/1.1/">
         ... Dublin Core properties go here
   </rdf:Description>
</rdf:RDF>
```

By convention, all properties from a given schema, and only that schema, are listed within a single `rdf:Description` element. (This is not a requirement, just a means to improve readability.) In this example, properties from the Dublin Core schema are specified within the `rdf:Description` element. The `xmlns:dc` attribute defines the namespace prefix (`dc:`) to be used. Properties from other schemas would be specified inside additional `rdf:Description` elements.

NOTE: The `rdf:Description` element is also used when specifying structured properties (see "Structures" on page 23).

The rdf:about attribute

The `rdf:about` attribute on the `rdf:Description` element is a required attribute that identifies the resource whose metadata this XMP describes. The value of this attribute must follow URI syntax and may be either:

- an empty string (as in the example above), which means that the XMP is physically local to the resource being described. Applications must rely on knowledge of the file format to correctly associate the XMP with the resource.
- a unique *instance ID* that is generated every time a file is saved. The next section gives guidelines for creating instance IDs.

Instance IDs

When referring to computer files, there can often be ambiguity. The contents of a file can change over time. Depending on the situation, it might be desirable to refer to either:

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

Some characteristics of a file (such as the application that created it) would normally be expected to be persistent over its life. Other characteristics (such as word count) would be expected to change as the content of the file 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 file are inherently ambiguous as to whether they refer to the current content of a file or to the file in general. XMP itself provides no mechanisms for distinguishing these. Schemas are encouraged, but not required, to define properties in a way that makes this clear.

This document uses the term *resource* to refer to the "thing the metadata is about." Depending on the context, resources may refer to either the specific or persistent aspects described above. In order to refer unambiguously to a specific state of the file, we use the term *instance.*

NOTE: This terminology should be distinguished from HTTP terminology, where *resource* is most often used in the sense of "container", while *entity* or *entity-part* is always used to mean "the current content of all or part of a resource at some point in time."

The instance IDs mentioned above are specific IDs, since they are created every time a file is saved. They do not provide any connection between different versions of a document. However, in many cases, an instance ID can also be used to locate the resource, because if the instance referred to is the content of a resource at some point in time, the instance identifier also denotes that resource at that time. Therefore, using an instance ID in the `rdf:about` attribute allows identification of both the resource and the particular content it had at the time the metadata was generated or stored.

NOTE: In some situations, more persistent identification might be desired. It can be provided by using the xmpMM:DocumentID property in the XMP Media Management schema.

An instance ID should be a GUID/UUID-style ID, which is a large integer that is guaranteed to be globally unique (in practical terms, the probability of a collision is so remote as to be effectively impossible). Typically 128- or 144-bit integers are used, encoded as 22 or 24 base-64 characters.

XMP does not require any specific scheme for generating the unique number. There are various common schemes available for that purpose, such as:

- Using physical information such as a local Ethernet address and a high resolution clock.

  NOTE: When creating a unique ID, applications must consider tradeoffs between privacy and the desire to create an audit trail. Adobe applications favor privacy and do not include Ethernet addresses.

- Using a variety of locally unique and random data, then computing an MD5 hash value. This avoids privacy concerns about the use of Ethernet addresses. It also allows for regeneration of the ID in some cases; for example if the MD5 hash is computed using the image contents for a resource that is a digital photograph.

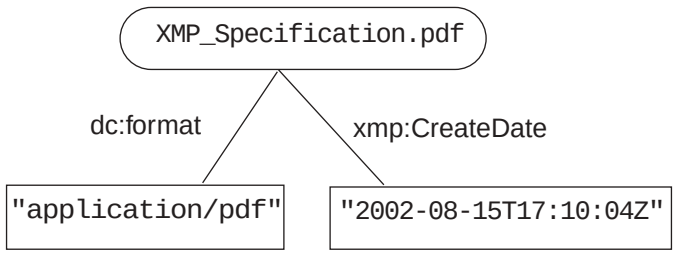
Because the `rdf:about` attribute is the only identification of the resource from the RDF point of view, it is useful to format its value in a standard manner. This lets other RDF-aware software know what kind of URI is used (in particular, that it is not a URL). There is no formal W3C recommendation for URIs that are based on an abstract UUID. The following two proposals may be relevant:

- http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-01.txt
- http://www.ietf.org/internet-drafts/draft-king-vnd-urlscheme-03.txt

XMP Properties

This section shows how the properties diagrammed in "Property Values" on page 13 would be serialized in XMP. The data diagrams are repeated for convenience.

Simple Types



In XMP, these properties would specified as follows:

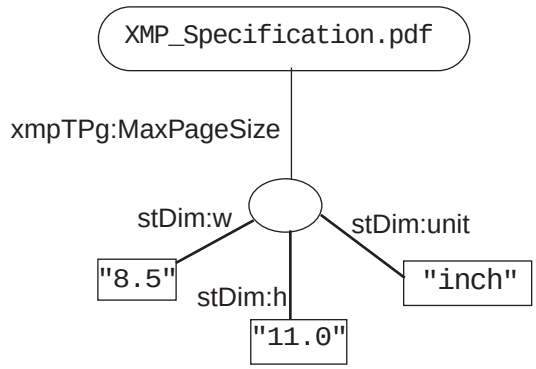
```
<rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:format>application/pdf</dc:format>
</rdf:Description>

<rdf:Description rdf:about="" xmlns:xmp="ns.adobe.com/xap/1.0/">
  <xmp:CreateDate>2002-08-15T17:10:04Z</xmp:CreateDate>
</rdf:Description>
```

Alternatively, there is a common form of RDF shorthand that writes simple properties as attributes of the `rdf:Description` element. The second `rdf:Description` element above would be specified as follows:

```
<rdf:Description rdf:about="" xmlns:xmp="ns.adobe.com/xap/1.0/"
  xmp:CreateDate="2002-08-15T17:10:04Z"/>
```

NOTE: All property names must be legal XML names.

Structures



This example shows a property that is a structure containing three fields. It would be serialized in XML as:

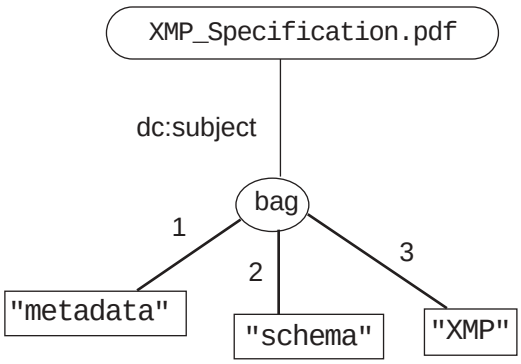
```
<rdf:Description rdf:about=""
        xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/">
  <xmpTPg:MaxPageSize>
     <rdf:Description
           xmlns:stDim="http:ns.adobe.com/xap/1.0/sType/Dimensions#">
        <stDim:w>4</stDim:w>
        <stDim:h>3</stDim:h>
        <stDim:unit>inches</stDim:unit>
     </rdf:Description>
  </xmpTPg:MaxPageSize>
</rdf:Description>
```

The element hierarchy consists of:

- The `rdf:Description` element, described above, which specifies the namespace for the property.
- The `xmpTPg:MaxPageSize` element, which is a property of type Dimensions
- An inner `rdf:Description` element, which is necessary to declare the presence of a structure. It also defines the namespace that is used by the structure fields. Inner `rdf:Description` elements do not have an `rdf:about` attribute.

  NOTE: Structure fields are not required to use a schema namespace; they must conform to the rules of XML qualified names.
- The fields of the Dimensions structure.

Arrays



This example (from "Arrays" on page 14) is serialized as follows:

```
<rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:subject>
     <rdf:Bag>
        <rdf:li>metadata</rdf:li>
        <rdf:li>schema</rdf:li>
        <rdf:li>XMP</rdf:li>
     </rdf:Bag>
  </dc:subject>
</rdf:Description>
```

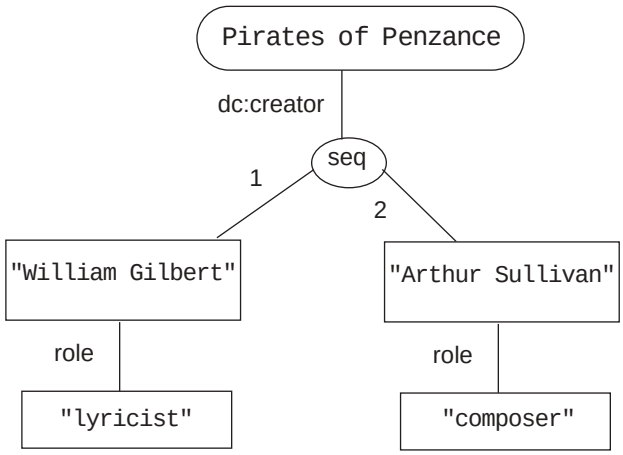
The dc:subject property is an unordered array, represented by the type `rdf:Bag`. It contains one `rdf:li` element for each item in the array. Ordered and alternative arrays are similar, except that they use the types `rdf:Seq` and `rdf:Alt`, respectively. An example of an alternative array is shown below in "Language Alternatives".

Property Qualifers

Property qualifiers can be serialized in one of two ways:

- There is a general representation, as shown in the following figure.
- There is a special representation for xml:lang qualifiers (see "Language Alternatives" on page 26)

Here is a general example, repeated from "Property Qualifiers" on page 15.



The figure above shows an array with two elements, each of which has a property qualifier called role. It would be serialized as follows:

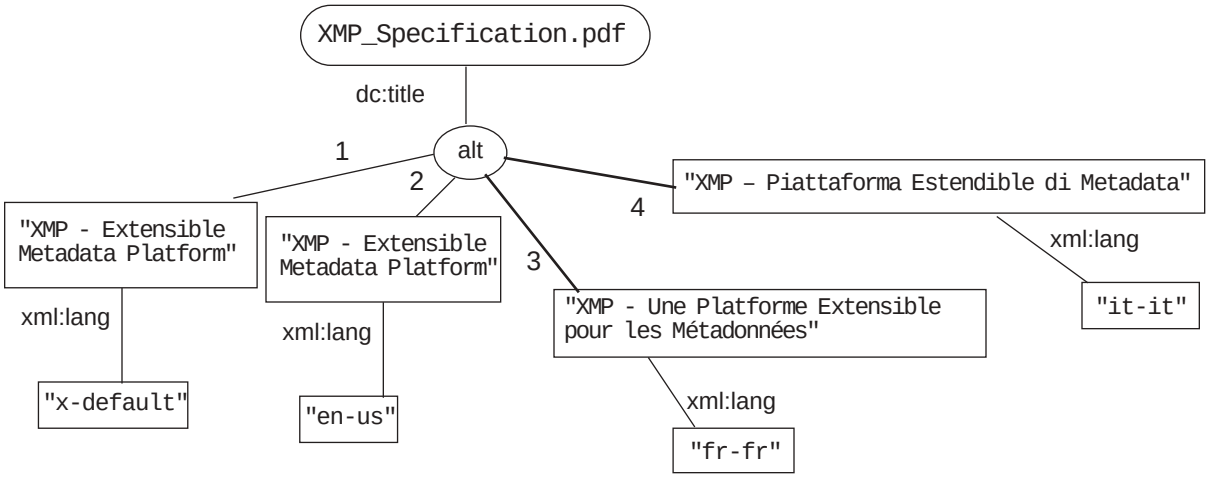
```
<rdf:Description rdf:about=""
        xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator>
     <rdf:Seq>
        <rdf:li>
           <rdf:Description>
              <rdf:value>William Gilbert</rdf:value>
              <role>lyricist</role>
           </rdf:Description>
        </rdf:li>
           <rdf:Description >
              <rdf:value>Arthur Sullivan</rdf:value>
              <role>composer</role>
           </rdf:Description>
        </rdf:li>
     </rdf:Seq>
  </dc:creator>
</rdf:Description>
```

The presence of property qualifiers is indicated by a special use of the `rdf:Description` element. Each `rdf:li` array item in the example contains an `rdf:Description` element, which itself contains the following:

- a special element called `rdf:value` that represents the value of the property
- zero or more other elements that represent qualifiers of the value. In this case, there is one property qualifier called role.

Language Alternatives

Text properties may have an xml:lang property qualifier that specifies the language of the text. A common use is with language alternative arrays.



Language alternatives are a form of `rdf:Alt` array, referred to as the Lang Alt type. In this example, each array item is a simple text value; the value has a property qualifier, specified as the property xml:lang, giving the language of that value.

The XMP for this array looks like this:

