



Adding Intelligence to Media

# **XMP SPECIFICATION PART 3**

## **STORAGE IN FILES**

Copyright © 2008 Adobe Systems Incorporated. All rights reserved.

*Extensible Metadata Platform (XMP) Specification: Part 3, Storage in Files*

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

Adobe, the Adobe logo, Acrobat, Acrobat Distiller, Flash, FrameMaker, InDesign, Illustrator, Photoshop, PostScript, and XMP are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Macintosh, Mac OS and QuickTime are trademarks of Apple Computer, Inc., registered in the United States and other countries. UNIX is a trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd. All other trademarks are the property of their respective owners.

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

# Contents

<b>Preface</b> .....	<b>6</b>
About this document .....	6
How this document is organized .....	6
Conventions used in this document .....	7
Where to go for more information .....	7
<b>1 Introduction</b> .....	<b>8</b>
Embedding metadata in files .....	8
External storage of metadata .....	8
Native metadata .....	9
<b>2 XMP Packets</b> .....	<b>10</b>
The XMP packet format .....	10
Header .....	11
XMP data .....	12
Padding .....	12
Trailer .....	12
Scanning files for XMP packets .....	13
Scanning hints .....	13
<b>3 Embedding XMP Metadata in Application Files</b> .....	<b>15</b>
Image formats .....	15
DNG (Digital Negative) .....	16
GIF (Graphic Interchange Format) .....	16
JPEG .....	18
JPEG 2000 .....	21
PNG (Portable Network Graphics) .....	22
TIFF (Tagged Image File Format) .....	23
Dynamic media formats .....	26
ASF (WMA, WMV) .....	26
AVI .....	28
FLV (Flash Video) .....	29
MOV (QuickTime) .....	30
MP3 .....	31
MPEG-2 .....	31
MPEG-4 .....	31
SWF (Flash) .....	32
WAV .....	33
Video package formats .....	33
AVCHD .....	34
P2 .....	34
Sony HDV (High Definition Video) .....	35

XDCAM EX .....	36
XDCAM FAM .....	36
XDCAM SAM .....	37
Adobe application formats .....	37
AI (Adobe Illustrator) .....	38
INDD, INDT (Adobe InDesign) .....	38
PSD (Adobe Photoshop) .....	42
Markup formats .....	44
HTML .....	44
XML .....	45
Document formats .....	45
PDF .....	45
PS, EPS (PostScript and Encapsulated PostScript) .....	46
UCF (Universal Container Format) .....	53
<b>4 Handling Native Metadata .....</b>	<b>55</b>
Reconciling metadata in different formats .....	55
Text encodings in import and export .....	55
Native metadata export policy .....	56
Native metadata import policy .....	56
Native metadata in PDF files .....	57
User-defined keys .....	58
Resolving metadata conflicts .....	58
Native metadata in dynamic media formats .....	59
Native metadata in ASF (WMA, WMV) .....	59
Native metadata in AVI .....	60
Native metadata in MP3 .....	60
Native metadata in MPEG-4 .....	61
Native metadata in WAV .....	62
Native metadata in digital photography formats .....	62
Reconciliation issues .....	64
Encoding of text in metadata .....	64
<b>5 Digital Photography Native Metadata .....</b>	<b>66</b>
Metadata storage in native formats .....	66
TIFF metadata .....	66
Exif metadata .....	66
Photoshop image resources .....	67
IPTC (IIM) metadata .....	68
Reconciling metadata properties .....	70
Photoshop image resources for metadata .....	70
IPTC DataSets for metadata .....	71
TIFF and Exif tags for metadata .....	73
Metadata storage .....	81
Storage by metadata form .....	81
Metadata storage in JPEG files .....	82
Metadata storage in Photoshop files .....	83
Metadata storage in TIFF files .....	83

Metadata storage in Mac OS file resources ..... 85

# Preface

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

The specification has three parts:

- *Part 1, Data and Serialization Model*, covers the basic metadata representation model that is the foundation of the XMP standard format. The Data Model prescribes how XMP metadata can be organized; it is independent of file format or specific usage. The Serialization Model prescribes how the Data Model is represented in XML, specifically RDF.

This document also provides details needed to implement a metadata manipulation system such as the XMP Toolkit (which is available from Adobe).

- *Part 2, Standard Schemas*, provides detailed property lists and descriptions for standard XMP metadata schemas; these include general-purpose schemas such as Dublin Core, and special-purpose schemas for Adobe applications such as Photoshop. It also provides information on extending existing schemas and creating new schemas.
- *Part 3, Storage in Files*, provides information about how serialized XMP metadata is packaged into XMP packets and embedded in different file formats. It includes information about how XMP relates to and incorporates other metadata formats, and how to reconcile values that are represented in multiple metadata formats.

## About this document

This document, *XMP Specification Part 3, Storage in Files*, describes how XMP metadata is embedded within various file formats. This includes information about the reconciliation of XMP metadata with other forms of metadata, referred to as *native* (or sometimes *legacy*) metadata.

The intended audience of this document are developers writing file I/O code, users who need to understand the relationship between XMP and native metadata, or others requiring detailed knowledge of file content.

## How this document is organized

This document has the following sections:

- [Chapter 1, "Introduction,"](#) provides an overview of the issues and problems associated with storing metadata in or with different file formats. It includes an introduction to the subject of native metadata formats, and a discussion of external metadata storage.
- [Chapter 2, "XMP Packets,"](#) describes the format of the XMP packet, which wraps serialized XMP for storage in files.
- [Chapter 3, "Embedding XMP Metadata in Application Files,"](#) provides basic information about how XMP packets are embedded in various file formats.

- [Chapter 4, “Handling Native Metadata,”](#) explains general policies of how to reconcile metadata values among XMP and native metadata formats. It provides details of native metadata for simpler cases, and introduces the complexities of native metadata for the digital-photography formats (JPEG, TIFF, and PSD).
- [Chapter 5, “Digital Photography Native Metadata,”](#) provides details of metadata formats used in the digital-photography formats (TIFF/Exif, IPTC, and PSIR), and specifics of how properties in those formats map to XMP properties.

## Conventions used in this document

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

Typeface Style	Used for:
Monospaced bold	XMP property names. For example, <code>xmp:CreationDate</code>
Monospaced Regular	XML code and other literal values, such as value types and names in other languages or formats

## Where to go for more information

See these sites for external specifications mentioned in this document:

XMP Specification	<a href="http://www.adobe.com/devnet/xmp/">http://www.adobe.com/devnet/xmp/</a>
Photoshop SDK	<a href="http://www.adobe.com/devnet/photoshop/">http://www.adobe.com/devnet/photoshop/</a>
TIFF Specification	<a href="http://partners.adobe.com/public/developer/tiff/index.html">http://partners.adobe.com/public/developer/tiff/index.html</a>
JPEG Specification	<a href="http://www.w3.org/Graphics/JPEG/itu-t81.pdf">http://www.w3.org/Graphics/JPEG/itu-t81.pdf</a>
JFIF Specification	<a href="http://www.w3.org/Graphics/JPEG/jfif3.pdf">http://www.w3.org/Graphics/JPEG/jfif3.pdf</a>
Exif Specification	<a href="http://www.exif.org/Exif2-2.pdf">http://www.exif.org/Exif2-2.pdf</a>
IPTC Specification	<a href="http://www.iptc.org/IPTC4XMP/">http://www.iptc.org/IPTC4XMP/</a>
Unicode	<a href="http://www.unicode.org/">http://www.unicode.org/</a>

# 1 Introduction

This document discusses how serialized XMP metadata is packaged and stored with the files it describes. It includes information about how XMP relates to and incorporates other metadata formats, and how to reconcile values that are represented in multiple metadata formats.

This chapter provides a basic overview of the concepts that are important to XMP storage and retrieval:

- Generally, XMP metadata is embedded in the file which the metadata describes; see [“Embedding metadata in files” on page 8](#). The details of how it is embedded vary according to the file format, and are discussed in later chapters.
- It is occasionally appropriate to store metadata separately from the file it describes; this is discussed briefly in [“External storage of metadata” on page 8](#).
- Any application that uses or modifies XMP metadata must be aware of native metadata formats, in order to detect them, preserve them, and reconcile any changes to values among all formats; see [“Native metadata” on page 9](#).

## Embedding metadata in files

XMP metadata is serialized into XML, specifically RFD, for storage in files. The serialized data is known as an *XMP packet*. The structure of the XMP packet is discussed in [Chapter 2, “XMP Packets.”](#)

The XMP packet is completely self-contained and independent of any particular file format. Most file formats predate XMP, and do not have built-in specifications for how to include it. However, XMP can be placed into a file of any format that has a well-defined extension mechanism; that is, a way for developers to customize files for new uses.

File format specifications offer extension mechanisms as a portion of format-specific data that surrounds a chunk of user-defined custom data. In the case of XMP, the XMP packet is the custom data. The surrounding data, as defined by the format specification, identifies the extension’s boundaries and type.

The parameters for embedding custom data vary with different file formats. [Chapter 3, “Embedding XMP Metadata in Application Files,”](#) provides embedding details for a variety of file formats.

## External storage of metadata

It is recommended that XMP metadata be embedded in the file that the metadata describes. 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. 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 values of the `xmpMM:DocumentID`, `xmpMM:InstanceID`, or other appropriate properties 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.)

## Native metadata

Files in many formats can contain metadata in other, previously-defined or *native* formats. Still image formats, for example, frequently contain IPTC and TIFF/Exif metadata. When a file is imported, it can contain metadata in one or more of these native formats. Similarly, when a file is exported, it may be read by a device or application that is expecting metadata in native formats, and may or may not support the XMP format. The information represented overlaps to a great degree, and values that are present in any format should be reflected in all others.

An application that supports XMP must read metadata values from the native formats, and represent it correctly in the XMP metadata. When modifying XMP metadata, an application is responsible for correctly reflecting changes in any other metadata formats that are present. A file that has been manipulated by an application that supports XMP might later be opened in an application that relies on another metadata format, and any changes that have been made to the XMP metadata must be properly represented.

Users of the XMP Toolkit, which handles most native metadata reconciliation, should still be aware of the issues in order to recognize potential problems to look for when testing with legacy application or file versions.

- For general issues and policy on metadata reconciliation, see [Chapter 4, "Handling Native Metadata"](#).
- For an in-depth discussion of complex issues pertaining to still image formats (JPEG, TIFF, and PSD), see [Chapter 5, "Digital Photography Native Metadata"](#).

## 2 XMP Packets

The XMP packet wrapper can enable the use of embedded XMP by software that does not understand the format of the file. The packet wrapper is not the sole aspect of embedding XMP in a file. The entire XMP packet must still be placed in the file as an appropriate component of the file's structure.

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 which are XML syntax-compatible (as is the entire packet) 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 of metadata (if the metadata size does not exceed the packet boundaries);
- allow multiple packets to be embedded in a single data file.

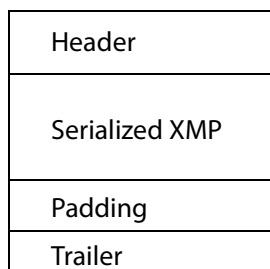
[Chapter 3, "Embedding XMP Metadata in Application Files"](#) gives information on how XMP packets are embedded in specific file formats. Applications may also scan files for XMP packets without knowledge of the file format itself, although this should be regarded as a last resort (see ["Scanning files for XMP packets" on page 13](#)).

### The XMP packet format

The packet wrapper, which consists of a *header*, *padding*, and *trailer* wrapped around serialized XMP) is formally optional; that is, a file that contains only the serialized XMP is legal. The packet wrapper, if used, allows byte-oriented packet scanning and in-place editing of the XMP.

Some file formats or use cases might forbid the packet wrapper; for instance, in cases where minimal size is paramount. It can also happen in cases where the XMP might not be contiguous, and thus byte-oriented packet scanning must be prevented.

The following figure shows a schematic of an XMP packet. It contains a header, XML data, padding, and a trailer.



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"?>

```

In this example, ‘■’ 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 this example packet.

## 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 ASCII space (U+0020) character.

**NOTE:** The astute reader might note that XML processing instructions do not actually contain “attributes”; formally, they have an undifferentiated text body. The term attribute is used here colloquially to denote the general syntax of the `xpacket` processing instruction’s body.

### 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 13](#) 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:

```
W5M0MpCehiHzreSzNTczkc9d
```

**Attribute: bytes**

This attribute is deprecated.

The optional `bytes` attribute specifies the total length of the packet in bytes, which was intended to allow faster scanning of XMP packets. It was of minimal actual value, and would not work properly in text files.

**Attribute: encoding**

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 was intended to specify the character encoding of the packet, but is redundant with the information from 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 *XMP Specification Part 1, Data and Serialization Models*.

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.

**Padding**

It is recommended that applications place 2 KB to 4 KB of padding within 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 ASCII 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. Other unrecognized attributes can follow, but should be ignored. Attributes must be separated by exactly one ASCII 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:

- The value `r` means the packet is “read-only” and must not be updated in place. This would be used for example if a file contained an overall checksum that included the embedded XMP.

The use of the value `r` does restrict the behavior of applications that understand the file format and are capable of properly rewriting the file.

- The value `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

Knowledge of individual file formats provides the best way for an application to get access to XMP packets. See [Chapter 3, “Embedding XMP Metadata in Application Files”](#) for detailed information on how XMP data is stored in specific file formats.

It is always best to use format-aware file parsing when possible. Lacking this information, applications can find XMP packets by scanning the file. Byte scanning for XMP packets is not recommended; it is slow and unreliable, and should be used only if absolutely necessary. This section explains how files can, if necessary, be scanned for XMP packets, and why this should be done with extreme caution.

Without knowledge of the file format, simply locating packets is not 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):