```
<xmp: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-fr">
         XMP - Une Platforme Extensible pour les Métadonnées</rdf:li>
    <rdf:li xml:lang="it-it">
         XMP - Piattaforma Estendible di Metadata</rdf:li>
  </rdf:Alt>
</xmp:Title>
```

The xml:lang qualifier is written as an attribute of the XML element whose character data is the value (in this case, the `rdf:li` elements). Note also the special language value `"x-default"`, which specifies the default title to be used.

RDF Issues

Unsupported Features

XMP uses a subset of RDF. Valid XMP is limited to the RDF described in the previous sections, along with all equivalent shorthand. All XMP is valid RDF, but a number of features of the RDF specification are not valid XMP, in particular:

- The `rdf:RDF` element is required by XMP (it is optional in RDF).
- Top-level elements must be `rdf:Description` elements.
- The `rdf:ID` attribute is ignored.
- The `rdf:bagID` attribute is ignored.
- The `rdf:aboutEach` or `rdf:aboutEachPrefix` attributes are not supported (entire `rdf:Description` ignored).
- The `rdf:parseType='Literal'` attribute is not supported.

Validation

If DTD or XML Schema validation is required, be aware that RDF provides many equivalent ways to express the same model. Also, the open nature of XMP means that it is in general not possible to predict or desirable to constrain the allowable set of XML elements and attributes. There appears to be no way to write a DTD that allows arbitrary elements and attributes. Even use of `ANY` requires declared child elements (see validity constraint #4 in section 3 of the XML specification).

The recommended approach to placing XMP in XML using DTD validation is to wrap the XMP Packet in a `CDATA` section. This requires escaping any use of "`]]>`" in the packet.

XMP Packets

The XMP Packet format specifies how XMP metadata is embedded in files. It consists of a "wrapper" around the serialized XMP data described in the previous section. XMP Packets:

- may be embedded in a wide variety of binary and text formats, including native XML files.
- are delimited by easy-to-scan markers. Such markers are XML syntax-compatible to allow transmission to an XML parser without additional filtering.
- deal with arbitrary positioning within a byte stream (so as not to rely on machine word boundaries, and so on).
- enable in-place editing, including expansion, of metadata.
- allow multiple packets to be embedded in a single data file.

Chapter 5, "Embedding XMP Metadata in Application Files", gives information on how XMP Packets are embedded in specific file formats. Applications may also scan files for XMP Packets without knowledge of the file format itself, although this should be regarded as a last resort (see "Scanning Files for XMP Packets" on page 31).

Figure 3.1 shows a schematic of an XMP Packet. It contains a header, XML data, padding, and a trailer.

FIGURE 3.1 XMP Packet Schematic

Here is an outline of an XMP Packet, showing the text of the header and trailer:

```
<?xpacket begin='■' id='W5M0MpCehiHzreSzNTczkc9d'?>
 ... the serialized XMP as described above: ...
       <x:xmpmeta xmlns:x='adobe:ns:meta/'>
          <rdf:RDF xmlns:rdf= ...>
             ...
          </rdf:RDF>
       </x:xmpmeta>
 ... XML whitespace as padding ...
<?xpacket end='w'?>
```

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

An XMP Packet must conform to the Well-Formedness requirements of the XML specification, except for the lack of an XML declaration at its start. Different packets in a file can be in different character encodings, and packets must not nest.

The following sections describe the parts of the packet illustrated in Figure 3.1.

Header

The header is an XML processing instruction of the form:

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

The processing instruction contains information about the packet in the form of XML attributes. There are two required attributes: `begin` and `id`, in that order. Other attributes can follow in any order; unrecognized attributes should be ignored. Attributes must be separated by exactly one space (U+0020) character.

Attribute: begin

This required attribute indicates the beginning of a new packet. Its value is the Unicode zero-width non-breaking space character U+FEFF, in the appropriate encoding (UTF-8, UTF-16, or UTF-32). It serves as a byte-order marker, where the character is written in the natural order of the application (consistent with the byte order of the XML data encoding).

For backward compatibility with earlier versions of the XMP Packet specification, the value of this attribute can be the empty string, indicating UTF-8 encoding.

"Scanning Files for XMP Packets" on page 31 describes how an XMP Packet processor should read a single byte at a time until it has successfully determined the byte order and encoding.

Attribute: id

The required `id` attribute must follow `begin`. For all packets defined by this version of the syntax, the value of `id` is the following string of 7-bit ASCII characters:

```
W5M0MpCehiHzreSzNTczkc9d
```

The string must be encoded in the character encoding of the overall packet. For example, if the overall encoding is big-endian UTF-16, the `id` value should be converted from 7-bit ASCII to UTF-16 by inserting nulls.

Attribute: bytes

NOTE: This attribute is deprecated.

The optional `bytes` attribute specifies the total length of the packet in bytes, which can allow faster scanning of XMP Packets. If the length extends beyond the end of the trailer processing instruction, the additional bytes must be properly encoded Unicode whitespace and are considered padding.

Use the `bytes` attribute only for an XMP Packet embedded in a binary file. Do not use it for XMP Packets embedded in text files, since the length of text can innocently change when moved among computers. For example, moving a text file from a Macintosh or UNIX system to Windows typically causes all single byte line endings (*CR* or *LF*) to become two bytes (*CRLF*). This would invalidate the length given by the `bytes` attribute.

Attribute: encoding

NOTE: This attribute is deprecated.

The optional `encoding` attribute is identical to the `encoding` attribute in the XML declaration (see productions [23] and [80] in the XML specification). It specifies the character encoding of the entire packet. It should be consistent with the Unicode encoding implied by the `begin` attribute.

XMP Data

The bytes of the XMP data are placed here. Their encoding must match the encoding implied by the header's `begin` attribute. The structure of the data is described in "Serializing XMP" above.

The XMP data should not contain an `XML` declaration. The XML specification requires that the `XML` declaration be "the first thing in the entity"; this is not the case for an embedded XMP Packet.

NOTE: An XMP Packet should not contain other XML that does not conform to XMP.

Padding

It is recommended that applications allocate 2 KB to 4 KB of padding to the packet. This allows the XMP to be edited in place, and expanded if necessary, without overwriting existing application data. The padding must be XML-compatible whitespace; the recommended practice is to use the space character (U+0020) in the appropriate encoding, with a newline about every 100 characters.

Trailer

This required processing instruction indicates the end of the XMP Packet.

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

Attribute: end

The `end` attribute is required, and must be the first attribute.

NOTE: Other unrecognized attributes can follow, but should be ignored. Attributes must be separated by exactly one space (U+0020) character.

The value of `end` indicates whether applications that do not understand the containing file format are allowed to update the XMP Packet:

- `r` means the packet is "read-only" and must not be updated in place.

  NOTE: `r` is not meant to restrict the behavior of applications that understand the file format and are capable of properly rewriting the file.

- `w` means the packet can be updated in place, if there is enough space. The overall length of the packet must not be changed; padding should be adjusted accordingly. The original encoding and byte order must be preserved, to avoid breaking text files containing XMP or violating other constraints of the original application.

Scanning Files for XMP Packets

This section explains how files can be scanned for XMP Packets, and why this should be done with caution.

Caveats

Knowledge of individual file formats provides the best way for an application to get access to XMP Packets. See Chapter 5, "Embedding XMP Metadata in Application Files" for detailed information on how XMP data is stored in specific file formats.

Lacking this information, applications can find XMP Packets by scanning the file. However, this should be considered a last resort, especially if it is necessary to modify the data. Without knowledge of the file format, simply locating packets may not be sufficient. The following are some possible drawbacks:

- It may not be possible to determine which resource the XMP is associated with. If a JPEG image with XMP is placed in a page layout file of an application that is unaware of XMP, that file has one XMP Packet that refers to just the image, not the entire layout.
- When there is more than one XMP Packet in a file, it may be impossible to determine which is the "main" XMP, and what the overall resource containment hierarchy is in a compound document.
- Some packets could be obsolete. For example, PDF files allow incremental saves. Therefore, when changes are made to the document, there might be multiple packets, only one of which reflects the current state of the file.

Scanning Hints

A file should be scanned byte-by-byte until a valid header is found. First, the scanner should look for a byte pattern that represents the text

```
<?xpacket begin=
```

which will be one of the following byte patterns:

- 8-bit encoding (UTF-8, ASCII 7-bit, ISOLatin-1):
  `0x3C 0x3F 0x78 0x70 0x61 0x63 0x6B`
  `0x65 0x74 0x20 0x62 0x65 0x67 0x69 0x6E 0x3D`
- 16-bit encoding (UCS-2, UTF-16): (either big- or little-endian order)
  `0x3C 0x00 0x3F 0x00 0x78 0x00 0x70 0x00 0x61 0x00`
  `0x63 0x00 0x6B 0x00 0x65 0x00 0x74 0x00 0x20 0x00 0x62 0x00`
  `0x65 0x00 0x67 0x00 0x69 0x00 0x6E 0x00 0x3D [0x00]`
- 32-bit encoding (UCS-4): the pattern is similar to the UCS-2 pattern above, except with three `0x00` bytes for every one in the UCS-2 version.

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

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

After finding a matching byte pattern, the scanner must consume a quote character, which can be either the single quote (apostrophe) (U+0027) or double quote (U+0022) character.

NOTE: Individual attribute values in the processing instruction can have either single or double quotes. The following header is well-formed:

```
<?xpacket begin="■" id='W5M0MpCehiHzreSzNTczkc9d'?>
```

The scanner is now ready to read the value of the `begin` attribute, followed by the closing quote character:

UTF-8:	`0xEF 0xBB 0xBF`
UTF-16, big-endian:	`0xFE 0xFF`
UTF-16, little-endian:	`0xFF 0xFE`
UTF-32, big-endian:	`0x00 0x00 0xFE 0xFF`
UTF-32, little-endian:	`0xFF 0xFE 0x00 0x00`

NOTE: If the attribute has no value, the encoding is UTF-8.

The scanner now has enough information to process the rest of the header in the appropriate character encoding.

External Storage of Metadata

It is suggested, though not required, that XMP metadata be embedded in the file that the metadata describes (as XMP Packets). There are cases where this is not appropriate or possible, such as database storage models, extremes of file size, or due to format and access issues. Small content intended to be frequently transmitted over the Internet might not tolerate the overhead of embedded metadata. Archival systems for video and audio might not have any means to represent the metadata. In addition, some high-end digital cameras have a proprietary, non-extensible file format for "raw" image data and typically store EXIF metadata as a separate file.

If metadata is stored separately from content, there is a risk that the metadata can be lost. The question arises of how to associate the metadata with the file containing the content. Applications should:

- Write the external file as a complete well-formed XML document, including the leading `XML` declaration.
- The file extension should be `.xmp`. For Mac OS, optionally set the file's type to `'TEXT'`.
- If a MIME type is needed, use `application/rdf+xml`.
- Write external metadata as though it were embedded and then had the XMP Packets extracted and catenated by a postprocessor.
- If possible, place the instance ID used in the `rdf:about` attribute within the file the XMP describes, so that format-aware applications can make sure they have the right metadata.

For applications that need to find external XMP files, look in the same directory for a file with the same name as the main document but with an `.xmp` extension. (This is called a *sidecar* XMP file.)

4 XMP Schemas

This chapter contains the following information:

- Definitions for the standard XMP Schemas
  - “Dublin Core Schema” on page 37
  - “XMP Basic Schema” on page 38
  - “XMP Rights Management Schema” on page 40
  - “XMP Media Management Schema” on page 41
  - “XMP Basic Job Ticket Schema” on page 44
  - “XMP Paged-Text Schema” on page 45
- Definitions for a set of specialized schemas:
  - “Adobe PDF Schema” on page 46
  - “Photoshop Schema” on page 47
  - “EXIF Schemas” on page 48
- Definitions and explanations of property values used by the schemas (“Property Value Types” on page 62)
- Guidelines for extending XMP (“Extensibility of Schemas” on page 70).

XMP metadata may include properties from one or more of the schemas. For example, a typical subset used by many Adobe applications might include the following:

- Dublin Core schema: dc:title, dc:creator, dc:description, dc:subject, dc:format, dc:rights
- XMP basic schema:xmp:CreateDate, xmp:CreatorTool, xmp:ModifyDate , xmp:MetadataDate
- XMP rights management schema: xmpRights:WebStatement, xmpRights:Marked
- XMP media management schema: xmpMM:DocumentID

XMP Schema Definitions

The schema definitions in this chapter show the namespace string that identifies the schema, and a preferred schema namespace prefix, followed by a table that lists all properties defined for the schema. Each table has the following columns:

- **Property**: the name of the property, including the preferred namespace prefix.
- **Value Type:** The value type of the property, with links to where each value type is described in "Property Value Types" on page 62. Array types are preceded by the container type: alt, bag, or seq. (see "Arrays" on page 14 for details).
- **Category:** Schema properties are *internal* or *external*:
  - Internal metadata must be maintained by an application. It can include system-level information (such as modification date) or information that an editing application has access to (such as the number of words in a document). An example is xmp:ModifyDate. Users should not be allowed to change the values of such properties. When a file is saved, an application should provide valid values for all internal properties. If an application does not set the value of an internal property, it should discard any value that may have existed previously.
  - External metadata must be set by a user, and is independent of the contents of the document. External modifications should be displayed by the editing application but are not acted upon. Unless changed by the user, external properties are preserved on output. An example is dc:creator.
- **Description:** The description of the property.

NOTE: Some XMP properties have been deprecated since earlier versions of the specification. They are defined here for compatibility purposes, but should not be used in the future.

NOTE: Previous versions of this specification referred to *aliased* properties. Specific XMP implementations may treat a property in one schema as equivalent to a property in another schema. However, to foster interchange, applications must always write the standard, "base" form of the property. In this version of the specification, only the base properties are listed.

The schemas define a set of properties. In any given XMP, a property may be:

- Absent; that is, it has no value. Properties are absent until given a value for the first time.
- Present; that is, it has a defined value.

  NOTE: A present property may have the empty string as its value; this is different from an absent property. However, writers are encouraged not to set properties with a value of the empty string.

For any given XMP, there is no requirement that all properties from a given schema must be present. For structured properties, there is no requirement that all fields be present (unless otherwise specified by a schema).

Dublin Core Schema

The Dublin Core schema provides a set of commonly used properties.

- The schema namespace URI is `http://purl.org/dc/elements/1.1/`
- The preferred schema namespace prefix is dc

Property	Value Type	Category	Description
dc:contributor	bag ProperName	External	Contributors to the resource (other than the authors).
dc:coverage	Text	External	The extent or scope of the resource.
dc:creator	seq ProperName	External	The authors of the resource (listed in order of precedence, if significant).
dc:date	seq Date	External	Date(s) that something interesting happened to the resource.
dc:description	Lang Alt	External	A textual description of the content of the resource. Multiple values may be present for different languages.
dc:format	MIMEType	Internal	The file format used when saving the resource. Tools and applications should set this property to the save format of the data. It may include appropriate qualifiers.
dc:identifier	Text	External	Unique identifier of the resource.
dc:language	bag Locale	Internal	An unordered array specifying the languages used in the resource.
dc:publisher	bag ProperName	External	Publishers.
dc:relation	bag Text		Relationships to other documents.
dc:rights	Lang Alt	External	Informal rights statement, selected by language.
dc:source	Text	External	Unique identifier of the work from which this resource was derived.
dc:subject	bag Text	External	An unordered array of descriptive phrases or keywords that specify the topic of the content of the resource.
dc:title	Lang Alt	External	The title of the document, or the name given to the resource. Typically, it will be a name by which the resource is formally known.
dc:type	bag open Choice	External	A document type; for example, novel, poem, or working paper.

XMP Basic Schema

The XMP Basic Schema contains properties that provide basic descriptive information.

- The schema namespace URI is `http://ns.adobe.com/xap/1.0/`
- The preferred schema namespace prefix is `xmp`

Property	Value Type	Category	Description
xmp:Advisory	bag XPath	External	An unordered array specifying properties that were edited outside the authoring application. Each item should contain a single namespace and XPath separated by one ASCII space (U+0020).
xmp:BaseURL	URL	Internal	The base URL for relative URLs in the document content. If this document contains Internet links, and those links are relative, they are relative to this base URL. This property provides a standard way for embedded relative URLs to be interpreted by tools. Web authoring tools should set the value based on their notion of where URLs will be interpreted.
xmp:CreateDate	Date	External	The date and time the resource was originally created.
xmp:CreatorTool	AgentName	Internal	The name of the first known tool used to create the resource. If history is present in the metadata, this value should be equivalent to that of xmpMM:History's softwareAgent property.
xmp:Identifier	bag Text	External	An unordered array of text strings that unambiguously identify the resource within a given context. An array item may be qualified with xmpidq:Scheme to denote the formal identification system to which that identifier conforms. NOTE: The dc:identifier property is not used because it lacks a defined scheme qualifier and has been defined in the XMP Specification as a simple (single-valued) property.
xmp:MetadataDate	Date	Internal	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.

Property	Value Type	Category	Description
xmp:ModifyDate	Date	Internal	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 set before the file is saved.
xmp:Nickname	Text	External	A short informal name for the resource.
xmp:Thumbnails	alt Thumbnail	Internal	An alternative array of thumbnail images for a file, which can differ in characteristics such as size or image encoding.

An item in the xmp:Identifier array may be qualified with xmpidq:Scheme to denote the formal identification system to which that identifier conforms.

- The qualifier namespace URI is `http://ns.adobe.com/xmp/Identifier/qual/1.0/`
- The preferred qualifier namespace prefix is `xmpidq`

Qualifier	Value Type	Category	Description
xmpidq:Scheme	Text	External	The name of the formal identification system used in the value of the associated xmp:Identifier item.

XMP Rights Management Schema

This schema includes properties related to rights management. These properties specify information regarding the legal restrictions associated with a resource.

NOTE: XMP is not a rights-enforcement mechanism.

- The schema namespace URI is `http://ns.adobe.com/xap/1.0/rights/`
- The preferred schema namespace prefix is `xmpRights`

Property	Value Type	Category	Description
xmpRights:Certificate	URL	External	Online rights management certificate.
xmpRights:Marked	Boolean	External	Indicates that this is a rights-managed resource.
xmpRights:Owner	bag ProperName	External	An unordered array specifying the legal owner(s) of a resource.
xmpRights:UsageTerms	Lang Alt	External	Text instructions on how a resource can be legally used.
xmpRights:WebStatement	URL	External	The location of a web page describing the owner and/or rights statement for this resource.

XMP Media Management Schema

The XMP Media Management Schema is primarily for use by digital asset management (DAM) systems.

The following properties are “owned” by the DAM system and should be set by applications under their direction; they should not be used by unmanaged files: xmpMM: ManagedFrom, xmpMM:Manager, xmpMM:ManageTo, xmpMM:ManageUI, xmpMM: ManagerVariant.

The following properties are owned by the DAM system for managed files, but can also be used by applications for unmanaged files: xmpMM:DerivedFrom, xmpMM:DocumentID, xmpMM: RenditionClass, xmpMM: RenditionParams, xmpMM:VersionID, xmpMM:Versions.

The xmpMM:History property is always owned by the application.

- The schema namespace URI is `http://ns.adobe.com/xap/1.0/mm/`
- The preferred schema namespace prefix is `xmpMM`

Property	Value Type	Category	Description
xmpMM:DerivedFrom	ResourceRef	Internal	A reference to the original document from which this one is derived. It is a minimal reference; missing components can be assumed to be unchanged. For example, a new version might only need to specify the instance ID and version number of the previous version, or a rendition might only need to specify the instance ID and rendition class of the original.
xmpMM:DocumentID	URI	Internal	The common identifier for all versions and renditions of a document. It should be based on a UUID; see discussion under “The rdf:about attribute” on page 21.
xmpMM:History	seq ResourceEvent	Internal	An ordered array of high-level user actions that resulted in this resource. It is intended to give human readers a general indication of the steps taken to make the changes from the previous version to this one. The list should be at an abstract level; it is not intended to be an exhaustive keystroke or other detailed history.

Property	Value Type	Category	Description
xmpMM: ManagedFrom	ResourceRef	Internal	A reference to the document as it was prior to becoming managed. It is set when a managed document is introduced to an asset management system that does not currently own it. It may or may not include references to different management systems.
xmpMM:Manager	AgentName	Internal	The name of the asset management system that manages this resource. Along with xmpMM: ManagerVariant, it tells applications which asset management system to contact concerning this document.
xmpMM:ManageTo	URI	Internal	A URI identifying the managed resource to the asset management system; the presence of this property is the formal indication that this resource is managed. The form and content of this URI is private to the asset management system.
xmpMM:ManageUI	URI	Internal	A URI that can be used to access information about the managed resource through a web browser. It might require a custom browser plug-in.
xmpMM: ManagerVariant	Text	Internal	Specifies a particular variant of the asset management system. The format of this property is private to the specific asset management system.
xmpMM: RenditionClass	RenditionClass	Internal	The rendition class name for this resource. This property should be absent or set to `default` for a document version that is not a derived rendition.
xmpMM: RenditionParams	Text	Internal	Can be used to provide additional rendition parameters that are too complex or verbose to encode in xmpMM: RenditionClass.
xmpMM:VersionID	Text	Internal	The document version identifier for this resource. Each version of a document gets a new identifier, usually simply by incrementing integers 1, 2, 3 . . . and so on. Media management systems can have other conventions or support branching which requires a more complex scheme.

Property	Value Type	Category	Description
xmpMM:Versions	seq Version	Internal	The version history associated with this resource. Entry `[1]` is the oldest known version for this document, entry `[last()]` is the most recent version. Typically, a media management system would fill in the version information in the metadata on check-in. It is not guaranteed that a complete history of versions from the first to this one will be present in the xmpMM:Versions property. Interior version information can be compressed or eliminated and the version history can be truncated at some point.
xmpMM:LastURL *(deprecated)*	URL	Internal	Deprecated for privacy protection.
xmpMM:RenditionOf *(deprecated)*	ResourceRef	Internal	Deprecated in favor of xmpMM:DerivedFrom. A reference to the document of which this is a rendition.
xmpMM:SaveID *(deprecated)*	Integer	Internal	Deprecated. Previously used only to support the xmpMM:LastURL property.

XMP Basic Job Ticket Schema

The following schema describes very simple workflow or job information.

- The schema namespace URI is `http://ns.adobe.com/xap/1.0/bj/`
- The preferred schema namespace prefix is `xmpBJ`

Property	Value Type	Category	Description
xmpBJ:JobRef	bag Job	External	References an external job management file for a job process in which the document is being used. Use of job names is under user control. Typical use would be to identify all documents that are part of a particular job or contract. There are multiple values because there can be more than one job using a particular document at any time, and it can also be useful to keep historical information about what jobs a document was part of previously.

XMP Paged-Text Schema

The Paged-Text schema is used for text appearing on a page in a document.

- The schema namespace URI is `http://ns.adobe.com/xap/1.0/t/pg/`
- The preferred schema namespace prefix is `xmpTPg`

Property	Value Type	Category	Description
xmpTPg:MaxPageSize	Dimensions	Internal	The size of the largest page in the document (including any in contained documents).
xmpTPg:NPages	Integer	Internal	The number of pages in the document (including any in contained documents).

Adobe PDF Schema

This schema specifies properties used with Adobe PDF files.

- The schema namespace URI is `http://ns.adobe.com/pdf/1.3/`
- The preferred schema namespace prefix is `pdf`

Property	Value Type	Category	Description
pdf:Keywords	Text	External	Keywords.
pdf:PDFVersion	Text	Internal	The PDF file version (for example: 1.0, 1.3, and so on).
pdf:Producer	AgentName	Internal	The name of the tool that created the PDF document.

Photoshop Schema

This schema specifies properties used by Adobe Photoshop.

- The schema namespace URI is `http://ns.adobe.com/photoshop/1.0/`
- The preferred schema namespace prefix is `photoshop`

Property	Value Type	Category	Description
photoshop:AuthorsPosition	Text	External	By-line title.
photoshop:CaptionWriter	ProperName	External	Writer/editor.
photoshop:Category	Text	External	Category. Limited to 3 7-bit ASCII characters.
photoshop:City	Text	External	City.
photoshop:Country	Text	External	Country/primary location.
photoshop:Credit	Text	External	Credit.
photoshop:DateCreated	Date	External	The date the intellectual content of the document was created (rather than the creation date of the physical representation), following IIM conventions. For example, a photo taken during the American Civil War would have a creation date during that epoch (1861-1865) rather than the date the photo was digitized for archiving.
photoshop:Headline	Text	External	Headline.
photoshop:Instructions	Text	External	Special instructions.
photoshop:Source	Text	External	Source.
photoshop:State	Text	External	Province/state.
photoshop: SupplementalCategories	Text	External	Supplemental category.
photoshop: TransmissionReference	Text	External	Original transmission reference.
photoshop:Urgency	Integer	External	Urgency. Valid range is 1-8.

EXIF Schemas

EXIF is a metadata standard for image files, used widely by digital cameras. The EXIF 2.2 specification can be found at http://www.exif.org/specifications.html.

There are two XMP schemas that correspond to parts of the EXIF 2.2 specification, described in the following sections:

- "EXIF Schema for TIFF Properties" on page 48
- "EXIF Schema for EXIF-specific Properties" on page 50

The property descriptions assume that the reader has some familiarity with EXIF metadata. The XMP property names are identical to the names used within the EXIF specification; more complete descriptions of the properties can be found in the specification.

The following sections provide further information:

- "Data Representation and Conversion" on page 59 describes guidelines for converting between the XMP and EXIF formats, with examples.
- "EXIF Schema Value Types" on page 67 describes EXIF-specific value types.

NOTE: XMP properties of type Date include fractional seconds; therefore EXIF properties for fractional seconds (`SubSecTime`, `SubSecTimeOriginal`, `SubSecTimeDigitized`) are included in the "main XMP property".

EXIF Schema for TIFF Properties

The following table lists the properties for TIFF-derived data. Only those TIFF properties that are mentioned in the EXIF 2.2 specification are included here.

- The schema name is `http://ns.adobe.com/tiff/1.0/`
- The preferred schema namespace prefix is `tiff`

Property	Value Type	Category	Description
tiff:ImageWidth	Integer	Internal	TIFF tag 256, 0x100. Image width in pixels.
tiff:ImageLength	Integer	Internal	TIFF tag 257, 0x101. Image height in pixels.
tiff:BitsPerSample	seq Integer	Internal	TIFF tag 258, 0x102. Number of bits per component in each channel.
tiff:Compression	Closed Choice of Integer	Internal	TIFF tag 259, 0x103. Compression scheme:1 = uncompressed; 6 = JPEG.
tiff: PhotometricInterpretation	Closed Choice of Integer	Internal	TIFF tag 262, 0x106. Pixel Composition: 2 = RGB; 6 = YCbCr.

Property	Value Type	Category	Description
tiff:Orientation	Closed Choice of Integer	Internal	TIFF tag 274, 0x112. Orientation: 1 = 0th row at top, 0th column at left 2 = 0th row at top, 0th column at right 3 = 0th row at bottom, 0th column at right 4 = 0th row at bottom, 0th column at left 5 = 0th row at left, 0th column at top 6 = 0th row at right, 0th column at top 7 = 0th row at right, 0th column at bottom 8 = 0th row at left, 0th column at bottom
tiff:SamplesPerPixel	Integer	Internal	TIFF tag 277, 0x115. Number of components per pixel.
tiff:PlanarConfiguration	Closed Choice of Integer	Internal	TIFF tag 284, 0x11C. Data layout:1 = chunky; 2 = planar.
tiff:YCbCrSubSampling	Closed Choice of seq Integer	Internal	TIFF tag 530, 0x212. Sampling ratio of chrominance components: [2, 1] = YCbCr4:2:2 [2, 2] = YCbCr4:2:0
tiff:YCbCrPositioning	Closed Choice of Integer	Internal	TIFF tag 531, 0x213. Position of chrominance vs. luminance components: 1 = centered; 2 = co-sited.
tiff:XResolution	Rational	Internal	TIFF tag 282, 0x11A. Horizontal resolution in pixels per unit.
tiff:YResolution	Rational	Internal	TIFF tag 283, 0x11B. Vertical resolution in pixels per unit.
tiff:ResolutionUnit	Closed Choice of Integer	Internal	TIFF tag 296, 0x128. Unit used for XResolution and YResolution. Value is one of: 2 = inches; 3 = centimeters.
tiff:TransferFunction	seq Integer	Internal	TIFF tag 301, 0x12D. Transfer function for image described in tabular style with 3 * 256 entries.
tiff:WhitePoint	seq Rational	Internal	TIFF tag 318, 0x13E. Chromaticity of white point.
tiff:PrimaryChromaticities	seq Rational	Internal	TIFF tag 319, 0x13F. Chromaticity of the three primary colors.
tiff:YCbCrCoefficients	seq Rational	Internal	TIFF tag 529, 0x211. Matrix coefficients for RGB to YCbCr transformation.
tiff:ReferenceBlackWhite	seq Rational	Internal	TIFF tag 532, 0x214. Reference black and white point values.

Property	Value Type	Category	Description
tiff:DateTime	Date	Internal	TIFF tag 306, 0x132 (primary) and EXIF tag 37520, 0x9290 (subseconds). Date and time of image creation (no time zone in EXIF), stored in ISO 8601 format, not the original EXIF format. This property includes the value for the EXIF SubsecTime attribute. **NOTE:** This property is stored in XMP as xmp:ModifyDate.
tiff:ImageDescription	Lang Alt	External	TIFF tag 270, 0x10E. Description of the image. **NOTE:** This property is stored in XMP as dc:description.
tiff:Make	ProperName	Internal	TIFF tag 271, 0x10F. Manufacturer of recording equipment.
tiff:Model	ProperName	Internal	TIFF tag 272, 0x110. Model name or number of equipment.
tiff:Software	AgentName	Internal	TIFF tag 305, 0x131. Software or firmware used to generate image. **NOTE:** This property is stored in XMP as xmp:CreatorTool.
tiff:Artist	ProperName	External	TIFF tag 315, 0x13B. Camera owner, photographer or image creator. **NOTE:** Each entry in EXIF string should become an individual entry in the dc:creator property. When round-tripping, each entry in the dc:creator property should be separated by a semicolon.
tiff:Copyright	Lang Alt	External	TIFF tag 33432, 0x8298. Copyright information. **NOTE:** This property is stored in XMP as dc:rights.

EXIF Schema for EXIF-specific Properties

The following table lists the properties defined solely by EXIF.

NOTE: A number of EXIF 2.2 properties are not included in XMP. These are generally properties that relate directly to the image stream, or that are of little use without access to the image stream. A general XMP principle is that XMP metadata should have value in and of itself, separate from the primary file content. The omitted properties include: StripOffsets, RowsPerStrip, StripByteCounts, JPEGInterchangeFormat, and JPEGInterchangeFormatLength

NOTE: Properties beginning with “GPS” are GPS properties that are also used by DIG-35 and are part of the JPEG-2000 standard.

- The schema name is `http://ns.adobe.com/exif/1.0/`
- The preferred schema namespace prefix is `exif`

Property	Value Type	Category	Description
exif:ExifVersion	Closed Choice of Text	Internal	EXIF tag 36864, 0x9000. Version number; must be `"0210"`.
exif:FlashpixVersion	Closed Choice of Text	Internal	EXIF tag 40960, 0xA000. Version of FlashPix: must be `"0100"`.
exif:ColorSpace	Closed Choice of Integer	Internal	EXIF tag 40961, 0xA001. Color space information: 1 = sRGB –32786 = uncalibrated
exif:ComponentsConfiguration	Closed Choice of seq Integer	Internal	EXIF tag 37121, 0x9101. Configuration of components in data: 4 5 6 0 (if RGB compressed data), 1 2 3 0 (other cases). 0 = does not exist 1 = Y 2 = Cb 3 = Cr 4 = R 5 = G 6 = B
exif:CompressedBitsPerPixel	Rational	Internal	EXIF tag 37122, 0x9102. Compression mode used for a compressed image is indicated in unit bits per pixel.
exif:PixelXDimension	Integer	Internal	EXIF tag 40962, 0xA002. Valid image width, in pixels.
exif:PixelYDimension	Integer	Internal	EXIF tag 40963, 0xA003. Valid image height, in pixels.
exif:MakerNote	Text	Internal	EXIF tag 37500, 0x927C. Undefined information in EXIF, is of `UNDEFINED` type in EXIF spec.
exif:UserComment	Lang Alt	External	EXIF tag 37510, 0x9286. Comments from user.
exif:RelatedSoundFile	Text	Internal	EXIF tag 40964, 0xA004. An “8.3” file name for the related sound file.

Property	Value Type	Category	Description
exif:DateTimeOriginal	Date	Internal	EXIF tags 36867, 0x9003 (primary) and 37521, 0x9291 (subseconds). Date and time when original image was generated, in ISO 8601 format. Includes the EXIF `SubsecTimeOriginal` data.
exif:DateTimeDigitized	Date	Internal	EXIF tag 36868, 0x9004 (primary) and 37522, 0x9292 (subseconds). Date and time when image was stored as digital data, can be the same as DateTimeOriginal if originally stored in digital form. Stored in ISO 8601 format. Includes the EXIF `SubsecTimeDigitized` data.
exif:ExposureTime	Rational	Internal	EXIF tag 33434, 0x829A. Exposure time in seconds.
exif:FNumber	Rational	Internal	EXIF tag 33437, 0x829D. F number.
exif:ExposureProgram	Closed Choice of Integer	Internal	EXIF tag 34850, 0x8822. Class of program used for exposure: 0 = not defined 1 = Manual 2 = Normal program 3 = Aperture priority 4 = Shutter priority 5 = Creative program 6 = Action program 7 = Portrait mode 8 = Landscape mode
exif:SpectralSensitivity	Text	Internal	EXIF tag 34852, 0x8824. Spectral sensitivity of each channel.
exif:ISOSpeedRatings	seq Integer	Internal	EXIF tag 34855, 0x8827. ISO Speed and ISO Latitude of the input device as specified in ISO 12232.
exif:OECF	OECF/SFR	Internal	EXIF tag 34856, 0x8828. Opto-Electoric Conversion Function as specified in ISO 14524.
exif:ShutterSpeedValue	Rational	Internal	EXIF tag 37377, 0x9201. Shutter speed, unit is APEX. See Annex C of the EXIF specification.
exif:ApertureValue	Rational	Internal	EXIF tag 37378, 0x9202. Lens aperture, unit is APEX.
exif:BrightnessValue	Rational	Internal	EXIF tag 37379, 0x9203. Brightness, unit is APEX.

Property	Value Type	Category	Description
exif:ExposureBiasValue	Rational	Internal	EXIF tag 37380, 0x9204. Exposure bias, unit is APEX.
exif:MaxApertureValue	Rational	Internal	EXIF tag 37381, 0x9205. Smallest F number of lens, in APEX.
exif:SubjectDistance	Rational	Internal	EXIF tag 37382, 0x9206. Distance to subject, in meters.
exif:MeteringMode	Closed Choice of Integer	Internal	EXIF tag 37383, 0x9207. Metering mode: 0 = unknown 1 = Average 2 = CenterWeightedAverage 3 = Spot 4 = MultiSpot 5 = Pattern 6 = Partial 255 = other
exif:LightSource	Closed Choice of Integer	Internal	EXIF tag 37384, 0x9208. EXIF tag , 0x. Light source: 0 = unknown 1 = Daylight 2 = Fluorescent 3 = Tungsten 17 = Standard light A 18 = Standard light B 19 = Standard light C 20 = D55 21 = D65 22 = D75 255 = other
exif:Flash	Flash	Internal	EXIF tag 37385, 0x9209. Strobe light (flash) source data.
exif:FocalLength	Rational	Internal	EXIF tag 37386, 0x920A. Focal length of the lens, in millimeters.
exif:SubjectArea	seq Integer	Internal	EXIF tag 37396, 0x9214. The location and area of the main subject in the overall scene.
exif:FlashEnergy	Rational	Internal	EXIF tag 41483, 0xA20B. Strobe energy during image capture.
exif: SpatialFrequencyResponse	OECF/SFR	Internal	EXIF tag 41484, 0xA20C. Input device spatial frequency table and SFR values as specified in ISO 12233.

Property	Value Type	Category	Description
exif:FocalPlaneXResolution	Rational	Internal	EXIF tag 41486, 0xA20E. Horizontal focal resolution, measured pixels per unit.
exif:FocalPlaneYResolution	Rational	Internal	EXIF tag 41487, 0xA20F. Vertical focal resolution, measured in pixels per unit.
exif:FocalPlaneResolutionUnit	Closed Choice of Integer	Internal	EXIF tag 41488, 0xA210. Unit used for FocalPlaneXResolution and FocalPlaneYResolution. 2 = inches 3 = centimeters
exif:SubjectLocation	seq Integer	Internal	EXIF tag 41492, 0xA214. Location of the main subject of the scene. The first value is the horizontal pixel and the second value is the vertical pixel at which the main subject appears.
exif:ExposureIndex	Rational	Internal	EXIF tag 41493, 0xA215. Exposure index of input device.
exif:SensingMethod	Closed Choice of Integer	Internal	EXIF tag 41495, 0xA217. Image sensor type on input device: 1 = Not defined 2 = One-chip color area sensor 3 = Two-chip color area sensor 4 = Three-chip color area sensor 5 = Color sequential area sensor 7 = Trilinear sensor 8 = Color sequential linear sensor
exif:FileSource	Closed Choice of Integer	Internal	EXIF tag 41728, 0xA300. Indicates image source: 3 (DSC) is the only choice.
exif:SceneType	Closed Choice of Integer	Internal	EXIF tag 41729, 0xA301. Indicates the type of scene: 1 (directly photographed image) is the only choice.
exif:CFAPattern	CFAPattern	Internal	EXIF tag 41730, 0xA302. Color filter array geometric pattern of the image sense.
exif:CustomRendered	Closed Choice of Integer	Internal	EXIF tag 41985, 0xA401. Indicates the use of special processing on image data: 0 = Normal process 1 = Custom process

Property	Value Type	Category	Description
exif:ExposureMode	Closed Choice of Integer	Internal	EXIF tag 41986, 0xA402. Indicates the exposure mode set when the image was shot: 0 = Auto exposure 1 = Manual exposure 2 = Auto bracket
exif:WhiteBalance	Closed Choice of Integer	Internal	EXIF tag 41987, 0xA403. Indicates the white balance mode set when the image was shot: 0 = Auto white balance 1 = Manual white balance
exif:DigitalZoomRatio	Rational	Internal	EXIF tag 41988, 0xA404. Indicates the digital zoom ratio when the image was shot.
exif:FocalLengthIn35mmFilm	Integer	Internal	EXIF tag 41989, 0xA405. Indicates the equivalent focal length assuming a 35mm film camera, in mm. A value of 0 means the focal length is unknown. Note that this tag differs from the FocalLength tag.
exif:SceneCaptureType	Closed Choice of Integer	Internal	EXIF tag 41990, 0xA406. Indicates the type of scene that was shot: 0 = Standard 1 = Landscape 2 = Portrait 3 = Night scene
exif:GainControl	Closed Choice of Integer	Internal	EXIF tag 41991, 0xA407. Indicates the degree of overall image gain adjustment: 0 = None 1 = Low gain up 2 = High gain up 3 = Low gain down 4 = High gain down
exif:Contrast	Closed Choice of Integer	Internal	EXIF tag 41992, 0xA408. Indicates the direction of contrast processing applied by the camera: 0 = Normal 1 = Soft 2 = Hard

Property	Value Type	Category	Description
exif:Saturation	Closed Choice of Integer	Internal	EXIF tag 41993, 0xA409. Indicates the direction of saturation processing applied by the camera: 0 = Normal 1 = Low saturation 2 = High saturation
exif:Sharpness	Closed Choice of Integer	Internal	EXIF tag 41994, 0xA40A. Indicates the direction of sharpness processing applied by the camera: 0 = Normal 1 = Soft 2 = Hard
exif:DeviceSettingDescription	DeviceSettings	Internal	EXIF tag 41995, 0xA40B. Indicates information on the picture-taking conditions of a particular camera model.
exif:SubjectDistanceRange	Closed Choice of Integer	Internal	EXIF tag 41996, 0xA40C. Indicates the distance to the subject: 0 = Unknown 1 = Macro 2 = Close view 3 = Distant view
exif:ImageUniqueID	Text	Internal	EXIF tag 42016, 0xA420. An identifier assigned uniquely to each image. It is recorded as a 32 character ASCII string, equivalent to hexadecimal notation and 128-bit fixed length.
exif:GPSVersionID	Text	Internal	GPS tag 0, 0x00. A decimal encoding of each of the four EXIF bytes with period separators. The current value is “2.0.0.0”.
exif:GPSLatitude	GPSCoordinate	Internal	GPS tag 2, 0x02 (position) and 1, 0x01 (North/South). Indicates latitude.
exif:GPSLongitude	GPSCoordinate	Internal	GPS tag 4, 0x04 (position) and 3, 0x03 (East/West). Indicates longitude.
exif:GPSAltitudeRef	Closed Choice of Integer	Internal	GPS tag 5, 0x5. Indicates whether the altitude is above or below sea level: 0 = Above sea level 1 = Below sea level
exif:GPSAltitude	Rational	Internal	GPS tag 6, 0x06. Indicates altitude in meters.

Property	Value Type	Category	Description
exif:GPSTimeStamp	Date	Internal	GPS tag 29 (date), 0x1D, and, and GPS tag 7 (time), 0x07. Time stamp of GPS data, in Coordinated Universal Time. **NOTE:** The `GPSDateStamp` tag is new in EXIF 2.2. The GPS timestamp in EXIF 2.1 does not include a date. The formatted XMP value in this case should still follow ISO 8601, namely "`HH:MM:SS.sss<tz>`", where `SS.sss` includes as many fractional second digits as necessary and `<tz>` is a time zone designator. If the denominator of the original fractional sections is not a power of 10, the recommendation is to use 6 fractional digits for microsecond resolution.
exif:GPSSatellites	Text	Internal	GPS tag 8, 0x08. Satellite information, format is unspecified.
exif:GPSStatus	Closed Choice of Text	Internal	GPS tag 9, 0x09. Status of GPS receiver at image creation time: A = measurement in progress V = measurement is interoperability
exif:GPSMeasureMode	Closed Choice of Integer	Internal	GPS tag 10, 0x0A. GPS measurement mode, Text type: 2 = two-dimensional measurement 3 = three-dimensional measurement
exif:GPSDOP	Rational	Internal	GPS tag 11, 0x0B. Degree of precision for GPS data.
exif:GPSSpeedRef	Closed Choice of Text	Internal	GPS tag 12, 0x0C. Units used to speed measurement: K = kilometers per hour M = miles per hour N = knots
exif:GPSSpeed	Rational	Internal	GPS tag 13, 0x0D. Speed of GPS receiver movement.
exif:GPSTrackRef	Closed Choice of Text	Internal	GPS tag 14, 0x0E. Reference for movement direction: T = true direction M = magnetic direction

Property	Value Type	Category	Description
exif:GPSTrack	Rational	Internal	GPS tag 15, 0x0F. Direction of GPS movement, values range from 0 to 359.99.
exif:GPSImgDirectionRef	Closed Choice of Text	Internal	GPS tag 16, 0x10. Reference for movement direction: T = true direction M = magnetic direction
exif:GPSImgDirection	Rational	Internal	GPS tag 17, 0x11. Direction of image when captured, values range from 0 to 359.99.
exif:GPSMapDatum	Text	Internal	GPS tag 18, 0x12. Geodetic survey data.
exif:GPSDestLatitude	GPSCoordinate	Internal	GPS tag 20, 0x14 (position) and 19, 0x13 (North/South). Indicates destination latitude.
exif:GPSDestLongitude	GPSCoordinate	Internal	GPS tag 22, 0x16 (position) and 21, 0x15 (East/West). Indicates destination longitude.
exif:GPSDestBearingRef	Closed Choice of Text	Internal	GPS tag 23, 0x17. Reference for movement direction: T = true direction M = magnetic direction
exif:GPSDestBearing	Rational	Internal	GPS tag 24, 0x18. Destination bearing, values from 0 to 359.99.
exif:GPSDestDistanceRef	Closed Choice of Text	Internal	GPS tag 25, 0x19. Units used for speed measurement: K = kilometers M = miles N = knots
exif:GPSDestDistance	Rational	Internal	GPS tag 26, 0x1A. Distance to destination.
exif:GPSProcessingMethod	Text	Internal	GPS tag 27, 0x1B. A character string recording the name of the method used for location finding.
exif:GPSAreaInformation	Text	Internal	GPS tag 28, 0x1C. A character string recording the name of the GPS area.
exif:GPSDifferential	Closed choice of Integer	Internal	GPS tag 30, 0x1E. Indicates whether differential correction is applied to the GPS receiver: 0 = Without correction 1 = Correction applied

Data Representation and Conversion

This section describes the mapping from the native EXIF 2.2 metadata format to the XMP format. It explains how to do the conversion without losing significant data, and describes the resulting XMP representation.

> **NOTE:** If a particular tag is omitted from an EXIF file, the corresponding XMP property must also be omitted. An XMP property must not be created based on the default value of a missing EXIF tag.

The EXIF to XMP type mappings are designed to be lossless in most cases. The main issues are for EXIF text values. When converting from XMP, integers that are specified optionally as `short` or `long` in EXIF should be represented as `short` if the value is in the range −32768 to +32767, otherwise they should be `long`.

EXIF Text

EXIF text values are a sequence of ASCII characters with a null terminator; XMP text values are Unicode characters in UTF-8 with no null terminator. When converting EXIF to XMP, the null terminator is dropped; the remaining ASCII codes are legitimate UTF-8 values. When converting from XMP to EXIF, non-ASCII characters are escaped (using URL escaping as specified in http://www.w3.org/Addressing/URL/4_Recommentations.html); ASCII characters in the range of 0 through 127 are not escaped (for example, spaces); and a null terminator is added.

XMP text values can be localized. For properties of type Lang Alt, an array of localized text values can be supplied. When converting from EXIF to XMP, the value supplied by the EXIF metadata should be written to the default entry (`[@xml:lang='x-default']`). When converting from XMP to EXIF, the default entry should be used to supply the EXIF metadata.

EXIF Dates

All date/time values are stored in XMP using ISO 8601 format. This is a combined date and time, with fractional seconds, and a time zone designation. The binary EXIF values generally separate the fractional seconds. EXIF 2.1 lacks time zone information; this has been partially added in EXIF 2.2. When converting to XMP, the fractional seconds should be included. If no time zone is contained in the EXIF, convert to XMP assuming a local time.

Example

The following is an example of EXIF 2.2 metadata and the corresponding XMP metadata as it might be converted from the EXIF data.

The EXIF data:

```
IFD 0 [1]
Make = "Canon"
Model = "Canon PowerShot S300"
Orientation = "1"
XResolution = "180/1" (180.00)
YResolution = "180/1" (180.00)
ResolutionUnit = "2"
DateTime = "2001:07:25 20:18:27"
YCbCrPositioning = "1"
ExposureTime = "1/60" (0.0167)
FNumber = "27/10" (2.70)
ExifVersion = "30 32 31 30"
DateTimeOriginal = "2001:07:25 20:18:27"
DateTimeDigitized = "2001:07:25 20:18:27"
ComponentsConfiguration = "1 2 3 0"
CompressedBitsPerPixel = "3/1" (3.00)
ShutterSpeedValue = "189/32" (5.91)
ApertureValue = "93/32" (2.91)
ExposureBiasValue = "0/3" (0.00)
MaxApertureValue = "187820/65536" (2.8659)
SubjectDistance = "913/1000" (0.9130)
MeteringMode = "5"
Flash = "0x01"
FocalLength = "173/32" (5.41)
```

The XMP Metadata:

NOTE: This example uses the RDF shorthand notation of representing simple properties as XML attributes instead of XML elements.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <rdf:Description about='' xmlns:tiff='http://ns.adobe.com/tiff/1.0'
     tiff:Make='Canon'
     tiff:Model='Canon PowerShot S300'
     tiff:Orientation='1'
     tiff:XResolution='180/1'
     tiff:YResolution='180/1'
     tiff:ResolutionUnit='2'
     tiff:DateTime='2001-07-25T20:18:27-07:00'
     tiff:YCbCrPositioning='1'>
  </rdf:Description>

  <rdf:Description about='' xmlns:exif='http://ns.adobe.com/exif/1.0'
     exif:ExposureTime='1/60'
     exif:FNumber='27/10'
     exif:ExifVersion='0210'
     exif:DateTimeOriginal='2001-07-25T20:18:27-07:00'
     exif:DateTimeDigitized='2001-07-25T20:18:27-07:00'
     exif:CompressedBitsPerPixel='3/1'
     exif:ShutterSpeedValue='189/32'
     exif:ApertureValue='93/32'
     exif:ExposureBiasValue='0/3'
     exif:MaxApertureValue='187820/65536'
     exif:SubjectDistance='913/1000'
     exif:MeteringMode='5'
     exif:Flash='1'
     exif:FocalLength='173/32'>
     <exif:ComponentsConfiguration>
        <rdf:Seq>
           <rdf:li>1</rdf:li>
           <rdf:li>2</rdf:li>
           <rdf:li>3</rdf:li>
           <rdf:li>0</rdf:li>
        </rdf:Seq>
     </exif:ComponentsConfiguration>
  </rdf:Description>
</rdf:RDF>
```