```
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 and 32-bit encodings, a scanner cannot be sure whether the `0x00` values are in the high- or low-order portion 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 that individual attribute values in the processing instruction can have either single or double quotes. The following header is well-formed:

```
xpacket begin="i>>¿" 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

If the attribute has no value (that is, the value is the empty string), the encoding is UTF-8.

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

# 3 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.

The formats are listed here alphabetically within general categories:

---

<a href="#">"Image formats" on page 15</a>	<a href="#">DNG (Digital Negative)</a> <a href="#">GIF (Graphic Interchange Format)</a> <a href="#">JPEG</a> <a href="#">JPEG 2000</a> <a href="#">PNG (Portable Network Graphics)</a> <a href="#">TIFF (Tagged Image File Format)</a>
<a href="#">"Dynamic media formats" on page 26</a>	<a href="#">ASF (WMA, WMV)</a> <a href="#">AVI</a> <a href="#">FLV (Flash Video)</a> <a href="#">MOV (QuickTime)</a> <a href="#">MP3</a> <a href="#">MPEG-2</a> <a href="#">MPEG-4</a> <a href="#">SWF (Flash)</a> <a href="#">WAV</a>
<a href="#">"Video package formats" on page 33</a>	<a href="#">AVCHD</a> <a href="#">P2</a> <a href="#">Sony HDV (High Definition Video)</a> <a href="#">XDCAM EX</a> <a href="#">XDCAM FAM</a> <a href="#">XDCAM SAM</a>
<a href="#">"Adobe application formats" on page 37</a>	<a href="#">AI (Adobe Illustrator)</a> <a href="#">INDD, INDT (Adobe InDesign)</a> <a href="#">PSD (Adobe Photoshop)</a>
<a href="#">"Markup formats" on page 44</a>	<a href="#">HTML</a> <a href="#">XML</a>
<a href="#">"Document formats" on page 45</a>	<a href="#">PDF</a> <a href="#">PS, EPS (PostScript and Encapsulated PostScript)</a> <a href="#">UCF (Universal Container Format)</a>

---

## Image formats

- [DNG \(Digital Negative\)](#)
- [GIF \(Graphic Interchange Format\)](#)
- [JPEG](#)

- [JPEG 2000](#)
- [PNG \(Portable Network Graphics\)](#)
- [TIFF \(Tagged Image File Format\)](#)

## DNG (Digital Negative)

DNG is a publicly documented, standardized format created by Adobe for storage of raw image data as captured by digital cameras. It was created as an alternative to the more than 400 wildly varying formats all called "camera raw;" see ["Camera raw formats" on page 16](#).

DNG files can embed XMP metadata; they are, in fact, well-behaved TIFF. They can be processed by the Adobe DNG SDK, by the Adobe Camera Raw (ACR) SDK, or by the TIFF handler in the XMP Toolkit. The primary difference between the Adobe DNG/ACR handlers and the XMP Toolkit TIFF handler is that the TIFF handler does not interpret the Exif `MakerNote` tag.

Metadata can be embedded in DNG in the following ways:

- Using TIFF or Exif metadata tags
- Using the IPTC metadata tag (33723)
- Using the XMP metadata tag (700)

Note that TIFF and Exif use nearly the same metadata tag set, but TIFF stores the tags in IFD 0, while Exif store the tags in a separate IFD. Either location is allowed by DNG, but the Exif location is preferred. See ["TIFF and Exif tags for metadata" on page 73](#).

**Reference** For further information on this file format, see:

<http://www.adobe.com/products/dng/>

## Camera raw formats

Many raw formats look like TIFF, but do not behave as expected in various ways. Do not attempt to process camera raw files with generic TIFF software. Similarly, you should not pass a raw-format file directly to XMP Toolkit file handlers (XMPFiles), but should use the Adobe Camera Raw (ACR) SDK instead.

When writing your own metadata handler for camera raw files, it can be difficult to distinguish a possible camera raw file from a generic TIFF file. A pragmatic partial solution is to filter by file extension. Known camera raw extensions include:

```
.ARW   .CR2   .CRW   .DCR   .ERF   .FFF
.MEF   .MFW   .MOS   .MRW   .NEF   .ORF
.PEF   .RAW   .SR2   .SRF   .STI   .X3F
```

## GIF (Graphic Interchange Format)

In a GIF 89a 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.

**NOTE:** The “Application Extension” mechanism used for XMP was added in the 89a revision of GIF; the GIF 87a standard does not support it. XMP should not be placed into GIF 87a files.

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.

The following figure shows how XMP is embedded in the 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, must be encoded as UTF-8	Byte
	0x01	“Magic trailer”	258 Bytes
	0xFF		
	0xFE		
	⋮		
	0x01		
	0x00		
	0x00	Block Terminator	Byte

**Reference** For reference information, see the GIF 89a specification at:

<http://www.w3.org/Graphics/GIF/spec-gif89a.txt>

## JPEG

The JPEG (Joint Photographic Experts Group) specification concerns itself almost entirely with the image compression algorithm, and has very little to say about the remainder of the file format. It specifies a sequence of 2-byte markers, interspersed among data. Each marker defines the interpretation of data that follows it. According to the JPEG standard, any number of marker segments may appear in any order; the JFIF and Exif standards built upon JPEG define some ordering restrictions.

The first byte of each marker is 0xFF, the second byte is a type identifier the range 0x01..0xFE; the marker is followed by data which extends to the next marker. The pairs 0xFF00 and 0xFFFF are not markers. A marker can be preceded by any number of 0xFF fill bytes. Restart markers can appear within the compressed image stream: any 0xFF byte in the compressed image has a 0x00 byte inserted after it so that it does not appear to be a marker.

The marker types FFE0-FFE7 are generally used for application data, named APP*n*. By convention, an APP*n* marker begins with a string identifying the usage, called a namespace or signature string. An APP1 marker identifies Exif and TIFF metadata; an APP13 marker designates a Photoshop Image Resource (PSIR) that contains IPTC metadata; another APP1 marker designates the location of the XMP packet.

Markers
<i>(other markers)</i>
APP1: Exif/TIFF
APP1: XMP
APP13: PSIR/IPTC
<i>(other markers)</i>
Image data

Both JFIF and Exif specify a particular APP*n* marker segment as immediately following the SOI marker. Neither of the JFIF and Exif specifications references the other, so there is no declared standard for mixing them. Readers should be prepared to encounter files that contain an Exif APP1 marker segment following the JFIF and JFXX APP0 marker segments.

After the type, the marker contains a length value and the identifying namespace string. The length value is 2 (the length field itself) plus the length of the namespace field, plus the length of the data in bytes. Metadata markers share the convention of having NULL-terminated namespace strings:

Marker	Signature, including NULLs	Usage
APP1	"Exif\0\0" (2 NULLs)	TIFF and Exif metadata
APP1	"http://ns.adobe.com/xap/1.0\0"	XMP
APP13	"Photoshop 3.0\0"	Photoshop image resources, including IPTC metadata, but not including XMP or Exif metadata

The following table shows the entry format for the XMP section:

Byte offset, length	Field value	Field name	Comments
0, 2 bytes	0xFFE1	APP1	APP1 marker identifies metadata section.
2, 2 bytes	2 + 29 + length of XMP packet	lp	Size in bytes of this count plus the following two portions.
4, 29 bytes	Null-terminated ASCII string without quotation marks	namespace	XMP namespace URI, used as unique ID:  <code>http://ns.adobe.com/xap/1.0/</code>
33, < 65503	XMP packet		Must be encoded as UTF-8.

The JPEG standard does not prescribe ordering among APP $n$  segments, but some related standards do. For example, JIFF requires that JFIF and JFXX APP0 segments be immediately after the SOI. Also, some applications improperly assume that the segments are in a particular order. For compatibility, it is best to put any JFIF APP0 first, the Exif APP1 next, the XMP APP1 next, the PSIR APP13 next, followed by all other marker segments.

JPEG is inherently a sequential file structure; however, nothing prevents the content of an APP $n$  marker segment from having its own internal formatting. Exif, for example, embeds the linked TIFF data structure as the content of an APP1 marker segment.

## Extended XMP in JPEG

Following the normal rules for JPEG sections, the header plus the following data can be at most 65535 bytes long. If the XMP packet is not split across multiple APP1 sections, the size of the XMP packet can be at most 65502 bytes. It is unusual for XMP to exceed this size; typically, it is around 2K.

If the serialized XMP packet becomes larger than the 64K limit, you can divide it into a main portion (*StandardXMP*) and an extended portion (*ExtendedXMP*), and store it in multiple JPEG marker segment. A reader must check for the existence of ExtendedXMP, and if it is present, integrate the data with the main XMP. Each portion (standard and extended) is a fully formed XMP metadata tree, although only the standard portion contains a complete packet wrapper. If the data is more than twice the 64K limit, the extended portion can also be split and stored in multiple marker segments; in this case, the split portions are not fully formed metadata trees.

When ExtendedXMP is required, the metadata must be split according to some algorithm that assigns more important data to the main portion, and less important data to the extended portions or portions. The definition of importance is up to the application; see [“Partitioning XMP” on page 20](#).

The main portion of the metadata tree must be serialized and written as the standard XMP packet, in the APP1 marker segment described above, called StandardXMP. The extended portion must be serialized without a packet wrapper, and written as a series of APP1 marker segments, collectively called the ExtendedXMP.

When written into the JPEG file, the serialized text for the ExtendedXMP can be further split as necessary into a series of roughly 65400 byte chunks. This number is not fixed; readers must tolerate other sizes. The upper limit on the ExtendedXMP chunk size is 65458 (65535 less the additional description bytes, 2+35+32+4+4). This second 64K split is a simple separation of the XML text into data chunks, without regard to XML tokens or even UTF-8 characters.

Each chunk is written into the JPEG file within a separate APP1 marker segment. Each ExtendedXMP marker segment contains:

- A null-terminated signature string of "http://ns.adobe.com/xmp/extension/".
- A 128-bit GUID stored as a 32-byte ASCII hex string, capital A-F, no null termination. The GUID is a 128-bit MD5 digest of the full ExtendedXMP serialization.
- The full length of the ExtendedXMP serialization as a 32-bit unsigned integer
- The offset of this portion as a 32-bit unsigned integer.
- The portion of the ExtendedXMP

The GUID is also stored in the StandardXMP as the value of the `xmpNote:HasExtendedXMP` property. This allows detection of mismatched or modified ExtendedXMP. A reader must only incorporate ExtendedXMP blocks whose GUID matches the value of `xmpNote:HasExtendedXMP`. The URI for the `xmpNote:` namespace is "http://ns.adobe.com/xmp/note/".

When partitioning the XMP and determining the remaining size of the StandardXMP, be sure to first add the `xmpNote:HasExtendedXMP` property to the StandardXMP with an initial 32-byte dummy value. This ensures accurate values for the size of the StandardXMP serialization, which must contain the `xmpNote:HasExtendedXMP` property.

A JPEG writer should write the ExtendedXMP marker segments in order, immediately following the StandardXMP. However, the JPEG standard does not require preservation of marker segment order. A robust JPEG reader should tolerate the marker segments in any order.

The offset field of the marker segment is the offset of this chunk of the ExtendedXMP serialization within the full ExtendedXMP serialization. The first chunk has offset 0, the second chunk has an offset equal to the first chunk's size, and so on. The offsets allow proper reconstruction of the ExtendedXMP serialization even if the APP1 marker segments are reordered.

A JPEG reader must recompose the StandardXMP and ExtendedXMP into a single data model tree containing all of the XMP for the JPEG file, and remove the `xmpNote:HasExtendedXMP` property.

## Partitioning XMP

The specification does not mandate any particular procedure for separating XMP into standard and extended portions. However, it is a good idea for certain things to be in the first packet, so that older handlers and implementations that recognize only the first packet will still work. Adobe products follow these recommendations and guidelines in order to ensure that the most important data fields go into the first packet:

- In order to avoid partitioning when possible, the StandardXMP serialization should be as small as possible, making full use of RDF shorthand and eliminating all formatting whitespace. If the StandardXMP is too large, try to reduce or eliminate the packet-padding whitespace.
- If it the StandardXMP is still too large, delete any existing `xmp:Thumbnails` property. (Applications should avoid use of the `xmp:Thumbnails` property in JPEG files. The XMP form of the thumbnail is relatively large, and the Exif form of JPEG has a standard native thumbnail that is already produced by most or all digital cameras.)
- If the StandardXMP packet is still too large, move these items to the extension portion, in order, until the remaining StandardXMP is small enough:

- All properties in the Camera Raw namespace.
- The `photoshop:History` property.
- Other top-level properties in order of estimated serialized size, largest first.

**Reference** For reference information, see:

- JPEG File Interchange Format (JFIF) Version 1.02.
- The JPEG specification on the W3C JPEG website:  
<http://www.w3.org/Graphics/JPEG/itu-t81.pdf>
- 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, APPn markers, SPIFF compression types and Registration Authorities (REGAUT).

This specifies the format of APPn markers and the file interchange format.

**NOTE:** TIFF, JPEG, and PSD share complex issues of how native metadata formats are stored; see [“Native metadata in digital photography formats” on page 62](#).

## JPEG 2000

JPEG 2000 uses the ISO Base Media File Format. This is the same basic format as MPEG-4, although some internal details differ. The JPEG 2000 flavor of the file format is ISO/IEC 15444-12, the MPEG-4 flavor is ISO/IEC 14496-12. The ISO Base Media File Format is a "chunky" file format, similar to QuickTime or RIFF (AVI and WAV). In this case the chunks are called *boxes*.

Each box consists of a header followed by data. The structure of the header is:

Offset, length	Description
0, 4 bytes	32-bit, unsigned, big-endian value, the size of the box. The size includes both the header and data portions of the box. <ul style="list-style-type: none"> <li>➤ If the size does not fit into 32 bits, this value is 1, and the size value is in the extended size field.</li> <li>➤ Value is 0 if the box extends to the end of the file; in this case, no extended size value is present.</li> </ul>
4, 4 bytes	32-bit, unsigned, big-endian value, the type code. Typically defined as 4 ASCII characters, shown in file order. The standard extension mechanism is the UUID box. The data portion of a UUID box must begin with a 16-byte unique ID ("uuid").
8, 8 bytes	64-bit, unsigned, big-endian value, the extended size (if needed).

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		Must be encoded as UTF-8

**Reference** For reference information, see the JPEG 2000 standard:

<http://www.jpeg.org/JPEG2000.html>

## PNG (Portable Network Graphics)

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. 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.

The PNG data format is shown in the following table.

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 <code>iTXt</code> 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

Field	Length	Comments
Null separator	1	value = 0x00
Translated keyword	0	Not used for XMP metadata
Null separator	1	value = 0x00
Text	length of packet	The XMP packet, must be encoded as UTF-8
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** For reference information, see:

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

## TIFF (Tagged Image File Format)

Tagged Image File Format (abbreviated TIFF or TIF) is a file format for storing images, including photographs and line art, and also a metadata format. A number of other formats, including DNG for camera raw data and Exif for metadata, are also well-behaved TIFF. Many camera raw formats, however, look like TIFF but are not well-behaved and cannot be processed by a TIFF handler.

The overall structure of a TIFF file is relatively simple, an 8 byte header and a chain of Image File Directories (IFDs). The file header contains:

Offset, length	Description
0, 2 bytes	Byte order, "II" (0x4949) for little-endian, "MM" (0x4D4D) for big-endian
2, 2 bytes	Identifies file as TIFF with the number 42 in the given byte order: 0x2A00 if little-endian, 0x002A if big-endian
4, 4bytes	<p>Offset of the first (0th) IFD. Each IFD begins with a 2-byte count of directory entries, followed by a sequence of 12-byte directory entries, followed by a 4 byte offset of the next IFD. The last IFD must have zero as the offset of the next IFD.</p> <p>The IFDs can be anywhere in the file (after the 8 byte header), in any order. Each IFD must be on a 2 byte boundary; that is, it must have an even offset.</p> <p>Within an IFD, entries must be sorted in ascending tag order.</p> <p>A TIFF file must have at least one IFD and each IFD must have at least one entry. All offsets are absolute (from the beginning of the file header), not relative offsets from some other point within the file.</p>

Each IFD entry starts with a 2-byte identifier, or *tag*. (The term *TIFF field* has been used to refer to an IFD entry or an IFD entry plus the associated value. The term *TIFF tag* has been used to refer to just the numeric ID or as a synonym for TIFF field. Here, tag is used for the numeric ID.) The entry then gives the data type for the value, then either the value itself or a pointer to it.

Each 12-byte IFD entry contains:

Offset, length	Description
0, 2 bytes	The TIFF tag, a numeric identifier. Tag identifiers for individual properties are listed in the TIFF and Exif specifications, and in <a href="#">Chapter 5, "Digital Photography Native Metadata."</a>
2, 2 bytes	The value data type. One of: <ol style="list-style-type: none"> <li>1, BYTE, An 8-bit unsigned integer</li> <li>2, ASCII, An 8-bit byte with a 7-bit ASCII character</li> <li>3, SHORT, A 16-bit unsigned integer</li> <li>4, LONG, A 32-bit unsigned integer</li> <li>5, RATIONAL, A pair of LONGs, numerator then denominator</li> <li>6, SBYTE, An 8-bit signed integer</li> <li>7, UNDEFINED, An undefined 8-bit byte</li> <li>8, SSHORT, A 16-bit signed integer</li> <li>9, SLONG, A 32-bit signed integer</li> <li>10, SRATIONAL, A pair of SLONGs, numerator then denominator</li> <li>11, FLOAT, A 4-byte IEEE floating point value</li> <li>12, DOUBLE, An 8-byte IEEE floating point value</li> </ol>
4, 4 bytes	The length, which is the number of values; depending on the data type, it is not necessarily the number of bytes. For the ASCII type, it is the number of characters. This is the exact number; it does not include any padding for odd lengths.  An ASCII value must have a terminating NULL (0x00) character, which is included in the count. An ASCII value can have multiple NULL terminated strings; the count is the total for all of the strings. Individual strings, other than the first, might begin on odd offsets.
8, 4 bytes	The value itself or a pointer to it: <ul style="list-style-type: none"> <li>➤ Small values (4 bytes or less) must be placed directly in the IFD entry. If less than 4 bytes, the value is placed in the lower numbered bytes.</li> <li>➤ For larger values, this is the byte offset that points to the data block. The offset must be even, but may otherwise point anywhere in the file regardless of IFD or IFD entry order.</li> </ul>

Each directory entry represents either an individual metadata property, or a pointer to a block of metadata in another format. For tags that identify blocks of metadata in other formats, including XMP, the entry provides a length/offset pair that points to the data block. These tags are:

Tag	Hex	Usage
700	0x2BC	XMP packet
33723	0x83BB	IPTC dataset
34377	0x8649	Photoshop Image Resources (PSIR) containing non-metadata resources, and possibly duplicating the IPTC metadata

34665	0x8769	Exif subsidiary IFD offset
34853	0x8825	GPS subsidiary IFD offset

The IFD entry for XMP looks like this:

Byte offset	Field value	Field name	Comments
0	700	TAG	Tag that identifies the field (decimal value).
2	1	Field type	The field type is <code>BYTE</code> , which is represented as a value of 1.
4		Count	The total byte count of the XMP packet.
8		Value or Offset	The byte offset of the XMP packet, which must be encoded as UTF-8.

**NOTE:** TIFF, JPEG, and PSD share complex issues of how native metadata formats are stored; see [“Native metadata in digital photography formats” on page 62](#).

**Reference** Official documentation for the TIFF file format is available from:

<http://partners.adobe.com/public/developer/tiff/index.html>

For reference information, see the TIFF 6.0 Specification:

<http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf>

## Dynamic media formats

Certain video formats have special considerations and are collected separately; see [“Video package formats” on page 33](#).

- [ASF \(WMA, WMV\)](#)
- [AVI](#)
- [FLV \(Flash Video\)](#)
- [MOV \(QuickTime\)](#)
- [MP3](#)
- [MPEG-2](#)
- [MPEG-4](#)
- [SWF \(Flash\)](#)
- [WAV](#)

### ASF (WMA, WMV)

Advanced Systems Format (formerly Advanced Streaming Format) is Microsoft's proprietary digital audio/digital video container format, especially meant for streaming media. ASF is part of the Windows Media framework. The most common filetypes contained within an ASF file are Windows Media Audio (WMA) and Windows Media Video (WMV). WMA and WMV are very similar.

The file extension of an ASF file indicates what kind of compression is used for the content:

- An ASF file that contains audio content compressed with the WMA `codec` typically uses the `.wma` extension.
- An ASF file that contains audio content, video content, or both, compressed with WMA and WMV `codecs` uses the `.wmv` extension.
- Content that is compressed with any other `codec` use the generic `.asf` extension.

**NOTE:** Software developers must carefully read the licensing terms in the ASF specification. Microsoft grants royalty-free permission to distribute executable and object code products that implement ASF support, but explicitly forbids distribution of source code.

**Reference** The ASF specification is available at:

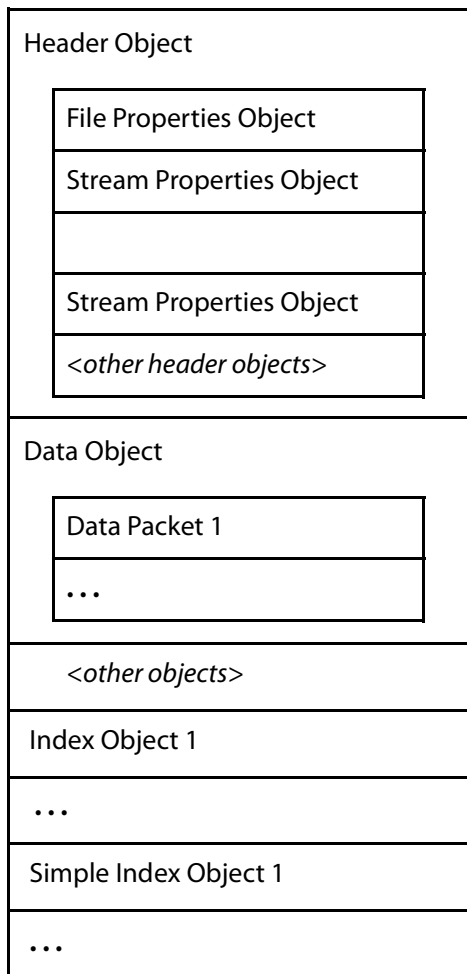
<http://www.microsoft.com/windows/windowsmedia/forpros/format/asfspec.aspx>

### ASF format

ASF files are logically composed of three types of top-level objects: Header, Data and Index Objects. A top-level object can contain other objects in its data section.

- The Header Object is mandatory and must be at the beginning of every ASF file.
- The Data Object is also mandatory and must immediately follow the Header Object.

- The Index Objects are optional, when present they must be the last objects in the ASF file.
- Other objects can appear between the Data Object and the first Index Object. XMP is embedded as one of these.



All ASF objects have a similar structure:

- A 16-byte GUID

The GUID identifies the purpose of an object. It is unique as a type identifier, not as an instance identifier; that is, the same conceptual object in different files has the same GUID. The ASF specification contains tables of standard GUIDs, and applications can create others.

The ASF specification does not define GUIDs in file byte order. The documentation lists them as though they contained:

- A 4-byte little-endian integer
- A 2-byte little-endian integer
- A 2-byte little-endian integer
- A 2-byte big-endian integer

- A 6-byte in-order sequence

Thus, if the documented value of a GUID is 00112233-4455-6677-8899-AABBCCDDEEFF, the file contains 33221100 55447766 8899AABB CCDDEEFF.

- An 8-byte little-endian size

The size includes both the GUID and size itself; that is, 24 plus the data size. The ASF specification does not say if the size is signed or unsigned. For safety it should be treated as unsigned, with a 63-bit range.

- The object's data

There appear to be no uses in ASF of relative offsets between objects, and no uses of absolute offsets of objects. Index offsets are relative to the appropriate data packet origin. This allows Header and XMP objects to be inserted, grown, or shrunk with relative ease. Only adjustments to the length of containing objects are necessary.

The ASF Header Object contains several nested objects that have native metadata fields mapped to XMP: See details of reconciliation in ["Native metadata in ASF \(WMA, WMV\)" on page 59](#).

## XMP embedded in ASF

XMP is embedded in ASF as an "Other" top-level object. The GUID in ASF notation is:

```
BE7ACFCB-97A9-42E8-9C71-999491E3AFAC
```

The GUID in file byte order is:

```
CBCF7ABE A997E842 9C719994 91E3AFAC
```

The data of the XMP object is the XMP packet, using UTF-8 encoding.

## AVI

AVI (Audio-Video Interleaved) is a multimedia container format. AVI files can contain both audio and video data in a standard container that allows synchronous audio-with-video playback.

### About RIFF

AVI and WAV are both based on the RIFF file format, but are not otherwise similar. RIFF was created in 1991 by Microsoft and IBM, and there seem to be no actively maintained file-format specifications. RIFF is a chunky format: a chunk has a 4 byte ID, a 4 byte length, and content. The ID is ASCII text, documented in file order. The length is unsigned, little-endian, and is just the content length. A zero pad byte follows the content if the length is odd.

Some RIFF chunks are containers of other chunks. The content of a container (not a formal term) is a 4 byte ID followed by a sequence of nested chunks.

A normal RIFF file as a whole has an outer container chunk with an outer ID of "RIFF" and an inner ID that defines the file type. That is, the whole file is one chunk with the ID "RIFF", and that chunk is a container whose content begins with an ID for the file type followed by the "real" file structure. For AVI the inner ID is "AVI", for WAV the inner ID is "WAVE".

Another standard container is the "LIST" chunk. This is the normal grouping mechanism, used to create a tree structure instead of a single flat sequence of top level chunks.

The XMP in AVI and WAV is in a chunk with the ID "\_PMX", encoded as UTF-8. Note that the ID is backwards, due to a bug in the initial implementation concerning processor byte order. The XMP chunk is immediately within the outermost "RIFF" chunk. There is no ordering constraint of the XMP relative to other chunks.

**NOTE:** In the Adobe AVI and WAV handlers, for historical reasons, unused chunks get an ID of "JUNQ", not the standard RIFF "JUNK".

**Reference** The Library of Congress Digital Preservation project has general RIFF information at:

<http://www.digitalpreservation.gov/formats/fdd/fdd000025.shtml>

This includes a link to an HTML version of "Multimedia Programming Interface and Data Specifications 1.0":

[http://www.tactilemedia.com/info/MCI\\_Control\\_Info.html](http://www.tactilemedia.com/info/MCI_Control_Info.html)

## About AVI

The OpenDML extensions talk about a variety of extensions for AVI files. Of particular interest to XMP are issues about files larger than 1GB.

Because of bugs in early implementations of AVI, the standard RIFF/AVI chunk should be limited to a maximum length of 1GB. The OpenDML specification tells how to go beyond this. Larger AVI files contain an initial RIFF/AVI chunk, followed by a sequence of RIFF/AVIX chunks. The extension chunks must only contain media content in LIST/movi chunks, everything else must be in the initial RIFF/AVI chunk. In particular the XMP must be in the initial RIFF/AVI chunk.

AVI files can contain native metadata. See detail of how these are reconciled with XMP in ["Native metadata in AVI" on page 60](#).

**Reference** The Library of Congress Digital Preservation project has specific AVI information at:

<http://www.digitalpreservation.gov/formats/fdd/fdd000059.shtml>

This includes a link to OpenDML extensions at:

<http://www.morgan-multimedia.com/download/odmlff2.pdf>

## FLV (Flash Video)

FLV is designed to carry synchronized audio and video streams, and is used to deliver video over the Internet using Adobe Flash Player (in versions later than 6). Flash Video content may also be embedded within SWF files.

FLV is a fairly simple format, with a strong orientation to streaming use. It consists of a small file header then a sequence of tags that can contain audio data, video data, or ActionScript data. For FLV version 1, each tag begins with an 11 byte header:

- UI8 tag type - 8 = audio tag, 9 = video tag, 18 = script data tag
- UI24 content length in bytes
- UI24 time - low order 3 bytes
- UI8 time - high order byte
- UI24 stream ID

This is followed by the tag's content, then a UI32 "back pointer" which is the header size plus the content size. A UI32 zero is placed between the file header and the first tag as a terminator for backward scans. The time in a tag header is the start of playback for that tag. The tags must be in ascending time order. For a given time it is preferred that script data tags precede audio and video tags. Each audio or video tag typically contains one frame of data.

For metadata purposes, only the script data tags are of interest. Script data information becomes accessible to ActionScript at the playback moment of the script data tag through a call to a registered data handler. The content of a script data tag contains a string and an ActionScript data value. The string is the name of the handler to be invoked, the data value is passed as an ActionScript Object parameter to the handler.

A variety of native metadata is contained in a script data tag with the name `onMetaData`. This contains only internal information such as duration or width/height, nothing that is user- or author-editable, such as title or description. Some of these native items are imported into the XMP; none are updated from the XMP.

#### Placement of XMP

XMP is embedded in FLV as a script data tag with the name `onXMPData`. It must be placed at time 0 and must have stream ID 0. It should be after any time 0 `onMetaData` tag, and before any time 0 audio or video tags.

Software looking for existing XMP must examine all time 0 tags. One cannot presume that all third party modifiers of FLV files will preserve `onMetaData`, then `onXMPData`, then audio/video ordering.

The data value for `onXMPData` is an ECMA array. Standard serialized XMP is in an array item with the key `liveXML`; the data of this item is an ActionScript string containing a normal UTF-8 XMP packet (including padding). The ActionScript string can be a short or long string as appropriate.

#### Reference

The formal specification for SWF and FLV is *Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8*. This is available at:

<http://www.adobe.com/licensing/developer>

**NOTE :** In the public FLV specification version 8 and earlier, there are errors in the descriptions of `SCRIPTDATAOBJECT`, `SCRIPTDATAOBJECTEND`, `SCRIPTDATAVARIABLE`, `SCRIPTDATASTRING`, `SCRIPTDATALONGSTRING`, `SCRIPTDATADATE`, ECMA arrays, and strict arrays.

## MOV (QuickTime)

The QuickTime MOV file format was developed by Apple as a container for a wide variety of dynamic media including sound, video, and animation. It functions as a multimedia container file that contains one or more tracks, each of which stores a particular type of data: audio, video, effects, or text (for subtitles, for example). Each track either contains a digitally-encoded media stream (using a specific `codec`) or a data reference to the media stream located in another file. Tracks are maintained in a hierarchal data structure consisting of objects called *atoms*.

The basic structure of a QuickTime file is similar to the RIFF format used by AVI and WAV. QuickTime was used as the basis of the MPEG-4 format, ISO/IEC 14496-14.

The basic unit of a QuickTime file, the atom, is similar to the "chunks" in RIFF. An atom can either be a leaf atom containing data, or a container atom containing other atoms. There is no structural indication of a leaf or container atom, this is implicit in the atom's type. Other than nesting rules, there is generally no necessary ordering among atoms.

All atoms begin with a standard or extended header. The standard header has a 32-bit big-endian unsigned length followed by a 32-bit type code. The type codes are almost always 4 ASCII characters, but a few special cases exist (such as 0x00000001). The size field gives the atom's total size, including the 8-byte header. If the initial size field is set to 1, a 64-bit big-endian size follows the standard header, forming an extended header.

XMP is stored in QuickTime as an "XMP\_" atom, within a "udta" atom, within a top level "moov" atom. The XMP atom's content is the XMP packet, using UTF-8 encoding.

**Reference** The formal specification for the QuickTime file format is available at:

<http://developer.apple.com/reference/QuickTime/idxFileFormatSpecification-date.html>

## MP3

MPEG-1 Audio Layer 3, more commonly referred to as MP3, is a popular audio encoding format. MPEG stands for Moving Picture Experts Group. The formal standard is ISO/IEC IS 11172-3, but this only covers the raw audio aspects. The metadata in MP3 files uses the ID3v2 format. When used with XMP, this must be ID3v2.3 or ID3v2.4. The ID3v2.3 and ID3v2.4 formats are almost identical. The most notable difference is that ID3v2.4 allows text values to be UTF-8, in addition to ISO 8859-1 (Latin-1) or UTF-16.

The entire ID3 portion of the MP3 file is called the ID3 "tag" (rather confusingly, given other media file and metadata terminology). The individual metadata items are called ID3 "frames".

The XMP is placed within the ID3 as a "PRIV" frame with an Owner identifier of "XMP". The content of the XMP PRIV frame is the XMP packet, encoded as UTF-8.

MP3 files can contain native metadata; see detail of reconciliation with XMP in "[Native metadata in MP3](#)" on page 60.

**Reference** Specifications can be found at:

<http://www.id3.org/id3v2.3.0>

<http://www.id3.org/id3v2.4.0-structure>

## MPEG-2

MPEG-2 is a standard for the generic coding of moving pictures and associated audio information. It describes a combination of lossy video compression and lossy audio compression (audio data compression) methods which permit storage and transmission of movies using currently available storage media and transmission bandwidth. It is not the same as MPEG-1 Audio Layer II (MP2).

MPEG-2 is a common format for standard definition digital video. It is used for DVD discs, by DV camcorders, for terrestrial (over the air) broadcast, for cable, and direct broadcast satellite. The formal specification for MPEG-2 is ISO/IEC 13818.

XMP is not directly embedded within MPEG-2 files, but is specified as a *sidecar file*. This is a separate file containing just the XMP packet, which is stored at the same location as the MPEG-2 file, and uses the same file name, with the file extension `.xmp` replacing the original file extension.

## MPEG-4

MPEG-4 is a collection of methods defining compression of audio and visual (AV) digital data. MPEG-4 absorbs many of the features of MPEG-1 and MPEG-2 and other related standards, adding new features

such as (extended) VRML support for 3D rendering, object-oriented composite files (including audio, video and VRML objects), support for externally-specified Digital Rights Management and various types of interactivity. AAC (Advanced Audio Codec) was standardized as an adjunct to MPEG-2 (as Part 7) before MPEG-4 was issued.

The file format is defined by parts 12 and 14 of ISO 14496. Part 12 defines the "ISO Base Media File Format", which happens to also be used by JPEG 2000 (ISO 15444). Part 14 describes MPEG-4 specific aspects.

### Placement of XMP

XMP is embedded in MPEG-4 files in the same manner as in JPEG 2000, using a top-level UUID box. The UUID for both is:

```
BE7ACFCB 97A942E8 9C719994 91E3AFAC
```

The remainder of the box is a typical XMP packet, encoded as UTF-8, including packet wrapper and padding.

MPEG-4 files can contain native metadata; see detail of reconciliation with XMP in ["Native metadata in MPEG-4" on page 61](#).

## SWF (Flash)

SWF is a proprietary vector graphics file format produced by the Flash software from Adobe. Intended to be small enough for publication on the web, SWF files can contain animations or applets of varying degrees of interactivity and function. SWF is also sometimes used for creating animated display graphics and menus for DVD movies, and television commercials.

SWF was designed to deliver vector graphics, text, video and sound over the Internet. The SWF file format is designed to be an efficient delivery format, not a format for exchanging graphics between graphics editors. SWF is quite different from the Flash Video file format, FLV. FLV is designed to carry synchronized audio and video streams. See

An SWF file consists of a file header followed by a sequence of tagged data blocks, called *tags*. The tags share a common format, so any program parsing a SWF file can skip over blocks it does not understand. Data inside the block can point to offsets within the block, but can never point to an offset in another block. This enables tags to be removed, inserted, or modified by tools that process a SWF file.

SWF files always store integers in little-endian byte order. The common tag structure is very simple, a short or long tag header followed by the data. The short header is a 16-bit little-endian unsigned integer. The upper 10 bits are the tag's type code, the lower 6 bits are the length of the data. If the data is 63 bytes or longer, a long tag header is used. This is a short header with a length value of 63 (0x3F), followed by a 32-bit little-endian integer giving the actual data length.

Beginning with SWF 8, a `FileAttributes` tag is required immediately after the file header.

### Placement of XMP

XMP is supported beginning with SWF 8. There are two aspects to placing XMP in SWF:

- A flag in the `FileAttributes` tag denoting whether the file contains XMP.
- A tag containing the XMP.

The `FileAttributes` tag has a type code of 69. The tag data is 4 bytes containing several variable-length bit fields. Counting bits from the low-order end, the `HasMetadata` flag is bit 4 of the first byte; that is, the mask 0x10 selects the `HasMetadata` flag from the first byte. This flag must be set if and only if the file contains XMP.

The XMP tag (called the `Metadata` tag) has type code 77. The tag data is the serialized XMP, using UTF-8 encoding. It is best to omit the XMP packet wrapper (including padding) to save space. Similarly, you should use RDF shorthand and omit formatting whitespace.

**Reference** The formal specification for SWF and FLV is "Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8." This is available at:

<http://www.adobe.com/licensing/developer>

**NOTE:** The SWF specification (not this XMP Specification) is the authority for how XMP is embedded in SWF.

## WAV

WAV (or WAVE), short for Waveform audio format, is a Microsoft and IBM audio file format standard for storing audio on PCs. It is a variant of the RIFF bitstream format method for storing data in "chunks", and thus also close to the IFF (Amiga) and the AIFF (Mac OS) formats. It is the main format used on Windows systems for raw audio. See "[About RIFF](#)" on page 28 for details.

The WAV format is covered by the base RIFF specification ("Multimedia Programming Interface and Data Specifications 1.0") and the Exif 2.2 specification, with this exception: WAV files can have a top level DISP chunk, for which there appears to be no public specification. The DISP chunk's content is a 32-bit little-endian integer type code, followed by a null terminated string. The only recognized type code is 1. The top-level DISP chunk is reconciled with `dc:title["x-default"]`.

WAV files can contain native metadata; see details of reconciliation with XMP in "[Native metadata in WAV](#)" on page 62.

**Reference** The Library of Congress Digital Preservation project has specific WAV information at:

<http://www.digitalpreservation.gov/formats/fdd/fdd000001.shtml>

## Video package formats

Certain video formats have special considerations. In these formats a video entity ( a movie) consists of a collection, or package, of related files of various types—audio, video, voice, and so on. In video package formats, the document unit is called a *clip*. Information for an individual clip is stored in multiple files, in a directory structure defined by the format specification. XMP metadata relates to the clip as a whole, rather than to individual files, and is stored in its own file for the associated clip.

These folder-oriented formats use shallow trees with specific folder names and highly stylized file names. The user thinks of the tree as a collection of clips, consisting of multiple files for video, audio, metadata, and so on. For example, a portion of a P2 folder might look like this:

```
.../MyMovie
  CONTENTS
    CLIP
      0001AB.XML
      0002CD.XML
    VIDEO
      0001AB.MXF
      0002CD.MXF
    VOICE
      0001AB.WAV
      0002CD.WAV
```

The user thinks of `./MyMovie` as the container of P2 video, which in this case contains two clips identified with a file base name of `0001AB` and `0002CD`. Each clip is stored as a collection of files, each file holding some specific aspect of the clip's data. The exact folder structure and file layout differs, but the basic concepts carry across all of the folder-oriented video-package formats.

**NOTE:** The folder layout descriptions here are meant to show typical usage and describe where XMP files are found; they are not meant to be complete or definitive package format descriptions.

- [AVCHD](#)
- [P2](#)
- [Sony HDV \(High Definition Video\)](#)
- [XDCAM EX](#)
- [XDCAM FAM](#)
- [XDCAM SAM](#)

## AVCHD

AVCHD (Advanced Video Codec High Definition) is a high-definition video recording format for use in digital tapeless camcorders. The format is comparable to other handheld video camera recording formats, particularly TOD, HDV and MiniDV.

A typical AVCHD package layout looks like this:

```
BDMV/
  index.bdmv
  MovieObject.bdmv
  PLAYLIST/
    00000.mpls
    00001.mpls
  STREAM/
    00000.m2ts
    00001.m2ts
    00001.xmp
  CLIPINF/
    00000.clpi
    00001.clpi
  BACKUP/
```

A base name with a numeric sequence distinguishes files belonging to the same clip. The XMP is placed in the `STREAM` folder, using the clip base name and the extension `XMP`.

In this example, the files `00001.mpls`, `00001.m2ts`, and `00001.clpi` all belong to clip 1. The related metadata is in `00001.xmp`.

## P2

A P2 card is a solid-state memory device that plugs into the PCMCIA slot of a Panasonic P2 video camera, such as the AG-HVX200. The digital video and audio data from the video camera is recorded onto the card in a structured, codec-independent format known as MXF (Media eXchange Format). A clip is said to be in the P2 format if its audio and video are contained in Panasonic Op-Atom MXF files, and these files are located in a specific file structure.

The root of the P2 file structure is a `CONTENTS` folder. Each *essence* item (an item of video or audio) is contained in a separate MXF wrapper file; the video MXF files are in the `VIDEO` subfolder, and do on. The relationships between essence files and the metadata associated with them are tracked by XML files in the `CLIP` subfolder.

A typical P2 layout looks like this:

```
.../MyMovie
  CONTENTS/
    CLIP/
      0001AB.XML
      0001AB.XMP
      0002CD.XML
      0002CD.XMP
    VIDEO/
      0001AB.MXF
      0002CD.MXF
    AUDIO/
      0001AB00.MXF
      0001AB01.MXF
      0002CD00.MXF
      0002CD01.MXF
    ICON/
      0001AB.BMP
      0002CD.BMP
    VOICE/
      0001AB.WAV
      0002CD.WAV
    PROXY/
      0001AB.MP4
      0002CD.MP4
```

This shows two clips whose file base names are `0001AB` and `0002CD`.

A base name with a numeric sequence distinguishes files belonging to the same clip. The XMP is placed in the `CLIP` folder, beside the `.XML` file that defines the existence of the clip. The `.XML` file contains a variety of information about the clip, including some native metadata.

The XMP is stored using the clip base name and the extension XMP. In the example, the file `0001AB.XMP` contains the metadata associated with the clip defined by `0001AB.XML`.

## Sony HDV (High Definition Video)

A typical Sony HDV layout looks like this:

```
.../MyMovie/
  VIDEO/
    HDV/
      00_0001_2007-08-06_165555.IDX
      00_0001_2007-08-06_165555.M2T
      00_0001_2007-08-06_165555.XMP
      00_0001_2007-08-06_171740.IDX
      00_0001_2007-08-06_171740.M2T
      00_0001_2007-08-06_171740.XMP
      tracks.dat
```

This shows two clips with the same base name, but different time stamps. The `.IDX` file defines the existence of the clip. XMP is stored in a file with the clip base name, the date/time suffix from the associated `.IDX` file, and the `.XMP` extension.

## XDCAM EX

This package format is used by certain models of the Sony XDCAM line of high-definition disc video camcorders. See also [“XDCAM FAM” on page 36](#) and [“XDCAM SAM” on page 37](#).

**NOTE :** Sony documentation uses mixed-case folder names "General", "Clip", "Sub", and "Edit". The names are shown in all caps here.

A typical XDCAM EX package layout looks like this:

```

.../MyMovie/
  BPAV/
    MEDIAPRO.XML
    MEDIAPRO.BUP
    CLPR/
      709_001_01/
        709_001_01.SMI
        709_001_01.MP4
        709_001_01M01.XML
        709_001_01M01.XMP
        709_001_01R01.BIM
        709_001_01I01.PPN
      709_001_02/
      709_002_01/
      709_003_01/

```

The `CLPR` (clip root) folder contains clip subfolders, whose name is the base file name for that clip. Each clip folder contains a media file (`.MP4`), a clip information file (`.SMI`), a real-time native metadata file (`.BIM`), a non-real-time native metadata file (`.XML`), and a picture pointer file (`.PPN`).

XMP metadata is stored in the `.XMP` file within a clip directory, using the same base name as the XML file.

## XDCAM FAM

This package format is used by certain models of the Sony XDCAM line of high-definition disc video camcorders. See also [“XDCAM EX” on page 36](#) and [“XDCAM SAM” on page 37](#).

A typical FAM layout looks like this:

```

.../MyMovie/
  INDEX.XML
  DISCMETA.XML
  MEDIAPRO.XML
  CLIP/
    C0001.MXF
    C0001M01.XML
    C0001M01.XMP
    C0002.MXF
    C0002M01.XML
    C0002M01.XMP

```

The top-level folder `MyMovie/` contains XDCAM data for two clips whose raw names are `C0001` and `C0002`. The `CLIP` folder contains at least one `.XML` file that defines the existence of a clip with that base file name..

A clip file and its related XMP are kept together in the `CLIP` folder.

- The `.XML` file defines the existence of the clip. The name uses the base clip name plus additional identifying characters. It contains a variety of information about the clip, including some native metadata.
- The `.XMP` file with the same base file name as the XML file contains the XMP for the clip.

## XDCAM SAM

This package format is used by certain models of the Sony XDCAM line of high-definition disc video camcorders. See also [“XDCAM EX” on page 36](#) and [“XDCAM FAM” on page 36](#).

A typical SAM layout looks like this:

```

.../MyMovie/
  PROAV/
    INDEX.XML
    DISCMETA.XML
    DISCINFO.XML
    CLPR/
      C0001/
        C0001C01.SMI
        C0001V01.MXF
        C0001A01.MXF
        C0001A02.MXF
        C0001R01.BIM
        C0001I01.PPN
        C0001M01.XML
        C0001M01.XMP
        C0001S01.MXF
      C0002/
        ...

```

The `CLPR/` folder contains a subfolder for each clip, which use the base file name. This example shows two clips, with base names `C0001` and `C0002`.

A clip file and its related XMP are kept together in the `CLPR/<clip>` folder.

- The `.XML` file defines the existence of the clip. The name uses the base clip name plus additional identifying characters. It contains a variety of information about the clip, including some native metadata.
- The `.XMP` file with the same base file name as the XML file contains the XMP for the clip.

## Adobe application formats

- [“AI \(Adobe Illustrator\)” on page 38](#)
- [“INDD, INDT \(Adobe InDesign\)” on page 38](#)
- [“PSD \(Adobe Photoshop\)” on page 42](#)

## AI (Adobe Illustrator)

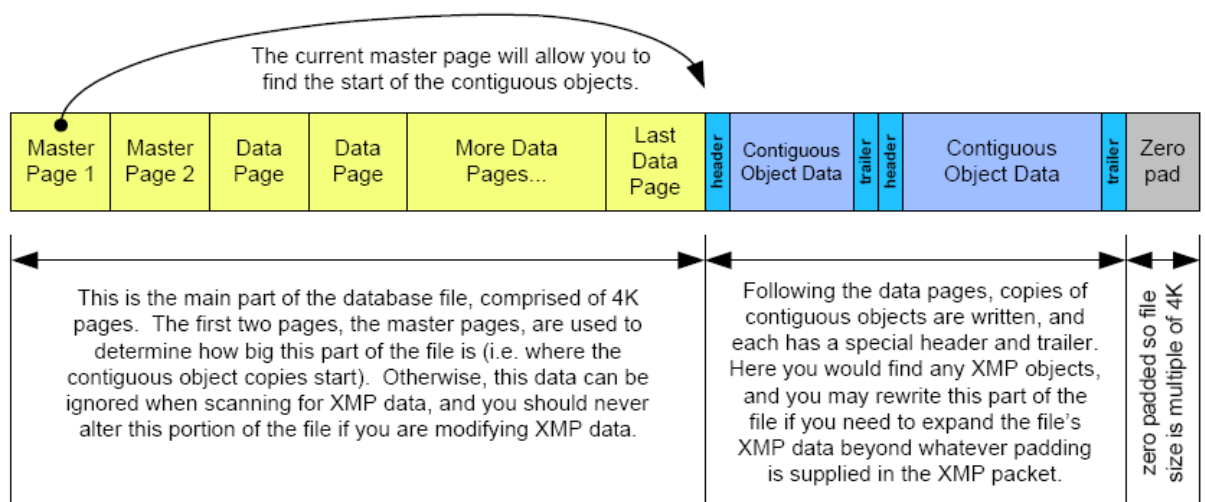
An `.ai` file generated by Adobe Illustrator® is in the Portable Document Format (PDF). Hence, the format for embedding XMP metadata is the same as for PDF files.

## INDD, INDT (Adobe InDesign)

InDesign document files (`.indd`) and InDesign template files (`.indt`) are primarily paged database files. They have 4KB pages with a leading pair of master pages to allow single write commits. The detailed structure of the database is proprietary.

Externally editable XMP is supported for database files from InDesign 2.0 and later through a "contiguous object" section at the end of the file. It is possible to modify, even extend, the XMP in the contiguous object section. It is not possible to add XMP to a database that has none without intimate knowledge of the database.

In brief, the active master page tells how many database pages the file contains. The contiguous object section begins after the last database page and extends to the end of the file. It contains copies of database objects, stored in a contiguous (non-paged) manner.



The majority of an InDesign database file consists of the 4K pages that comprise the transacted database storage. You can use the first two pages in the file, called the Master Pages, to determine whether contiguous objects are present, where they are, and whether they include any XMP metadata objects that contain XMP packets. InDesign document databases do typically contain XMP metadata, but this is not strictly required. Other types of InDesign files, such as book files do not currently contain metadata.

- For details of the Master Page structure, see ["Master page structure" on page 40](#).
- For details of the Header and Trailer that enclose each contiguous object, see ["Header and trailer structure" on page 41](#)

### Finding XMP in an InDesign file

To efficiently find XMP data when present in an InDesign database file:

1. Verify the first 16 bytes of the file are the GUID for an InDesign database file.
2. Read the two Master Page sequence numbers, and decide which master page is the current one, having the larger sequence number.
3. Using `fFilePages` from the current master page, seek to absolute file position `fFilePages * 4096`. Remember that a database file may exceed 4 gigabytes in size.
4. Search through the contiguous objects for an XMP packet. When the contiguous object you are examining is the persisted representation of an object that contains an XMP packet, the stream between the headers consists of a 4-byte integer specifying the length for the XMP packet, followed by the XMP packet itself.
  - > The byte-order of the length integer is governed by the `fObjectStreamEndian` field from the master page. This is true *only* of this value; all integer values within the data are in little-endian order.
  - > The packet header begins like this:

```
<?xpacket begin="W5M0MpCehiHzreSzNTczkc9d"
```

The quote characters can be double (") or single (').

## Rewriting XMP data beyond the packet padding

The length value that precedes the packet header must equal `fStreamLength` (from the header and trailer) minus 4 bytes (for the length value itself). You should verify that this is true before attempting to rewrite the XMP packet, and thus rewrite the length value. If the first four-byte integer in the stream (interpreted using the Master Page-specified byte order) is not `fStreamLength-4`, you should not enlarge the packet but simply use the padding if possible.

If you rewrite a contiguous object that contains an XMP packet, follow these rules:

1. While looking for a contiguous object header beyond the paged part of the file (as directed by `fFilePages` in the master page), if you find something which is not a header GUID but not zero padding at the end of the file, assume the file format has changed and fall back to default behavior.
2. InDesign might store other contiguous objects besides XMP packets at the end of the file in the same way. The API that allows this is public, so a third-party developer might also store arbitrary objects there. If you rewrite an XMP packet, make sure you are changing only the object that contains your XMP packet, and preserve any other contiguous object copies you find at the end of the file.
3. Verify that the XMP object you want to rewrite is marked externally writable in the contiguous object header (or trailer).
4. Verify that the first four-byte integer (variable endian) in the XMP object stream is indeed the length of the XMP packet immediately following, and that this value is the header's (or trailer's) `fStreamLength - 4`.
5. When you rewrite the XMP object, rewrite the header and trailer for the XMP object with an `fChecksum` field of `0xffffffff` and an updated `fStreamLength` field.
6. The size in bytes of an InDesign database file is required to be a multiple of 4K. A file that violates this constraint is considered corrupt. If you are rewriting contiguous objects at the end of the file to expand an XMP packet, pad the file with zeros if necessary after the last contiguous object trailer to ensure this condition is met.

## Master page structure

The following code shows the parts of the Master Pages of interest with respect to metadata:

```
struct MasterPage {
    // 16 byte GUID identifying this as an InDesign database
    // Must be: 0606EDF5-D81D-46e5-BD31-EFE7FE74B71D
    char fGUID [16];

    // 8 bytes; type of database (for example "DOCUMENT")
    char fMagicBytes [8];

    // Endian of object streams, 1=little endian, 2=big endian
    char fObjectStreamEndian;

    // Irrelevant stuff
    char fIrrelevant1 [239];

    // Master page sequence number. The master page with
    // the larger value is the current master page
    LittleEndianUnsignedInt64 fSequenceNumber;

    // More irrelevant stuff
    char fIrrelevant2 [8];

    // The number of pages in the file. fFilePages * 4096
    // is the absolute file offset in bytes of where any
    // contiguous data storage would begin (provided this
    // master page is the current master page).
    LittleEndianUnsignedInt32 fFilePages;

    // More irrelevant stuff
    char fIrrelevant3 [3812];
};
```

---

fGUID

The first 16 bytes of each master page (and thus the first 16 bytes of the file) are set to a GUID that identifies the file as an InDesign database.

The ID value of the master page is:

```
0606EDF5-D81D-46e5-BD31-EFE7FE74B71D
```

You must check for this value. If it is not present, something has changed in an incompatible way in the file format and you can make no further assumptions. You can the entire file to find XMP metadata, but you cannot add additional metadata beyond any padding provided in the packet.

---

fMagicBytes

Contains an 8 byte sequence identifying the type of database file.

This can be useful if you want this information and cannot get it from the file extension or type and creator. The 8 characters "DOCUMENT" designate a database that contains an InDesign document and would normally have a .indd extension.

---

<code>fObjectStreamEndian</code>	<p>The byte order (big-endian or little-endian) in which integers in an object stream are written, which depends on the creation platform of the database.</p> <p>You must use this byte-order to interpret the packet-length value if you enlarge XMP packets and rewrite the file format; see <a href="#">“Rewriting XMP data beyond the packet padding” on page 39</a>.</p> <p>You do not need it to find or interpret the XMP data; all integer values in the storage structures themselves are uniformly little-endian.</p>
<code>fSequenceNumber</code>	<p>The master page sequence number, a value that is incremented each time a new master page is written.</p> <p>To ensure data integrity, the current master page is written in an alternating fashion to the first two pages in the file. You must discover which master page of the pair is the current one by finding which has the larger sequence number. This integer field, like the others in the storage structures, is written in a standard little-endian form regardless of database creation platform.</p>
<code>fFilePages</code>	<p>The number of 4K pages (including the master pages) in the transacted database part of the file. To efficiently search for XMP metadata, seek to and read from the absolute file position <code>4096 * fFilePages</code>.</p>

---

## Header and trailer structure

The following code shows the parts of the Header and Trailer of a contiguous object which are of interest with respect to metadata:

```
struct ContiguousObjectStreamHeaderOrTrailer {
    // 16 byte GUID identifying this as either a header or trailer
    // Headers are: DE393979-5188-4b6c-8E63-EEF8AEE0DD38
    // Trailers are: FDCEDB70-F786-4b4f-A4D3-C728B3417106
    uint8 fGUID [16];

    // UID of the corresponding object in the database
    LittleEndianUnsignedInt32 fObjectUID;

    // ClassID of the object. This object is externally
    // writable if and only if fObjectClassID & 0x40000000 == 0x40000000.
    LittleEndianUnsignedInt32 fObjectClassID;

    // Length of the stream of data which is the persistent
    // representation of the object. This is the size in
    // bytes of the data between the header and trailer.
    LittleEndianUnsignedInt32 fStreamLength;

    // The ADLER32 checksum (see RFC1950) of the stream of
    // data at the time it was last written by InDesign.
    // InDesign only propagates externally writable contiguous
    // objects back into the database when there is a checksum mismatch.
    LittleEndianUnsignedInt32 fChecksum;
};
```

<code>fGUID</code>	<p>A16-byte GUID identifying the structure as either a contiguous object header or trailer. This is the only difference between a header and trailer, which otherwise hold copies of the same information.</p> <ul style="list-style-type: none"> <li>➤ The header ID value is:           <pre>DE393979-5188-4b6c-8E63-EEF8AEE0DD38</pre> </li> <li>➤ The trailer ID value is:           <pre>FDCEDB70-F786-4b4f-A4D3-C728B3417106</pre> </li> </ul>
<code>fObjectUID</code>	<p>The unique identifier of the original object in the database, whose persistent representation has a contiguous copy here at the end of the file.</p> <p>Because there is a one-to-one correspondence between object copies here at the end of the file and objects stored in the main database, you are not free to add an object containing an XMP packet if one does not exist. Also, removing a contiguous object copy will not result in the deletion of the corresponding database object when the file is next opened by InDesign; the best you could do to remove all metadata is rewrite an existing object to contain an XMP packet that is well formed but devoid of content.</p>
<code>fObjectClassID</code>	<p>Information about the object type, a set of bit flags. The second most significant bit of this field, when set, declares the object as externally writable.</p> <p>If you find an object containing an XMP packet but this bit is not set, changes to that object are not allowed and would not be propagated back into the database. InDesign makes all XMP objects writable.</p>
<code>fStreamLength</code>	<p>The number of bytes in the persistent representation of the object, which is also the number of bytes between the header and the trailer.</p> <p>If you are writing new headers and trailers because you are enlarging an XMP object, you need to update this field in the header and trailer. For an XMP object, this is the size of the XMP packet itself plus a 4-byte integer specifying the length of the XMP packet; see <a href="#">“Rewriting XMP data beyond the packet padding” on page 39</a>.</p>
<code>fChecksum</code>	<p>The ADLER32 (see RFC1950) checksum of the object stream (all data between header and trailer) at the time InDesign wrote it out. InDesign uses this when it opens the database to check whether a contiguous object copy has changed, and the changes need to be propagated into the primary part of the database.</p> <p>If you rewrite an object, set the <code>fChecksum</code> field to <code>0xffffffff</code> to ensure that a checksum mismatch occurs and InDesign recognizes your changes.</p>

## PSD (Adobe Photoshop)

An Adobe Photoshop® .psd file is divided into five sections. The file header and image resource sections are important for metadata access. Multibyte integers are stored big-endian on all platforms.

The file sections relevant to metadata are:

File header	A 26 byte block; see <a href="#">“PSD file header block” on page 43</a> . Only the signature and version fields affect metadata.
Color mode data	Contains a 4 byte length value, followed by the data.
Image resources	Used to store non-pixel data associated with an image. Begins at file offset 30 (26+4) plus the length of the color mode data.  The section contains a 4 byte length value, followed by the data for all image resources. The length value is the total length of the image resource section, which is a sequence of individual image resources.  The internal structure of an image resource is described in <a href="#">“Photoshop image resources for metadata” on page 70</a> . The TIFF and JPEG file formats can also contain Photoshop image resources, although the content varies according to what file format the resource is in.

**NOTE:** TIFF, JPEG, and PSD share complex issues of how native metadata formats are stored; see [“Native metadata in digital photography formats” on page 62](#).

#### PSD file header block

Offset	Length	Description
0	4	File signature, must be "8BPS"
4	2	Version, 1 for normal files, 2 for "big" files. This distinction is primarily in the maximum image dimensions. There is no distinction between them for metadata parsing.
6	6	Reserved, must be zero
12	2	Number of channels
14	4	Image height in pixels
18	4	Image width in pixels
22	2	Number of bits per channel
24	2	Color mode

#### PSD image resource blocks

Field	Type	Description
Type	OStype	Adobe applications always uses the signature 8BIM. Other values can appear, and must be tolerated by readers.
ID	2 bytes	ID = 1060 for XMP metadata.
Name	PString	There is no name value in XMP image resources.

Field	Type	Description
Size	4 bytes	Actual size of resource data. This does not include the <code>Type</code> , <code>ID</code> , <code>Name</code> , or <code>Size</code> fields.
Data	Variable	Resource data, padded to make size even (that is, an extra zero byte is appended to the “raw” field value if needed).  This is the XMP packet, which must be encoded as UTF-8.

**Reference** Official documentation for the Photoshop file format is part of the Photoshop SDK. Information about the Photoshop SDK is available from:

<http://partners.adobe.com/public/developer/photoshop/devcenter.html>

## Markup formats

- [“HTML” on page 44](#)
- [“XML” on page 45](#)

### HTML

XMP embedded in HTML should conform to one of the W3C recommendations for embedding XML in HTML. For reference information, see the meeting report for the May 1998 W3C meeting:

<http://www.w3.org/TR/NOTE-xh>

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.

### Embedding XML in HTML

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>
```

## 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** For reference information, see the XML specification:

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

## Document formats

- [“PDF” on page 45](#)
- [“PS, EPS \(PostScript and Encapsulated PostScript\)” on page 46](#)

## PDF

For PDF files, the XMP packet is embedded in a metadata stream contained in a PDF object (beginning with PDF 1.4). The XMP must be encoded as UTF-8.

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*:

[http://www.adobe.com/devnet/pdf/pdf\\_reference.html](http://www.adobe.com/devnet/pdf/pdf_reference.html)

**NOTE:** The PDF specification (not this XMP Specification) is the authority for how XMP is embedded in PDF.

## PS, EPS (PostScript and Encapsulated PostScript)

PostScript<sup>®</sup> is a page description and programming language. Encapsulated PostScript (EPS), is a DSC-conforming PostScript document with additional restrictions intended to make EPS files usable as a graphics file format. EPS files are more-or-less self-contained, reasonably predictable PostScript documents that describe an image or drawing, that can be placed within another PostScript document.