Property Value Types

The following tables list the value types used in the XMP schemas.

Basic Value Types

Boolean

Allowed values are `True` or `False` (the strings should be spelled exactly as shown).

Choice

A value chosen from a *vocabulary* of values, and represented by a string. Vocabularies provide a means of specifying a limited but extensible set of values for a property. The metadata schema can indicate whether the set of legal values is fixed or can be extended.

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

- An open choice has one or more lists of preferred values, but other values can be freely used.
- A closed choice only allows values from the defined lists.

If a property value is to have a very definite meaning and all users of that property must know the exact meaning, use a closed choice vocabulary. If there are well-defined sets of values whose meanings are known, but additional values might be used without causing problems, use an open choice.

Date

A date which is represented using ISO 8601 formatting, as described in http://www.w3.org/TR/NOTE-datetime.

Dimensions

A structure containing dimensions for a drawn object.

- The field namespace URI is `http:ns.adobe.com/xap/1.0/sType/Dimensions#`
- The preferred field namespace prefix is `stDim`

Field Name	Value Type	Description
w	Real	Width
h	Real	Height
unit	open Choice	Units. For example: `inch`, `mm`, `pixel`, `pica`, `point`

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.

Lang Alt

A *language alternative* (see "Language Alternatives" on page 16), which is an array of type "alt Text" that has a language property qualifier.

Locale

A closed choice that identifies a language, with values from RFC1766. See http://www.ietf.org/rfc/rfc1766.txt.

Real

A numeric value of arbitrary precision. Consists of a decimal numeric string with an optional single decimal point and an optional leading "+" or "–" sign.

It can optionally have the qualifier vQual:binRep, of type Text, which provides an alternate binary representation for the number when an exact value is needed. The text is interpreted as:

std size,endian,hexadecimal_value

– *std* is the standard name (`"IEEE754"`)
– *size* is `S` for 32-bit and `D` for 64-bit
– *endian* is `L` for little-endian, `B` for big-endian.

For example: `"IEEE754D,L,3A4901F387D31108"`

MIMEType

An open Choice that identifies the file format. MIME types are defined in RFC 2046, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types" at http://www.ietf.org/rfc/rfc2046.txt.

ProperName

A name of a person or organization, represented as a Unicode text string.

Text

A Unicode string.

Thumbnail

A thumbnail image for a file.

- The field namespace URI is `http://ns.adobe.com/xap/1.0/g/img/`
- The preferred field namespace prefix is `xapGImg`

Field Name	Value Type	Description
height	Integer	Height in pixels
width	Integer	Width in pixels
format	Closed Choice	The image encoding. Defined value: `JPEG`.
image	Text	The thumbnail image (pixel data only) converted to base 64 notation (according to section 6.8 of RFC 2045).

URI

An Internet Uniform Resource Identifier:a compact string of characters for identifying an abstract or physical resource. See http://www.w3.org/Addressing/.

URL

An Internet Uniform Resource Locator. See http://www.w3.org/Addressing/. An informal term (no longer used in technical specifications) associated with popular URI schemes: `http`, `ftp`, `mailto`, and so on.

XPath

XML Path Language (XPath), for addressing parts of an XML document; see http://www.w3.org/TR/xpath.

Media Management Value Types

AgentName

The name of a program. The suggested convention is "*vendor app version*"—for example "Adobe Acrobat Distiller 5.0".

RenditionClass

The type of rendition, from a controlled vocabulary of standard names (an open Choice). A series of colon-separated tokens and parameters, the first of which names the basic concept of the rendition. Additional tokens are optional and provide specific characteristics of the rendition. Defined values are:

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

ResourceEvent

A high level event that occured in the processing of this document.

- The field namespace URI is `http://ns.adobe.com/xap/1.0/sType/ResourceEvent#`
- The preferred field namespace prefix is `stEvt`

Field Name	Value Type	Description
action	open Choice	The action that occurred. Defined values are: `converted`, `copied`, `created`, `cropped`, `edited`, `filtered`, `formatted`, `version_updated`, `printed`, `published`, `managed`, `produced`, `resized` New values should be verbs in the past tense.
instanceID	URI	The instance ID of the modified resource.
parameters	Text	Additional description of the action.
softwareAgent	AgentName	The software agent that performed the action.
when	Date	Optional timestamp of when the action occurred.

ResourceRef

A multiple part reference to a resource. Used to indicate prior versions, originals of renditions, originals for derived documents, and so on. The fields present in any specific reference depend on usage and on whether the referenced resource is managed. Except for instanceID, the fields are all properties from the referenced resource's xmpMM schema.

- Field namespace URI is `http://ns.adobe.com/xap/1.0/sType/ResourceRef#`
- The preferred field namespace prefix is `stRef`

Field Name	Value Type	Description
instanceID	URI	The referenced resource's instance ID.
documentID	URI	The referenced resource's xmpMM:DocumentID.
versionID	Text	The referenced resource's xmpMM:VersionID.
renditionClass	RenditionClass	The referenced resource's xmpMM: RenditionClass.
renditionParams	Text	The referenced resource's xmpMM: RenditionParams.
manager	AgentName	The referenced resource's xmpMM:Manager.
managerVariant	Text	The referenced resource's xmpMM: ManagerVariant.
manageTo	URI	The referenced resource's xmpMM:ManageTo.
manageUI	URI	The referenced resource's xmpMM:ManageUI.

Version

Describes one version of a document.

- The field namespace URI is `http://ns.adobe.com/xap/1.0/sType/Version#`
- The preferred field namespace prefix is `stVer`

Field Name	Value Type	Description
comments	Text	Comments concerning what was changed.
event	ResourceEvent	High level, formal description of what operation the user performed.
modifyDate	Date	The date on which this version was checked in.
modifier	ProperName	The person who modified this version.
version	Text	The new version number.

Basic Job/Workflow Value Types

The following value type is used for the Basic Job/Workflow schema.

Job

Describes a job for a job-management system.

- The field namespace URI is `http://ns.adobe.com/xap/1.0/sType/Job#`
- The field namespace prefix is `stJob`

Field Name	Value Type	Description
name	Text	Informal name of job. This name is for user display and informal systems.
id	Text	Unique ID for the job. This field is a reference into some external job management system.
url	URL	A file URL referencing an external job management file.

EXIF Schema Value Types

These types are used only within the EXIF schema.

Rational

To represent EXIF rational values in XMP, they must be converted to text. The recommended approach is to use a value of type Text of the form: numerator; forward slash (‘/’); denominator. For example, the value 2/3 becomes the text value `"2/3"` when converted to XMP.

GPSCoordinate

A Text value in the form “`DD,MM,SSk`” or “`DD,MM.mmk`”, where:

- `DD` is a number of degrees
- `MM` is a number of minutes
- `SS` is a number of seconds
- `mm` is a fraction of minutes
- `k` is a single character `N`, `S`, `E`, or `W` indicating a direction (north, south, east, west)

Flash

A structure describing the flash state.

Field	Value Type	Description
Fired	Boolean	`True` if flash fired.
Return	Closed Choice	Whether strobe return is supported and if supported, detected. One of: 0 = no strobe return detection 2 = strobe return light not detected 3 = strobe return light detected
Mode	Closed Choice	The flash mode. One of: 0 = unknown 1 = compulsory flash firing 2 = compulsory flash suppression 3 = auto mode
Function	Boolean	`True` if flash function is not present.
RedEyeMode	Boolean	`True` if red-eye reduction is supported.

OECF/SFR

A structure describing the OECF/SFR.

Field	Value Type	Description
Columns	Integer	Number of columns, *n*.
Rows	Integer	Number of rows, *m*.
Names	seq Text	Column item names, *n* entries.

Values	seq Rational	OECF/SFR values, sequence should be, in order:
		`value [0,0]`
		…
		`value [n − 1, 0]`
		`value [0, m − 1]`
		…
		`value [n − 1, m − 1]`

CFAPattern

A structure describing the CFA pattern.

Field	Value Type	Description
Columns	Integer	Number of columns, *n*.
Rows	Integer	Number of rows, *m*.

Values	seq Integer	CFA values, sequence should be, in order:
		`value [0,0]`
		…
		`value [n − 1, 0]`
		`value [0, m − 1]`
		…
		`value [n − 1, m − 1]`

DeviceSettings

A structure describing the device settings.

Field	Value Type	Description
Columns	Integer	Display columns.
Rows	Integer	Display rows.
Settings	seq Text	Camera settings, in order.

Extensibility of Schemas

This section discusses how to create new schemas and extend existing ones.

Creating Custom Schemas

The schemas defined in this document are core schemas that are believed to be applicable to a wide variety of needs. If possible, it is always desirable to use properties from existing schemas. However, XMP was designed to be easily extensible by the addition of custom schemas. If your metadata needs are not already covered by the core schemas, you can define and use your own schemas.

If you are considering creating a new namespace, observe the following:

- Avoid including properties that have the same semantics as properties in existing namespaces.
- If your properties might be useful to others, try to collaborate in creating a common namespace, to avoid having a multitude of incompatible ones.

To define a new schema, you should write a human-readable schema specification document The specification document should be made available to any developers who need to write code that understands your metadata.

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

Your specification document should include:

- A unique name for your schema in the form of a URI and a preferred prefix.
- A table containing the name of each property, the value type, and the description of the property.

If you define properties that have structured value types, you may wish to use additional URI names to identify the components of a structured property value.

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

Extending Schemas

Keep in mind the following points when extending a schema:

- New properties may be added to existing namespaces without “breaking” applications.
- The definitions of properties in existing namespaces should always remain the same; otherwise, applications may produce incorrect behavior. If it is necessary to change the meaning of a property, a new property should be created (and the old one declared as deprecated).
- It is possible to create a “new version” of a schema namespace. However, there is no logical connection between the old version and the new version. The same name in two namespaces refers to two different properties.

5 Embedding XMP Metadata in Application Files

This chapter describes how XMP metadata in XMP Packets is embedded in a variety of file formats. Document interchange is best achieved by applications that understand how XMP is embedded.

These descriptions assume that the reader has a working knowledge of the referenced file formats.

- TIFF
- JPEG
- JPEG 2000
- GIF
- PNG
- HTML
- PDF
- AI (Adobe Illustrator)
- SVG/XML
- PSD (Adobe Photoshop)
- PostScript and EPS.

TIFF

In TIFF files, an entry in the Image File Directory (IFD) points to the XMP Packet. That entry has a Tag value of 700 (decimal), as shown here:

Byte offset	Field value	Field name	Comments
0, 1	700	TAG	Tag that identifies the field (decimal value).
2, 3	1	Field type	The field type is BYTE, which is represented as a value of 1.
4–7		Count	The total byte count of the XMP Packet.
8–11		Value or Offset	The byte offset of the XMP Packet.

Reference

TIFF 6.0 Specification:

http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf

JPEG

In JPEG files, an APP1 marker designates the location of the XMP Packet. The following table shows the entry format.

Byte Offset	Field value	Field name	Length (bytes)	Comments
0	0xFFE1	APP1	2	APP1 marker.
2	2 + length of namespace (29) + length of XMP Packet	Lp	2	Size in bytes of this count plus the following two portions
4	Null-terminated ASCII string without quotation marks.	namespace	29	XMP namespace URI, used as unique ID: `http://ns.adobe.com/xap/1.0/`
33	< XMP Packet >			

The header plus the following data must be less than 64 KB bytes. The XMP Packet cannot be split across the multiple markers, so the size of the XMP Packet must be at most 65502 bytes.

References

- JPEG File Interchange Format Version 1.02
- ISO/IEC 10918-1 Information technology - Digital Compression and Coding of continuous-tone still images: requirements and guidelines.
- ISO/IEC 10918-4 Information technology — Digital compression and coding of continuous-tone still images: Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF color spaces, APP*n* markers, SPIFF compression types and Registration Authorities (REGAUT)

  This specifies the format of APP*n* markers and the file interchange format.

JPEG 2000

The JP2 format consists of a set of “boxes”. XMP packets are stored in a UUID box, as shown in the following table:

Field value	Field name	Length (bytes)	Comments
Entire length in bytes (including the four used for this field)	`Length`	4	Big-endian unsigned integer
`0x75756964` (‘uuid’)	`Type`	4	Big-endian unsigned integer

`BE 7A CF CB`
`97 A9 42 E8`
`9C 71 99 94`
`91 E3 AF AC`
	`UUID`	16	16-byte binary UUID as defined by ISO/IEC 11578:1996

< XMP Packet >	`DATA`		

References

Information about the JPEG 2000 standard can be found at http://www.jpeg.org/JPEG2000.html.

GIF

In a GIF89a file, an XMP Packet is in an Application Extension (see the following figure). Its Application Identifier is `'XMP Data'` and the Application Authenticator is `'XMP'`. The Application Data consists of the XMP Packet, which must be encoded as UTF-8, followed by a 258-byte “magic” trailer, whose values are 0x01, 0xFF, 0xFE, 0xFD ....0x03, 0x02, 0x01, 0x00, 0x00. The final byte is the Block Terminator.

XMP in GIF File Format:

	7 6 5 4 3 2 1 0	Field Name	Type
0	0x21	Extension Introducer	Byte
1	0xFF	Extension Label	Byte
0	0x0B	Block Size	Byte
1	‘X’ 0x58	Application Identifier	8 Bytes
2	‘M’ 0x4D		
3	‘P’ 0x50		
4	‘ ’ 0x20		
5	‘D’ 0x44		
6	‘a’ 0x61		
7	‘t’ 0x74		
8	‘a’ 0x61		
9	‘X’ 0x58	Application Authentication Code	3 Bytes
10	‘M’ 0x4D		
11	‘P’ 0x50		
	<XMP Packet>	XMP Packet, UTF-8 encoded	Byte
	0x01	“Magic trailer”	258 Bytes
	0xFF		
	0xFE		
	⋮		
	0x01		
	0x00		
	0x00	Block Terminator	Byte

The XMP must be UTF-8-encoded, for the following reasons. GIF actually treats the Application Data as a series of GIF data sub-blocks. The first byte of each sub-block is the length of the sub-block’s content, not counting the first byte itself. To consume the Application

Data, a length byte is read. If it is non-zero, that many bytes of data are read, followed by the next length byte. The series ends when a zero length byte is encountered.

When XMP is encoded as UTF-8, there are no zero bytes in the XMP Packet. Therefore, software that is unaware of XMP views packet data bytes as sub-block lengths, and follows them through the packet accordingly, eventually arriving somewhere in the magic trailer. The trailer is arranged so whichever byte is encountered there will cause a skip to the Block Terminator at the end.

Reference

The GIF 89a specification is available at http://members.aol.com/royalef/gif89a.txt

PNG

An XMP Packet is embedded in a PNG graphic file by adding a chunk of type `iTXt`. This chunk is semantically equivalent to the `tEXt` and `zTXt` chunks, but the textual data is in the UTF-8 encoding of the Unicode character set, instead of Latin-1.

The Chunk Data portion is the XMP Packet. The packet must be marked as read-only.

NOTE: XMP software that is not aware of the file format must not be allowed to change the content of the XMP Packet because of the CRC checksum following the chunk data.

There should be no more than one chunk containing XMP in each PNG file. Encoders are encouraged to place the chunk at the beginning of the file, but this is not required.

PNG Data Format

Field	Length	Comments
Length	4	An unsigned integer representing the number of bytes in the chunk's data field (does not include the chunk type code or the CRC).
Chunk Type	4	`"iTXt"`
Chunk Data: Standard `iTXt` chunk header plus the XMP Packet		
Keyword	17	`"XML:com.adobe.xmp"`
Null separator	1	value = 0x00
Compression flag	1	value = 0x00, specifies uncompressed data.
Compression method	1	value = 0x00
Language tag	0	Not used for XMP metadata.
Null separator	1	value = 0x00
Translated keyword	0	Not used for XMP metadata.
Null separator	1	value = 0x00
Text	length of packet	<XMP Packet>
CRC	4	The Cyclic Redundancy Check, calculated on the preceding bytes in the chunk, including the chunk type code and chunk data fields, but not including the length field.

Reference

http://www.w3.org/TR/REC-png.html

HTML

XMP embedded in HTML should conform to one of the W3C recommendations for embedding XML in HTML; see Reference below.

XML can be embedded in a SCRIPT or XML element, placed in any legal location; the suggested location is the end of the HEAD element. The content of the SCRIPT or XML element is the XMP Packet.

The browser must recognize the SCRIPT or XML element so that text representing the value of RDF properties is not displayed as page content. Using the XML element is preferred unless there are known incompatibilities with older software; if so, the SCRIPT element is likely to be recognized.

There are three approaches to embedding XML in HTML, as shown in the examples below. Two use the SCRIPT element, and the third uses the XML element.

Using the SCRIPT element and LANGUAGE attribute

```
<html>
 <head>
  <SCRIPT LANGUAGE="XML">
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
  <?xpacket end='w'?>
  </SCRIPT>
 </head>
 <body>
 </body>
</html>
```

NOTE: Adobe has noticed problems with using the SCRIPT element and LANGUAGE attribute in Microsoft Word 2000 running under Microsoft Windows XP: the body content cannot be displayed.

Using the SCRIPT element and TYPE attribute

```
<html>
 <head>
  <SCRIPT TYPE="text/xml">
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
  <?xpacket end='w'?>
  </SCRIPT>
 </head>
 <body>
 </body>
</html>
```

Using the XML element

```
<html>
 <head>
  <XML>
  <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
  <!-- The serialized RDF goes here. It is removed for brevity. -->
  <?xpacket end='w'?>
  </XML>
 </head>
 <body>
 </body>
</html>
```

Reference

The meeting report for the May 1998 W3C meeting: http://www.w3.org/TR/NOTE-xh.

PDF

For PDF files, the XMP Packet is embedded in a metadata stream contained in a PDF object (beginning with PDF 1.4).

This is a partial example of XMP metadata embedded as an XMP Packet, stored as a metadata stream:

```
1152 0 obj
<< /Type /Metadata /Subtype /XML /Length 1706 >>
stream
<?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
<!-- The serialized RDF goes here. It has been removed for brevity. -->
<?xpacket end='w'?>
endstream
endobj
```

PDF files that have been incrementally saved can have multiple packets that all look like the "main" XMP metadata. During an incremental save, new data (including XMP Packets) is written to the end of the file without removing the old. Top-level PDF dictionaries are also rewritten, so an application that understands PDF can check the dictionary to find only the new packet.

Reference

Full documentation on metadata streams in PDF files is available in the *PDF Reference*, Version 1.5, at http://partners.adobe.com/asn/tech/pdf/specifications.jsp

AI (Adobe Illustrator)

An `.ai` file generated by Adobe Illustrator® is in the Portable Document Format (PDF). Hence, the format for embedding XMP metadata is the same as for PDF files.

SVG/XML

XMP metadata, because it is legal XML, can be directly embedded within an XML document. An XMP Packet is not intended to be a complete standalone XML document; therefore it contains no `XML` declaration. The XMP Packet can be placed anywhere within the XML document that an element or processing instruction would be legal.

It is recommended that the file be encoded as Unicode using UTF-8 or UTF-16. This provides compatibility for software that scans for XMP Packets and parses just their content.

Reference

The XML specification is available at http://www.w3.org/TR/REC-xml

PSD (Adobe Photoshop)

Adobe Photoshop® `.psd` files contain image resource blocks, which are used to store non-pixel data associated with an image. The following table shows the format of an image resource block:

Field	Type	Description
`Type`	OSType	Photoshop always uses its signature, 8BIM.
`ID`	2 bytes	ID = 1060 for XMP metadata.
`Name`	PString	A Pascal string, padded to make size even (that is, an extra zero byte is appended if needed). A null name consists of two bytes of 0. For Photoshop 7, XMP metadata uses a `Name` value of `"XMP"`.
`Size`	4 bytes	Actual size of resource data. This does not include the Type, ID, Name, or Size fields.
`Data`	Variable	Resource data, padded to make size even. This is the XMP Packet.

For the Name and Data fields in the above table, “padded to make size even” means that an extra zero byte is appended to the “raw” field value, if necessary.

PostScript and EPS

XMP metadata can be placed in PostScript® or EPS files, for use in either PostScript or PDF workflows. This section describes how to place XMP into PostScript or EPS for both the outer document level (main XMP) and for internal objects such as an image (object XMP). It also specifically discusses issues involving Acrobat Distiller, since workflows often use Distiller to produce PDF from PostScript and EPS.

NOTE: This does not imply that use of Distiller is necessary, or that other application issues do not exist.

There are three important "flavors" of PostScript files that can affect how XMP is written, found, and used. They are:

- DSC PostScript (or just "PostScript"): PostScript conforming to the DSC conventions defined in Appendix G of the PostScript Language Reference.
- Raw PostScript: PostScript following no particular structural conventions. The use of raw PostScript is discouraged. As mentioned in "Ordering of Content" on page 86, a special DSC comment is required to support fast and reliable location of the main XMP.
- EPS: PostScript conforming to the EPS conventions defined in Appendix H of the PostScript Language Reference. EPS is a subset of DSC PostScript.

Because of common usage issues, document-level XMP should be written differently for PostScript and EPS. Object-level XMP is written identically for PostScript and EPS.

Document-Level Metadata

As with any file format, locating contained XMP in PostScript or EPS is most reliably done by fully processing the file format. For PostScript, this means executing the PostScript interpreter. Packet scanning is not reliable whenever a file contains multiple XMP packets, or object XMP without main XMP.

It is often worthwhile to find the main XMP and ignore (at least temporarily) object XMP. Interpretation of the entire PostScript file to locate the main XMP can be very expensive. A hint and careful ordering are used to allow a combination of XMP packet scanning and PostScript comment scanning to reliably find the main XMP.

To write document-level metadata in PostScript, an application must:

- Write the `%ADO_ContainsXMP` comment as described under "Ordering of Content" on page 86.
- Write the XMP packet as described under "Document-Level XMP in PostScript" on page 87.

To write document-level metadata in EPS an application must:

- Write the %ADO_ContainsXMP comment as described under "Ordering of Content" on page 86.
- Write the XMP packet as described under "Document-Level XMP in EPS" on page 88.

Use of raw PostScript is discouraged specifically because it lacks the %ADO_ContainsXMP comment. If raw PostScript must be used, the XMP must be embedded as described under "Document-Level XMP in PostScript" on page 87.

Ordering of Content

Many large publications use PostScript extensively. It is common to have very large layouts with hundreds or thousands of placed EPS files. Because PostScript is text, locating XMP embedded within PostScript in general requires parsing the entire PostScript program, or at least scanning all of its text. Placed PostScript files can be quite large. They can even represent compound documents, and might contain multiple XMP packets. For PostScript files containing XMP at all, the entire file would have to be searched to make that simple determination.

All of this presents performance challenges for layout programs that want to process XMP embedded in PostScript. As a pragmatic partial solution, a special marker comment can be placed in the PostScript header comments to provide advice about locating the main XMP. This marker must be before the %%EndComments line.

The purpose of this marker is to tell applications consuming the PostScript whether a main XMP is present at all, and how to look for the main XMP. The form of the XMP marker is:

```
%ADO_ContainsXMP: <option> ...
```

An option is a contiguous sequence of characters that does not include spaces, tabs, linefeeds, or carriage returns; options are case sensitive. There must be no whitespace before the colon. Applications should ignore options they do not understand.

There are three options defined at present. They are mutually exclusive and provide a hint about how to find the main XMP. Note that the main XMP is not necessarily the document-level XMP:

- MainFirst: the main XMP is the first XMP packet in the file and is located near the front of the file. The XMP should be in front of as much PostScript content as possible.
- MainLast: the main XMP is the last XMP packet in the file and is located near the back of the file. The XMP should be behind as much PostScript content as possible.
- NoMain: there is no main XMP packet for the PostScript file. The file might still contain XMP packets, for example within embedded EPS sections or attached to internal objects.

NOTE: The XMP location option applies to both the location of the main XMP in the file and to its position relative to other object-level XMP. The main XMP packet must be before all other XMP if MainFirst is used; it must be after all other XMP if MainLast is used. It is not necessary for the other XMP packets to be adjacent to the main packet.

NOTE: When EPS files are concatenated, it is necessary to provide a new set of PostScript header comments for the aggregate, and optionally new a main XMP packet. Otherwise the XMP marker comment from the first EPS portion would erroneously be taken to refer to the aggregate.

Document-Level XMP in PostScript

This section assumes that PostScript devices are level 2 or newer, and that Distiller version 6.0 or newer is used. Compatibility issues are discussed in "Compatibility With Distiller 5 for PostScript" on page 90 and "LanguageLevel 1 for PostScript and EPS" on page 90.

There are three main steps to setting up the document-level XMP:

1. Creating a PostScript stream object to contain the XMP.
2. Placing the XMP into the stream object.
3. Associating the XMP stream object with the document.

XMP metadata must be embedded in a PostScript file in a way that it will be recognized by software that scans files for metadata, which means embedding the complete XMP packet. However, if that file were sent to a PostScript output device, the packet data would cause PostScript errors and the job would fail. To be able to handle arbitrary data, we need a procedure to read the XMP data from the current file, and discard the data if it is not intended to be interpreted.

NOTE: In what follows, we define some procedures in a private dictionary like:

```
privatedict /metafile_pdfmark {flushfile cleartomark} bind put
```

The name `privatedict` is for illustration purpose only. In the real product code, these procedures should be defined in a unique dictionary so that several EPS files can be used in one document and slightly different versions of these procedures can co-exist.

Here is an example that shows how to embed document-level XMP in PostScript. This example does not include the required marker comment.

```
% ==========================================================================
% We start with some Postscript prolog. This defines operators and
% procedures that we will use when processing the XMP metadata.

% Define pdfmark to cleartomark, so the data is discarded when consumed
% by a PostScript printer or by Distiller 4.0 or earlier. All following
% references to "privatedict" should be changed to a unique name to
% avoid potential conflicts. This is discussed later in the section
% "Avoiding Name Conflicts" on page 89.

/currentdistillerparams where
{pop currentdistillerparams /CoreDistVersion get 5000 lt} {true} ifelse

{privatedict /pdfmark /cleartomark load put
```

```
   privatedict /metafile_pdfmark {flushfile cleartomark} bind put}
{privatedict /metafile_pdfmark {/PUT pdfmark} bind put} ifelse

% ========================================================================
% We now create the stream containing the XMP metadata. This must follow
% the prolog shown above, but does not need to be adjacent to it.

% Create a pdfmark named stream object to hold the data. As with the
% privatedict above, use of a unique name is recommended, not literally
% my_metadata_stream_123. The name of this stream is local to the
% Postscript program, it has no outside significance.
%
% First define the stream object, then read the XMP packet into the
% stream, finally attach the stream as the main XMP.
%
% The "&&end XMP packet marker&&" comment is significant, it terminates
% the reading of the XMP packet.

% First: Create the XMP metadata stream object and say that it is XMP.
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
[{my_metadata_stream_123} 2 dict begin
   /Type /Metadata def /Subtype /XML def currentdict end /PUT pdfmark

% Second: Fill the stream with the XMP packet.
[{my_metadata_stream_123}
   currentfile 0 (% &&end XMP packet marker&&)
   /SubFileDecode filter metafile_pdfmark

... XMP packet goes here ...

% &&end XMP packet marker&&

% Third: Attach the stream as the main XMP metadata stream.
[{Catalog} {my_metadata_stream_123} /Metadata pdfmark
```

Document-Level XMP in EPS

Embedding XMP inside EPS is very similar to PostScript; however, there are issues raised by the common practice of embedding EPS within other EPS or PostScript. The notion of document-level XMP in EPS really means outermost XMP in the EPS. This will be document-level XMP in the PDF if the EPS is distilled alone. This will be appropriate marked content if the EPS is embedded in other EPS or PostScript.

The solution for EPS requires:

- The XMP must be placed before all EPS content (PostScript drawing commands).
- The `/BDC` and `/EMC` **pdfmark**s must be used to bracket the EPS content.
- The third XMP setup step uses different PostScript code.

Here is an abbreviated example, modified from the previous example:

```
%%EndPageSetup
[/NamespacePush pdfmark

... Do all of the XMP setup as above, up to step 3 ...

% Third: Attach the stream to the Marked Content dictionary.
% All drawing commands must be between the /BDC and /EMC operators.
[/Document 1 dict begin
   /Metadata {my_metadata_stream_123} def currentdict end /BDC pdfmark
[/NamespacePop pdfmark

... All drawing commands go here ...

%%PageTrailer
[/EMC pdfmark
```

Avoiding Name Conflicts

In the samples, we used the name *{my_metadata_stream_123}* and suggested that some form of unique name be used. The recommended approach is to generate a typical UUID and strip out all but the significant alphanumeric characters. Use this as a suffix to the name.

An alternate solution is to use NamespacePush and NamespacePop **pdfmark**s. This is also the recommended solution in the *Pdfmark Reference Manual* (it is accessible from Distiller's **Help** menu.) This is preferable if possible, but might require large and untenable separation of the push an pop.

It is important to put all pdfmarks using the named objects in the same block bracketed by NamespacePush and NamespacePop pair; for example, the following PostScript code is bad:

```
[/NamespacePush pdfmark
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
[{my_metadata_stream_123} 2 dict begin
   /Type /Metadata def /Subtype /XML def currentdict end /PUT pdfmark
[{my_metadata_stream_123}
   currentfile 0 (% &&end XML Packet marker&&)
/SubFileDecode filter metafile_pdfmark
... XML Packet goes here ...
% &&end XML Packet marker&&
[/NamespacePop pdfmark
% At this point, the name {my_metadata_stream_123} is no longer usable.
% next line will cause "undefined" error:
[{Catalog} {my_metadata_stream_123} /Metadata pdfmark
```

Compatibility With Distiller 5 for PostScript

Acrobat Distiller version 5 was the first to include XMP support, but it does not support the /Metadata **pdfmark**. There is no easy way to attach document-level XMP with Distiller 5. It will ignore the /Metadata **pdfmark**, without signaling a PostScript error.

LanguageLevel 1 for PostScript and EPS

The `SubFileDecode` filter became available in PostScript LanguageLevel 2. If the PostScript or EPS containing XMP must be processed by PostScript LanguageLevel 1 devices, such as older printers, another approach must be used to read the XMP into the stream object.

With PostScript LanguageLevel 1, there are at least two approaches: using `readstring` to read in the whole XMP packet, or `readline` to read in the XMP packet data line by line until an end marker is found.

We present the `readline` approach here. The `readline` approach solves two problems that exist for `readstring`:

- We don't have to know the exact size of the whole packet, just need to know the maximum length of the lines.
- The exact length of an XMP packet may change if the PostScript/EPS file is re-saved by a text editor with different line ending convention, CR, LF, or CRLF.

Here is an example showing how to use the `readline` approach for PostScript. It is very similar overall to the earlier example, differing only in step 2 and related prolog:

```
% ================================================================
% We start with some Postscript prolog. This defines operators and
% procedures that we will use when processing the XMP metadata.

% Define pdfmark to cleartomark, so the data is discarded when consumed
% by a PostScript printer or by Distiller 4.0 or earlier. All following
% references to “privatedict” should be changed to a unique name to
% avoid potential conflicts. This is discussed later in the section
% “Avoiding Name Conflicts”.

/currentdistillerparams where
{pop currentdistillerparams /CoreDistVersion get 5000 lt} {true} ifelse
{privatedict /pdfmark /cleartomark load put} if

% Define another procedure to read line by line from current file until
% marker line is found. The maximum line length is used to create a
% temporary buffer for reading the XMP lines.
% On stack: [ {name} maxLineLength MarkerString

privatedict /metastring_pdfmark
{ 2 dict begin
/markerString exch def string /tmpString exch def
{ currentfile tmpString readline pop
```

```
markerString anchorsearch
{pop pop cleartomark exit}
{3 copy /PUT pdfmark pop 2 copy (\n) /PUT pdfmark} ifelse
} loop
end
}bind put

% ========================================================================
% We now create the stream containing the XMP metadata. This must follow
% the prolog shown above, but does not need to be adjacent to it.

% Create a pdfmark named stream object in PDF to hold the data. As with
% privatedict above, use of a unique name is recommended, not literally
% my_metadata_stream_123. The name of this stream is local to the
% Postscript program, it has no outside significance.
%
% First define the stream object, then read the XMP packet into the
% stream, finally attach the stream as the main XMP.

% The <LineLength> below must be replaced with a value larger than the
% longest line in the XMP packet. There is no safe and general way to
% exactly determine this, the XMP can be modified in place after the
% Postscript is written and could legally all be on one line.
%
% The overall length of the packet cannot change though. You should set
% the <LineLength> to the lesser of the packet size and 65500. The upper
% limit keeps this within the 64KB limit of PostScript strings.
%
% The "&&end XML Packet marker&&" comment is significant, it terminates
% the reading of the XMP packet.

% First: Create the XMP metadata stream object and say that it is XMP.
[/_objdef {my_metadata_stream_123} /type /stream /OBJ pdfmark
[{my_metadata_stream_123} 2 dict begin
   /Type /Metadata def /Subtype /XML def currentdict end /PUT pdfmark

% Second: Read the XMP packet into the stream.
[{my_metadata_stream_123} <LineLength>
   (% &&end XMP Packet marker&&) metastring_pdfmark

... XMP packet goes here ...

% &&end XMP Packet marker&&

% Third: Attach the stream as the main XMP metadata stream.
[{Catalog} {my_metadata_stream_123} /Metadata pdfmark
```

Traditional PDF Metadata and XMP

The discussion here is primarily about explicitly embedding XMP in PostScript and EPS to provide metadata. However, when Distiller is used the document-level metadata in the PDf file can contain information that comes from other sources than the XMP embedded in the PostScript. This is metadata that traditionally went into the PDF document information dictionary, and with the advent of XMP is replicated in the PDF's document-level XMP.

There are two other methods for putting metadata in a PostScript file so Distiller will put it in the PDF document info dictionary and also create and embed an XMP packet for that data in the PDF document. You can use:

- DSC (Document Structuring Conventions) comments. The DSC comments are processed only if DSC parsing is enabled, that is, only if the job file contains the following line:

  ```
  /ParseDSCCommentsForDocInfo true
  ```

- `DOCINFO` **pdfmark** command. Information on **pdfmark** is available from the Distiller application **Help** menu, under "pdfmark Guide."

If more than one of the three possible sources of metadata for the PDF file are present, then a property value in the document-level XMP is taken from the first of these sources in the PostScript used to create the PDF that contains the property:

- Explicit document-level XMP.
- Explicit document info dictionary.
- DSC comments.

Because the **pdfmark** command is more reliable than DSC comments, many applications use it to set DocInfo properties for a PDF document. The following is an example of PostScript code, created by FrameMaker, which illustrates the use of the `DOCINFO` **pdfmark** operator:

```
/Creator (FrameMaker 6.0)
/CreationDate (D:20020214144924)
/ModDate (D:20020215142701)
/Author(John Doe)
/Title (Processing XMP Data in EPS Files)
/Subject (XMP)
/Keywords (XMP, pdfmark)
/DOCINFO pdfmark
```

Distiller will place these seven properties – plus "Producer" – into the resulting PDF file in two places: the document information dictionary and document-level Metadata as an XML Packet. The Producer is the product name, e.g. "Acrobat Distiller 5.0 (Windows)." It is possible to add other Key/Value pairs to PDF DocInfo, but they are not added to the document-level Metadata in Distiller 5.0.

Care must be taken if the file might be sent to a PostScript interpreter instead of to Distiller. Some PostScript interpreters may not recognize the **pdfmark** command, e.g. those in older printers. One way to avoid problems is to conditionally define the **pdfmark** operator to the "`cleartomark`" operator. This is shown in the earlier examples.

Object-Level Metadata

Object-level XMP is written identically for PostScript and EPS.

Metadata streams can be attached to specific objects in a PostScript file using the **pdfmark** operator. This is identical to the document-level PostScript method (see "Document-Level XMP in PostScript" on page 87), except that in step 3 the stream containing the XMP metadata is attached to the object. An example follows showing this for an image:

```
% ===========================================================================
% We assume that the XMP stream has been defined as shown earlier. All
% but the third step, defining the stream as a metadata stream. We also
% assume that the image has been defined as {myImage_123}. Again, a
% unique name should be used.
%
% The third step is replaced with one that associates the XMP metadata
% with the image. Since this must be located after both the image and
% XMP streams, it might not be adjacent to the other XMP parts. See the
% ordering issues discussed in "Ordering of Content".

% Third: Attach the XMP metadata stream to the image.
[{myImage_123} <</Metadata {my_metadata_stream_123}>> /PUT pdfmark
```

NOTE: The approach shown here is compatible with all PostScript devices. That is, no additional changes are needed to ensure that level 1 devices will properly ignore the XMP beyond those already mentioned, and Distiller 5 and later will attach the XMP to the associated object in the PDF file.

NOTE: Although Distiller 5 will attach the XMP to the associated object in the PDF file, the XMP stream in the PDF will be Flate-compressed. This makes the object XMP packet in the PDF invisible to external packet scanners. The XMP will be visible to software processing the PDF format and decompressing the stream. Distiller 6 and later do not compress the XMP packet stream.