XMP metadata can be placed in PostScript<sup>®</sup> 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 47](#), 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.

The XMP in a PostScript/EPS file must be encoded as UTF-8.

## Document-level metadata in PostScript

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 47](#).
- Write the XMP packet as described under [“Document-level XMP in PostScript” on page 48](#).

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

- Write the `%ADO_ContainsXMP` comment as described under [“Ordering of content” on page 47](#).
- Write the XMP packet as described under [“Document-level XMP in EPS” on page 49](#).

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 48](#).

### 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> ...
```

The marker must be at the beginning of a line. 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.

**NOTES:** 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.

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 50](#) and [“LanguageLevel 1 for PostScript and EPS” on page 50](#).

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, such as:

```
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 (see "Avoiding name conflicts" on page 50).

/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` **pdfmarks** 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` **pdfmarks**. 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 and 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 do not 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, for example “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, for example 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 in PostScript

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 48](#)), 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
```

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.

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.

## UCF (Universal Container Format)

UCF is a general-purpose container technology. As a general container format, UCF collects a related set of files into a single-file container. UCF can be used to collect files in various document and data formats and for classes of applications. The single-file container enables easy transport of, management of, and random access to, the collection.

UCF defines rules for how to represent an abstract collection of files (the “abstract container”) into physical representation within a Zip archive (the “physical container”). The rules for Zip containers build upon and are backward compatible with the Zip technology used by Open Document Format (ODF) 1.0. UCF is designed to provide a set of lightweight constraints on the use of Zip.

The position of XMP within UCF is defined in part 3.5 of the specification:

### 3.5 META-INF

*All valid UCF Containers MAY include a directory called "META-INF" at the root level of the container file system. This directory contains the files specified below that describe the contents, metadata, signatures, encryption, rights and other information about the contained publication.*

Specifically, the META-INF/metadata.xml file contains XMP metadata:

#### 3.5.3 Metadata – META-INF/metadata.xml (Optional)

*A file with the reserved name "metadata.xml" within the "META-INF" directory at the root level of the container file system may appear in a valid UCF container. This file, if present, MUST be used for container-level metadata. In version 1.0 of OCF, no such container-level metadata is specified.*

*If the "META-INF/metadata.xml" file exists, its contents MUST be valid XML with namespace-qualified elements to avoid collision with future versions of OCF that MAY specify a particular grammar and namespace for elements and attributes within this file.*

*Adobe-defined formats based on UCF MUST use XMP to specify metadata.*

**Reference** For further information about UCF, see:

<http://labs.adobe.com/technologies/mars/?tab:details=1#documentation>

# 4 Handling Native Metadata

There is a generic policy for how XMP handles native metadata, which covers common practices that should apply to all file formats. However, since application support for XMP long preceded this policy, there are historic and format specific cases where the generic policy is not followed.

- In most file formats there is just one form of native metadata, and the generic policy is presented in these terms.
  - The mappings for PDF are discussed in [“Native metadata in PDF files” on page 57.](#)
  - The individual mappings for dynamic media formats are discussed in [“Native metadata in dynamic media formats” on page 59.](#)
- The common still-image formats (JPEG, TIFF, and PSD) have multiple forms of native metadata; those additional complications are described in [“Native metadata in digital photography formats” on page 62,](#) and complete details are given in [Chapter 5, “Digital Photography Native Metadata.”](#)

## Reconciling metadata in different formats

Key to all native metadata handling is the notion of reconciling the native metadata with the XMP. Native metadata information can be imported into XMP, and XMP information can be exported to native formats. The generic policy defines when reconciliation happens and how values are transformed. The intent is to preserve the most recent and informative metadata, while allowing the use of both old and new applications.

The basic programming model is that XMP is the only active metadata at runtime:

- When a file is opened, all existing metadata is reconciled into XMP. This does not necessarily mean that data is *imported* to XMP; an import operation only happens if the native metadata appears to be newer than the XMP.
- The XMP is modified as appropriate while the file is open.
- When the file is saved, the relevant XMP is *exported* to native formats, and both forms are written to the file.

**NOTE:** This is a conceptual programming model meant to aid discussion of the issues of policy. The XMP specification does not require applications that support XMP to follow this specific model.

## Text encodings in import and export

It is important to understand text encoding issues before getting into the details of import and export.

The XMP is always written using Unicode, generally UTF-8 although which Unicode does not matter. What does matter is that Unicode is a single text encoding for all languages. The most recent Unicode standard defines over a million characters.

Native metadata often uses more restricted encodings such as 7-bit ASCII, ISO Latin-1, or JIS. In many cases it even uses an ill-defined "local" encoding, the default encoding in effect on a user's machine at any

particular moment. There is usually nothing in a file telling what the local encoding was when the file was written.

The use of non-Unicode encoding for native metadata means that information can be lost when exporting XMP to native formats. The text encoding used by the native format might not support some of the Unicode characters in the XMP. This typically comes across as question marks in the native format for unsupported characters. Information would be lost from the Unicode if that native format were then imported back to the XMP, downgrading the XMP value.

The XMP metadata can also be corrupted by interactions with locally encoded native metadata. Consider the case of a file saved in Japan using JIS for the local encoding, that is then sent to the US, where Latin-1 is the local encoding. Suppose, for example, that the file is first processed in the US by an application that does not understand XMP and makes the native metadata appear newer than the XMP. If the file is then opened in a XMP-aware application, the native metadata is imported to the XMP. At this time, the JIS values from Japan are interpreted as Latin-1, then converted to Unicode and placed in the XMP. The result is complete garbage (sometimes called “mojibake”) in the XMP, which is then exported back to the native format when the file is saved.

The native-metadata reconciliation policies are designed to minimize these problems.

## Native metadata export policy

An XMP export operation must create all appropriate native metadata formats, even if they did not previously exist in the file. It should delete any existing native-format items if their corresponding XMP properties are not present, but must not alter any native-format items that do not correspond to XMP properties.

There are exceptions to this policy; some metadata formats contain explicit marking for what to do when a file is modified. The mark usually has at least two forms:

- If you do not understand this item, delete it when updating the file content (that is, the described data, not the metadata).
- If you do not understand this item, delete it when doing anything to the file.

An export operation can include specific property value conversions, such as those for text encodings or date-time formats. Numeric values, in particular, must often be converted between the XMP text representation and format-specific binary representations.

When a file is saved, the XMP is first exported to the native format, then an overall MD5 signature is computed for the native metadata. This signature is saved in the XMP and used later to detect changes to the native metadata.

The exact input for the signature must be well defined for each file format. It should be convenient and efficient to compute the signature. For example, the entire legacy IPTC (IIM) in an image file is a reasonably small contiguous block, so the MD5 signature is computed for the entire block. The Exif native metadata is a distributed and unordered TIFF stream, so that signature is computed using just the relevant metadata values.

## Native metadata import policy

The decision of whether to import native metadata is based on both a digital signature and value comparison. When a file is opened, a new signature is computed for the native metadata in the file. If this

matches the signature saved in the XMP, then the native metadata is unchanged and the XMP can be imported as it is.

This new signature is computed based on the native metadata items that the current application would import, without regard to what items the previous application exported. If these sets differ, the signatures do not match and the next step adequately protects the XMP.

If the overall signatures do not match then something in the native metadata has changed and the XMP might need to be updated. In this case an import decision must be made on each native item by value comparison. The XMP is used to compute an expected native value; for example one that includes a text encoding change. This expected value is compared to the native value in the file. If they differ, the native item has changed and needs to be imported to the XMP.

This extra step prevents a downgraded value in an unchanged native item from clobbering the XMP when the overall signature changes because of some other native item. The extra step does not prevent the corruption of XMP due to an import after a shift in local encoding. In that case, the expected native value computed from the XMP would use the new local encoding; it would certainly differ from even an unchanged native value, and would cause an erroneous import.

Users can prevent this corruption by saving the file with an XMP-aware application before transfer, then opening and resaving a file in an XMP-aware application after transfer. Do not use any applications that might change just the native metadata in between. When the file is opened, the native signature will match, so no imports will happen. When the file is resaved, the new local encoding is used, so that the extra value check will work subsequently.

If a native item is missing and the corresponding XMP item is present, the import operation should not delete the XMP item. This is a pragmatic choice to accommodate a few applications that edit only the XMP.

## Native metadata in PDF files

PDF metadata is stored with a document in the Info dictionary. XMP metadata properties are mapped to defined document Info dictionary keys.

In PDF 1.5, properties are mapped as follows:

PDF Document Info key	XMP metadata property	Mapping notes
Title	dc:title	The <code>Title</code> key maps to the first of the alternatives given in the <code>dc:title</code> property.
Author	dc:creator	The <code>Author</code> key maps to the first of the creators listed in the <code>dc:creator</code> field.  Alternatively (available by user action in the Acrobat 7 UI), maps to a concatenated list of the creators listed in the <code>dc:creator</code> field separated by a standard separator character such as semicolon.
Subject	dc:description	The <code>Subject</code> key maps to the first of the alternatives given in the <code>dc:description</code> property.

PDF Document Info key	XMP metadata property	Mapping notes
Keywords	pdf:Keywords	The XMP properties <code>dc:subject</code> and <code>pdf:Keywords</code> have historically been separate. In Acrobat 7, Adobe allows user intervention to set them to corresponding values, where the value in the PDF schema (and in the DocInfo) is set to a delimiter-separated concatenation of the bag of values found in the <code>dc:subject</code> value.
Creator	xmp:CreatorTool	
Producer	pdf:Producer	
CreationDate ModDate	xmp:CreateDate xmp:ModifyDate	Info dictionary dates are in ISO/IEC 8824 format, XMP dates are in ISO 8601 format.
Trapped	pdf:Trapped	The <code>Trapped</code> key takes name values from a controlled set. The text values used for the <code>pdf:Trapped</code> property are the spellings of the corresponding names.

## User-defined keys

In addition to these correspondences, PDF defines an XMP schema (`pdfx:`) dedicated to representing any user-defined keys present in the Info dictionary. Because the names of such properties can contain arbitrary Unicode characters, and XMP names are constrained to be XML Name characters. The following escape convention applies in mapping between the PDF names and the XMP names:

- Characters that are not XML Name characters are represented by the character ROMAN NUMERAL TEN THOUSAND followed by four characters from among the hexadecimal digits; these four characters spell out the Unicode code point of the character being escaped.
- Characters having Unicode code points beyond the Basic Multilingual plane have their high, then low, surrogate parts spelled out in this fashion by two consecutive five-character sequences.

## Resolving metadata conflicts

If a PDF document contains both an Info dictionary and document-level XMP metadata, it is possible for the representations to get out of synchronization (for example, if Info values are set by software that is unaware of the XMP representation). In order to reconcile such differences, Adobe Acrobat compares the time given by the Info `ModDate` value with the time given by the `xmp:MetadataDate` value.

- If the Info value is more recent, all the Info dictionary entries supersede their corresponding XMP properties
- Otherwise, all the XMP values are assumed to be in force.

## Native metadata in dynamic media formats

The following dynamic media formats can contain native metadata which must be reconciled with XMP:

- [Native metadata in ASF \(WMA, WMV\)](#)
- [Native metadata in AVI](#)
- [Native metadata in MP3](#)
- [Native metadata in WAV](#)

Native metadata in these formats is generally more straightforward than for still image formats; the formats and details of individual properties are both discussed here.

### Native metadata in ASF (WMA, WMV)

The ASF Header Object contains several nested objects that have native metadata fields mapped to XMP:

ASF	XMP
File Properties Object, Creation Date	xmp:CreateDate
<p>This property is ignored, neither imported nor exported, if the <code>Broadcast</code> flag is set. The ASF <code>Creation Date</code> is a UTC time. Recommended policy is to add an appropriate time zone offset in the XMP value, using the local time zone as a default.</p>	
Content Description Object, Title	dc:title['x-default']
Content Description Object, Author	dc:creator[*]
<p>The native value is read as a single string of semicolon separated values, mapped to the <code>dc:creator</code> array in XMP.</p>	
Content Description Object, Copyright	dc:rights['x-default']
Content Description Object, Description	dc:description['x-default']
Content Branding Object, Copyright URL	xmpRights:WebStatement

All native metadata strings are presumed to be little endian UTF-16. The value contains no U+FEFF BOM. Values written to the native format should have a 16-bit NULL terminator. The terminator should be treated as optional when importing to XMP, and stripped if present (not propagated to XMP).

The ASF specification says that the Copyright and License URL values must be ASCII. They should be taken as is when importing to XMP, doing a normal UTF-16 to UTF-8 conversion. When exporting from XMP each byte that is not normal "display" ASCII should be replaced with '?'. The accepted range is 0x21..0x7E.

If an object has to be created to hold a native item, default values (generally 0) should be written for unknown portions of the object.

An MD5 digest of the native metadata is kept in the XMP as `asf:NativeDigest`. The `asf:` namespace URI is "`http://ns.adobe.com/asf/1.0/`". Only the MD5 digest value should be used for ASF comparisons.

The MD5 digest is computed using the value of the native items, accumulated in the order listed above. The digest is written as an optional series of comma-separated small integers, a semicolon, and a 32-character hexadecimal string for the actual digest value. A leading semicolon is present when the initial

small integers are omitted. (The small integers are `C_enum` values indicating the native items actually present when written. They are ignored since this is not in keeping with preferred native-metadata policy.)

## Native metadata in AVI

AVI files can have four pieces of reconciled native metadata in a LIST/Tdat container chunk:

AVI	XMP
tc_O	dm:startTimecode/dm:timeValue
tc_A	dm:altTimecode/dm:timeValue
rn_O	dm:tapeName
rn_A	dm:altTapeName

The native values are all written as UTF-8.

There is currently no digest in AVI to detect changes in the native metadata.

## Native metadata in MP3

The following native ID3 frames are mapped to XMP:

ID3	XMP
COMM	dm:logComment
TALB	dm:album
TCON	dm:genre
TIT2	dc:title["x-default"]
TPE1	dm:artist
TRCK	dc:trackNumber
TYER, TDAT, TIM	xmp:CreateDate (for ID3v2.3)
TDRC	xmp:CreateDate (for ID3v2.4)

A known genre should be mapped to XMP as the textual genre name, not the raw number string.

For maximum compatibility, text values in the native metadata frames should be written as true Latin-1 or as little-endian UTF-16. For example, some versions of the Windows Media Player ignore the U+FEFF BOM and presume little endian UTF-16.

## Native metadata in MPEG-4

MPEG-4 can contain native metadata, which can require reconciliation with the XMP. Metadata can occur in several boxes, as defined by in ISO 14496-12:

```
moov/mvhd
moov/udta/cprt
moov/trak/tkhd
moov/trak/udta/cprt
```

An MPEG-4 file must contain exactly one `moov` box, which must contain exactly one nested `mvhd` box; the `mvhd` box provides information about the file as a whole, including creation and modification times, duration, and timescale. The `mvhd` box is not a container; it is a fixed size record defined in ISO 14496-12 section 8.3.

The `moov` box can optionally contain at most one `udta` box, which may optionally contain any number of `cprt` boxes. A `cprt` box contains a copyright for the file as a whole. Each has a language code and copyright string.

The `moov` box also contains one or more `trak` boxes, each of which contains exactly one `tkhd` box, optionally at most one `udta` box, and any number of `udta/cprt` boxes. These boxes contain similar copyright information for the individual tracks.

### Reconciling native metadata with XMP

The XMP is reconciled with native metadata from the `moov`-level `mvhd` and `udta/cprt` boxes. The native values are imported into the XMP as needed. Individual native items are either always updated when the XMP is updated, or are never updated from the XMP.

- The `mvhd` `creation_time` and `modification_time` are mapped to `xmp:CreateDate` and `xmp:ModifyDate` respectively. The native dates are always be updated when updating the XMP.
- The `mvhd` `timescale` and `duration` are used to compute a value for `xmpDM:duration`, with enough fractional digits to cover the time scale; for example, a `timescale` of 60 (1/60th of a second) corresponds to two fractional digits (1/100th of a second) in the XMP. The `mvhd` `timescale` and `duration` are never updated from the XMP.
- The `cprt` values are mapped to items in the `dc:rightsLangAlt` array. The native values contain an encoded form of an ISO-639-2 language code. XMP uses RFC 3066 codes, which typically include an ISO-639-2 language code plus an ISO 3166 country code.

### Importing native metadata values

Native values are imported into the XMP as needed, as determined from an MD5 digest in the `xmp:NativeDigests/xmp:MPEG-4` property in the XMP.

- If no XMP is present, initial XMP is constructed by importing the native metadata.
- If XMP is present but lacks the MPEG-4 digest, existing XMP values are kept and new ones added from the native. (This is for the benefit of XMP writers who fail to write the digest.)
- If XMP is present with the MPEG-4 digest, a 128-bit MD5 digest is computed from the full data portion of the `mvhd` box, and the full data portion of all `moov`-level `cprt` boxes. The `mvhd` data is added first, followed by the `cprt` boxes in file order. This digest is converted to an ASCII hexadecimal string using capital A-F, with no internal separators. If the new digest differs from the old digest, the native metadata is imported again.

A new digest is computed and saved whenever the XMP is updated.

## Native metadata in WAV

Native metadata in a LIST/INFO container chunk is defined by "Multimedia Programming Interface and Data Specifications 1.0". This information is also contained in the Exif 2.2 specification, in section 5.6 for "Exif audio files". The values are specified as null terminated ASCII strings.

For WAV files, this subset of the LIST/INFO native metadata is reconciled with XMP:

WAV	XMP
IART	dm:artist
ICMT	dm:logComment
ICOP	dc:rights["x-default"]
ICRD	xmp:CreateDate
IENG	dm:engineer
IGNR	dm:genre
INAM	dm:album
ISFT	xmp:CreatorTool

All of the WAV native metadata is stored using local encoding, not necessarily ASCII. The file contains no indication of the local encoding in use.

There is currently no digest in WAV to detect changes in the native metadata.

## Native metadata in digital photography formats

Finding and interpreting the metadata embedded in PSD, TIFF, and JPEG files is complicated by the fact that all three file formats contain the same kinds of metadata (XMP, TIFF, Exif, and IPTC), but store it slightly differently.

For example, all of the kinds of metadata can be contained in Photoshop Image Resources (PSIRs), and all three file formats (PSD, TIFF, and JPEG) can contain PSIRs. However, the specific contents of the PSIRs are different when contained in different image file formats. Each type of metadata is stored inside the PSIR for some file formats, and separately for others.

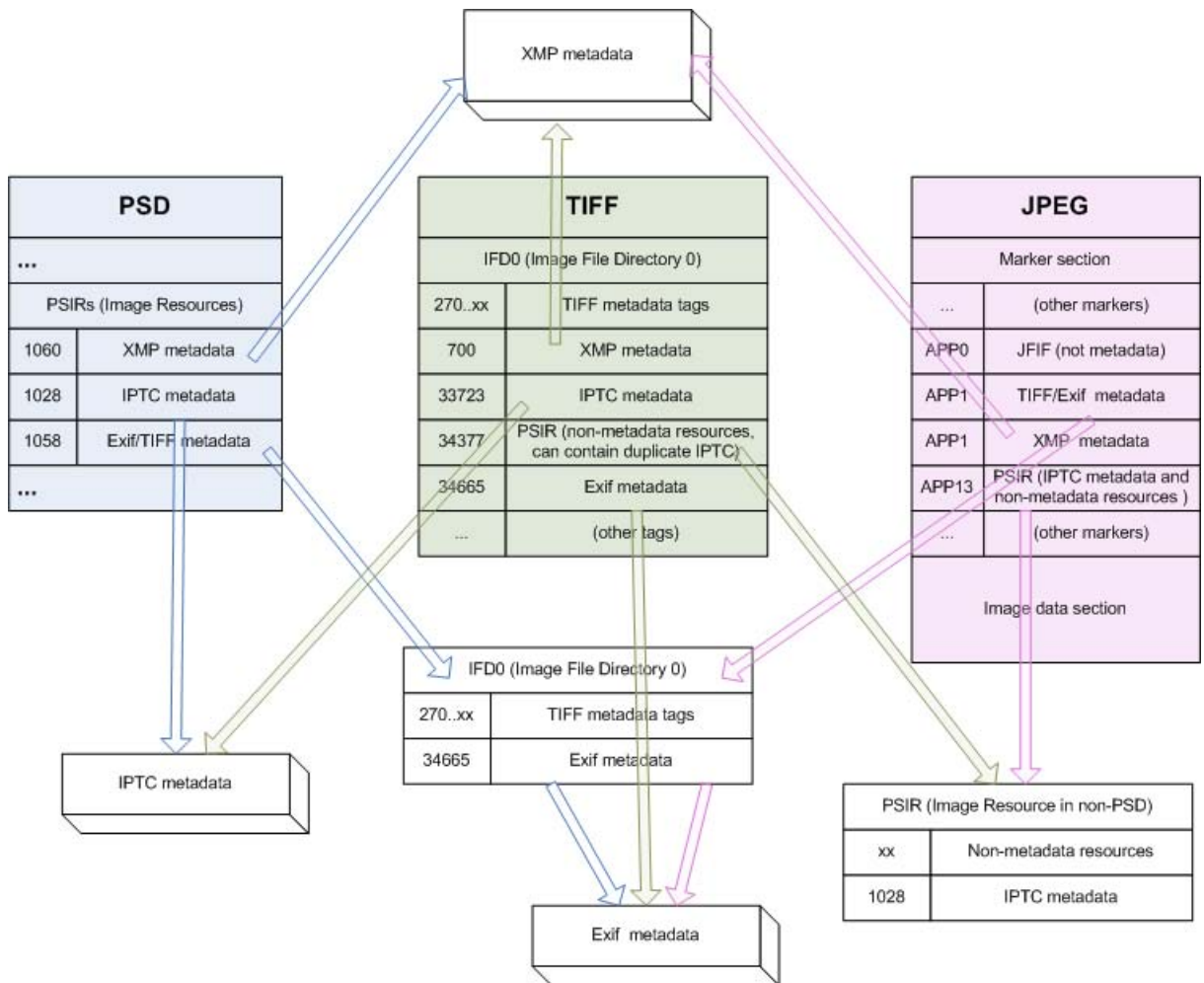
Similarly, TIFF contains both TIFF-specific and Exif metadata, which is stored slightly differently in the TIFF format itself than in the other formats.

This section provides an overview of the storage similarities and differences. The specific details of each native metadata format, and the specific properties that are present in different formats and need to be reconciled, are discussed in [Chapter 5, "Digital Photography Native Metadata."](#)

The following figure illustrates how metadata is stored in PSD, TIFF, and JPEG files, and what the PSIR contains in each case.

- In a TIFF file, the arrows in the diagram indicate offsets that point to the data; the data itself resides elsewhere in the file.

- A Photoshop image resource (PSIR) actually contains the data pointed to by the arrow in the diagram.
- In JPEG, the APP<sub>n</sub> sections also contain the data pointed to by the arrow.



Notice these points:

- A PSD file contains PSIRs, with one image resource for each type of metadata.
  - The resource for Exif/TIFF metadata points to a TIFF IFD0. The IFD contains the TIFF metadata tags and an Exif tag that points to the Exif metadata.
- TIFF, JPEG, and PSIRs all have direct pointers to XMP metadata.
  - The PSIR that appears within TIFF and JPEG does not include an XMP resource, since those formats have direct pointers to the XMP.
- PSD and TIFF point directly to the IPTC metadata, but JPEG contains a PSIR that contains the IPTC.
  - TIFF also contains a PSIR, which might contain a duplicate of the IPTC that is pointed to directly by its own tag.
- TIFF contains a tag that points directly to the Exif metadata; however, JPEG (like PSD) has a marker that points to a TIFF IFD0. The IFD contains the TIFF metadata tags and an Exif tag that points to the Exif metadata.

## Reconciliation issues

This section clarifies specific issues of reconciling native metadata with XMP for digital photography formats.

Prior to the advent of XMP, the reconciliation issues among the native forms of metadata were relatively small. The text encodings were the same (almost all local OS), and there was little duplication. The entire IPTC block was duplicated in TIFF files (tag 33723 and image resource 1028), and in the ANPA resource for all Mac OS files. Only 4 individual properties appeared in multiple forms (listed here by their XMP names):

<code>dc:description["x-default"]</code>	IPTC 2:120, Caption TIFF 270, ImageDescription Mac OS <code>pnot resource Desc item</code>
<code>dc:creator</code>	IPTC 2:80, By-line TIFF 315, Artist
<code>dc:rights["x-default"]</code>	IPTC 2:116, Copyright Notice TIFF 33432, Copyright
<code>dc:subject</code>	IPTC 2:25, Keyword Mac OS <code>pnot resource KeyW item</code>

IPTC is the one common form for these 4 properties, and the IPTC data structures can be used as the primary storage. The other forms can then be incorporated into the IPTC storage as necessary. Adobe practice has been to collate the IPTC while reading a file, using a priority scheme to decide whether to incorporate various sources of metadata. The priorities among non-XMP formats are:

- If IPTC from TIFF tag 33723 is present, use it.
- If TIFF text tags 270, 315, 33432 are present, use them (except for PSD files).
- If IPTC from PSIR 1028 is present, use it.

Here is the priority scheme as pseudocode:

```
if IPTC wins
    if TIFF digest is missing or differs then import TIFF tags
    if Exif digest is missing or differs then import Exif tags
    if IPTC digest is missing or differs then import IPTC DataSets
else
    if IPTC digest is missing or differs then import IPTC DataSets
    if TIFF digest is missing or differs then import TIFF tags
    if Exif digest is missing or differs then import Exif tags
endif
```

Notice that there are no Exif properties among those of concern; there are no conflicts, and hence no ordering constraints, between TIFF and Exif.

## Encoding of text in metadata

The encoding of text strings varies among the metadata formats, and Adobe practice does not always precisely follow the relevant specifications. The only easy case is XMP, which always uses Unicode; specifically UTF-8 for PSD, TIFF, JPEG, JPEG-2000, GIF, PNG, DNG, and camera raw files.

The Mac OS `STR#` and `TEXT` file resources use local OS single-byte encoding, with no indication in the file of what that is.

Photoshop image resources have one active and two obsolete standalone strings. These all use local OS single-byte encoding, with no indication in the file of what that is. The active string is image resource 1035, copyright information URL. The obsolete strings are 1008, old caption, and 1020, old print caption.

The major native standards specify 7-bit ASCII for the most part:

---

TIFF/ Exif	All but 3 string values have type ASCII, which means 7-bit ASCII. The remaining 3 have directly indicated encoding:
------------	---

UserComment (37510)  
GPSProcessingMethod (27)  
GPSAreaInformation (28).

---

IPTC	All values are written as ISO 646, which is 7-bit ASCII with national variants.
------	---

---

Photoshop takes liberties with the interpretation of ASCII and ISO 646. Through version 7, it uses local OS single-byte (8-bit) encoding where TIFF/Exif and IPTC specify 7-bit ASCII or ISO 646. There is no indication in the file of what this encoding is. Photoshop 9 continues to use local OS single-byte (8-bit) encoding for IPTC, but uses UTF-8 in TIFF/Exif for tags of type ASCII.

Photoshop also ignores the Exif admonition to use the UserComment tag instead of ImageDescription when the value contains non-ASCII characters. Photoshop always writes ImageDescription, never UserComment.

# 5 Digital Photography Native Metadata

This chapter discusses how native metadata is stored in the various still-image formats ([“Metadata storage in native formats” on page 66](#)), and provides details of the properties that are equivalent in different formats, and how they are mapped to XMP ([“Reconciling metadata properties” on page 70](#)).

## Metadata storage in native formats

This section provides details of the native metadata formats used in the common digital photography formats (PSD, JPEG, and TIFF). Each of these file formats can contain metadata in any of the following formats, in addition to XMP:

- [“TIFF metadata” on page 66](#)
- [“Exif metadata” on page 66](#)
- [“Photoshop image resources” on page 67](#)
- [“IPTC \(IIM\) metadata” on page 68](#)

For details of the specific properties that need to be reconciled among metadata formats, see [“Reconciling metadata properties” on page 70](#).

## TIFF metadata

Modification of TIFF files must be done with care to preserve the linked structure. The Exif and GPS IFD pointers (see [“Exif metadata” on page 66](#)) illustrate the potential problem of detecting implicit links. These tags have type LONG and a count of 1, the fact that they are links is not explicit. Nothing prevents new versions of Exif from adding similar IFD pointers, or applications from adding similar private tags. Updates to TIFF files should be done in one of these ways:

- Rewrite the entire file, output only those tags that are explicitly understood.
- Use an update by append approach. Append all new or resized IFDs and values to the end of the file and adjust known offsets to them.
- The TIFF specification does not allow tags to be repeated in an IFD. You should write tags in order, with no duplicates. A robust TIFF reader, however, should tolerate tags that are out of order. If duplicates do occur, Adobe practice is to keep the last encountered tag.

For information on specific metadata tags, see [“TIFF and Exif tags for metadata” on page 73](#).

## Exif metadata

The Exif image interchange format defines a set of TIFF tags that describe photographic images, and is widely used by digital cameras. Exif metadata is found in TIFF, JPEG, and PSD files.

According to Exif conventions, the first IFD (IFD0) describes the primary image, the next IFD (IFD1) describes thumbnails. It allows either uncompressed images based upon TIFF, or compressed images

based upon JPEG. In fact, the IFDs may or may not contain image data, depending on whether they are in a TIFF or a JPEG file.

The additional information defined by Exif is located through these TIFF tags, regardless of the surrounding file format:

Tag	Hex	Name	
34665	0x8769	Exif IFD pointer	➤ In a TIFF file, IFD0 contains tags of this type, which point to subsidiary IFDs that contain tags that point to the data blocks.
34853	0x8825	GPS IFD pointer	<ul style="list-style-type: none"> <li>➤ In PSD files, an image resource contains these IFDs, which point directly to the data. The data is contained within the PSIR.</li> <li>➤ In JPEG files, an APP1 marker contains these IFDs, which point directly to the data. The data is contained within the marker.</li> </ul>

**IMPORTANT:** Even though JPEG is a big-endian format overall, the TIFF within the Exif APP1 marker segment can be little-endian. Readers must recognize and honor the 8-byte TIFF header.

For information on specific metadata tags, see [“TIFF and Exif tags for metadata” on page 73](#).

**Reference** Official documentation for the Exif file format is available from:

<http://www.exif.org/Exif2-2.PDF>

## Photoshop image resources

Photoshop image resources are used in PSD, TIFF, and JPEG files. The contents differ in the different contexts:

- The image resource section is a native part of the PSD file format, and consists of a 4-byte length value, followed by a sequence of image resources, which include both non-metadata resources and all of the metadata (XMP, TIFF/Exif, and IPTC).
- JPEG and TIFF file formats contain just the sequence of image resources; the overall length is part of the containing file structure.
  - In TIFF, the resource pointer is a TIFF tag 34377 (0x8649). It points to sequence of image resources that contain non-metadata resource data, and can contain duplicate IPTC metadata (which also has its own TIFF tag). The image resources do not contain XMP or TIFF/Exif metadata, which have their own TIFF tags.
  - In JPEG, the resource is contained in an APP13 marker segment with a signature string of "Photoshop 3.0". It contains a sequence of image resources that contain non-metadata resource data and IPTC metadata. The image resources do not contain XMP or TIFF/Exif metadata, which have their own markers.

Each Photoshop image resource has the following form:

Offset	Length	Description
0	4	Signature, the 4 ASCII characters "8BIM"
4	2	Resource ID, a 16-bit big-endian unsigned integer. The XMP resource ID is 1060 in PSD files.
6	—	Resource name as a Pascal string.
—	4	Length of the resource data, a 32-bit big-endian unsigned integer
—	—	Resource data

The image resources need not be sorted in any way. The resource name and data are followed as necessary by a 0x00 pad byte to make the total length be even; that is, to make the following component begin on an even offset. The explicit length of the name or resource data does not include this pad.

This padding is relative to the image resource section, not necessarily to the overall file. For example, the image resources in a JPEG file might begin on an odd file offset because JPEG allows marker segments to begin on any offset. Readers and writers must use care to round local image resource relative offsets, not base file relative offsets.

The image resource ID and data length must always be written as big-endian, even when the image resources are embedded within little-endian TIFF.

For information on specific metadata properties in image resources, see ["Photoshop image resources for metadata" on page 70](#).

## IPTC (IIM) metadata

The International Press Telecommunications Council (IPTC) and NAA (Newspaper Association of America) have together defined the Information Interchange Model (IIM) that has come to be known as "IPTC metadata."

IPTC metadata is stored as a sequence of DataSets. The DataSets are logically grouped into Records. A DataSet is referenced as *record:dataset*; for example, DataSet 2 : 50 is DataSet 50 of Record 2. DataSets can be stored in a standard or extended form.

► The structure of a standard DataSet is:

Offset	Length	Description
0	1	Tag marker, 0x1C
1	1	Record number
2	1	DataSet number
3	2	Data length, big-endian, range 0..32767
5	—	Data

The structure of an extended DataSet is:

Offset	Length	Description
0	1	Tag marker, 0x1C
1	1	Record number
2	1	DataSet number
3	2	“Data length” length, the number of bytes in the variable-length “Data length” portion at offset 5. Big-endian, ORed with 0x8000
5	—	Data length, the number of bytes in the Data portion following. Big-endian, variable length as specified in preceding length value.
—	—	Data

The DataSets must be clustered by Record number, and the clusters written in ascending Record order. The DataSets within a Record cluster may be in any order.

The 16-bit data length field is always stored big-endian, on all platforms, even when the IPTC is placed within little-endian TIFF.

For information on specific metadata properties in DataSets, see [“IPTC DataSets for metadata” on page 71](#).

**Reference** The IIM specification can be found at:

<http://www.iptc.org/IIM/>

The more recent IPTC specification for metadata that is only in XMP, and not in the IIM structure, has no reconciliation issues and is not relevant to this discussion.

## Reconciling metadata properties

This section provides details about individual native metadata properties in native formats and how they are mapped to XMP.

- [“Photoshop image resources for metadata” on page 70](#)
- [“IPTC DataSets for metadata” on page 71](#)
- [“TIFF and Exif tags for metadata” on page 73](#)

Collections of metadata (for example, XMP or IPTC) are mentioned only where they are intermingled with individual properties. For example, the first table below shows this intermingling of individual properties in Photoshop image resources 1034 and 1035, and collections in 1028, 1058, and 1060.

### Photoshop image resources for metadata

There are several Photoshop image resources of interest as metadata:

ID	Hex	Description	XMP mapping	Notes
1028	0x404	IPTC metadata		
1034	0x40A	Copyright flag as a 0/1 Boolean	<code>xmpRights:Marked</code>	When importing, set the XMP to "True" if 1034 has length 1 and a non-zero value, otherwise leave the XMP alone.  When exporting, set 1034 to 0/1 if the XMP is present and True/False, else do not set 1034.
1035	0x40B	Text string for a copyright information URL	<code>xmpRights:WebStatement</code>	The image resource is treated as locally encoded text.
1058	0x422	Standard Exif (added in Photoshop 7)		In the standard TIFF format, including the initial 8 byte header. Even though Photoshop is a big-endian format overall, the TIFF within image resource 1058 can be little-endian. Readers must recognize and honor the 8 byte TIFF header.
1060	0x424	XMP (added in Photoshop 7)		
1061	0x425	MD5 digest of the IPTC (added in Photoshop 7)		The 16-byte binary value is the MD5 digest of the contents of PSIR 1028.

Some very old versions of Photoshop wrote image resource 1008 (0x3F0) or 1020 (0x3FC), but not both. These are now considered obsolete; both were dropped prior to Photoshop 6. The current Photoshop SDK does not even contain type information for 1020. Both of these are captions, rendered as Pascal strings; they are alternative forms corresponding to the IPTC Caption in DataSet 2:120. They are never exported to XMP.

## IPTC DataSets for metadata

Unless otherwise noted, the data in all DataSets of interest as metadata are graphic characters plus spaces from ISO 646. Which is essentially 7-bit ASCII overlain with a variety of national variants. These variants are all still within the 7-bit range. The national variant in use at any time is not specified and not detectable from the data. Graphic characters are in the range 0x21..0x7F, space (0x20) is not considered a graphic character (hence the “plus spaces” above). Alphabetic characters are the English alphabet in ASCII, A..Z (0x41..0x5A) and a..z (0x61..0x7A).

Unless otherwise noted, a given DataSet of interest as metadata can appear at most once. DataSets that can be repeated are marked in the following table with an X in the Repeat column. Those that can be repeated need not be contiguous.

A robust reader should tolerate improper repeats in input. Adobe practice depends on the mapping to XMP:

- A non-repeatable DataSet is always mapped to a simple XMP property; if there is an improper repeat, the last value encountered is kept.
- For a repeatable DataSet that is mapped to a simple XMP property, the last value encountered is kept. The XMP mappings for the repeatable DataSets 2:85 (`photoshop:AuthorsPosition`) and 2:122 (`photoshop:CaptionWriter`) are simple XMP properties, not arrays.
  - The DataSet 2:80 is repeatable, but is mapped to only the first item in the `dc:creator` array. It is treated as if it were mapped to a simple property. On import, only the last value is kept. On export from XMP, only the first item in the XMP `dc:creator` array is output as a 2:80 DataSet.
- For a repeatable DataSet that is mapped to an XMP array, all repeated values are both imported and exported as separate array items. An example is DataSet 2:25 (`dc:subject` array).

It is also Adobe practice to keep all of a value when importing to XMP, even if the value is larger than allowed (as shown in the following table). When exporting from XMP, large values are truncated to the allowed size.

The IPTC DataSets of interest as metadata are:

ID	Hex	Name	Repeat	Description
2:5	0x0205	Object Name		(title), maximum 64 bytes
2:10	0x020A	Urgency		one ASCII digit, '1'..'8'
2:15	0x020F	Category		alphabetic characters, maximum 3 bytes
2:20	0x0214	Supplemental Category	X	maximum 32 bytes each
2:25	0x0219	Keywords	X	maximum 64 bytes each
2:40	0x0228	Special Instructions		maximum 256 bytes

2:55	0x0237	Date Created		8 ASCII digits as CCYYMMDD, 00 for unknown parts
2:80	0x0250	By-line	X	maximum 32 bytes each
2:85	0x0255	By-line Title	X	maximum 32 bytes each
2:90	0x025A	City		maximum 32 bytes
2:95	0x025F	Province/State		maximum 32 bytes
2:101	0x0265	Country/Primary Location Name		maximum 64 bytes
2:103	0x0267	Original Transmission Reference		maximum 32 bytes
2:105	0x0269	Headline		maximum 256 bytes
2:110	0x026E	Credit		maximum 32 bytes
2:115	0x0273	Source		maximum 32 bytes
2:116	0x0274	Copyright Notice		maximum 128 bytes
2:120	0x0278	Caption		graphic characters, spaces, CR, LF, maximum 2000 bytes
2:122	0x027A	Writer	X	maximum 32 bytes each

The IPTC DataSet mappings to XMP are shown below. The namespace URIs and other details are in the *XMP Specification Part 2, Standard Schemas*. The notation `ns:array["x-default"]` means the item in the `ns:array` language alternative whose `xm1:lang` qualifier has the value "x-default".

ID	XMP mapping
2:4	<code>Iptc4xmpCore:IntellectualGenre</code>
2:5	<code>dc:title["x-default"]</code>
2:10	<code>photoshop:Urgency</code> (This is incorrectly given as 2:9 in the IPTC Core 1.0 specification.)
2:12	<code>iptc4xmpCore:SubjectCode</code>
2:15	<code>photoshop:Category</code>
2:20	<code>photoshop:SupplementalCategories</code> (gathered as an unordered array)
2:25	<code>dc:subject</code> (gathered as an unordered array)
2:40	<code>photoshop:Instructions</code>
2:55, 2:60	<code>photoshop:DateCreated</code> Reformatted as ISO 8601, which has both a date and a time; 2:55 is just a date; 2:60 is included for the time portion, if present.
2:80	<code>dc:creator[1]</code>

2:85	photoshop:AuthorsPosition
2:90	photoshop:City
2:92	iptc4xmpCore:Location
2:95	photoshop:State
2:100	iptc4xmpCore:CountryCode
2:101	photoshop:Country
2:103	photoshop:TransmissionReference
2:105	photoshop:Headline
2:110	photoshop:Credit
2:115	photoshop:Source
2:116	dc:rights["x-default"]
2:120	dc:description["x-default"]
2:122	photoshop:CaptionWriter

## TIFF and Exif tags for metadata

The IFDs used in TIFF and Exif can contain a large number of tags that are of interest as metadata. All but 5 of these tags are for individual metadata properties. The other 5 are for subsidiary IFDs or blocks of metadata in another format; these 5 tags must appear in the TIFF first (0th) IFD:

Tag	Hex	Usage
700	0x2BC	XMP
33723	0x83BB	IPTC DataSets
34377	0x8649	Photoshop image resources
34665	0x8769	Exif IFD offset
34853	0x8825	GPS IFD offset

For the most part, the tags in the TIFF 0th IFD are mapped to XMP properties in the `tiff:` namespace, and the tags in the Exif and GPS IFDs are mapped to XMP properties in the `exif:` namespace. In these cases the XMP property name is the tag name show below.

The following TIFF tags have mappings that do not follow the general rules. (The notation `ns:array["x-default"]` means the item in the `ns:array` language alternative whose `xml:lang` qualifier has the value "x-default".)

Tag	Name	XMP mapping
270	ImageDescription	dc:description["x-default"]

305	Software	xmp:CreatorTool
306	DateTime	xmp:ModifyDate
315	Artist	dc:creator[*]
33432	Copyright	dc:rights["x-default"]

ASCII tags in TIFF can have embedded NULLs to separate individual internal strings. Adobe practice is to process ASCII tags to the first NULL, dropping the rest.

- The individual strings in tag 315, Artist, should be mapped to separate items in the `dc:creator` array. Adobe applications map only to the first item of the `dc:creator` array on input, and write only the first item on output.
- Adobe applications map only the first string in tags 270, ImageDescription, and 33432, Copyright.

The TIFF specification also has this to say about tag 315, Artist:

*This tag records the name of the camera owner, photographer or image creator. The detailed format is not specified, but it is recommended that the information be written as in the example below for ease of Interoperability. When the field is left blank, it is treated as unknown. Ex. "Camera owner, John Smith; Photographer, Michael Brown; Image creator, Ken James"*

The XMP mapping does not attempt to recognize this informal separation of Artist names.

The TIFF specification has this to say about tag 33432, Copyright:

*In this standard the tag is used to indicate both the photographer and editor copyrights. It is the copyright notice of the person or organization claiming rights to the image. The Interoperability copyright statement including date and rights should be written in this field; for example, "Copyright, John Smith, 19xx. All rights reserved." In this standard the field records both the photographer and editor copyrights, with each recorded in a separate part of the statement. When there is a clear distinction between the photographer and editor copyrights, these are to be written in the order of photographer followed by editor copyright, separated by NULL (in this case, since the statement also ends with a NULL, there are two NULL codes) (see example 1). When only the photographer copyright is given, it is terminated by one NULL code (see example 2). When only the editor copyright is given, the photographer copyright part consists of one space followed by a terminating NULL code, then the editor copyright is given (see example 3). When the field is left blank, it is treated as unknown. Ex. 1) When both the photographer copyright and editor copyright are given. Photographer copyright + NULL[00.H] + editor copyright + NULL[00.H] Ex. 2) When only the photographer copyright is given. Photographer copyright + NULL[00.H] Ex. 3) When only the editor copyright is given. Space[20.H]+ NULL[00.H] + editor copyright + NULL[00.H]*

It is Adobe practice to process ASCII tags only to the first NULL, not recognizing the special rules for tag 33432. Adobe applications read to the first NULL and write a single logical string.

The following Exif and GPS tags are not mapped directly to XMP, but are combined with others:

Tag	Name	Combined with
37520	SubSecTime	306, tiff:DateTime
37521	SubSecTimeOriginal	36867, exif:DateTimeOriginal
37522	SubSecTimeDigitized	36868, exif:DateTimeDigitized

1	GPSLatitudeRef	2, exif:GPSLatitude
3	GPSLongitudeRef	4, exif:GPSLongitude
29	GPSDateStamp	7, exif:GPSTimeStamp
19	GPSDestLatitudeRef	20, exif:GPSDestLatitude
21	GPSDestLongitudeRef	22, exif:GPSDestLongitude

Most TIFF and Exif tags with text values are of type ASCII. A few have an UNDEFINED physical type, but actually contain generally well defined text with flexible encoding. The value includes a header defining the encoding of the subsequent text. These tags are:

37510	UserComment
27	GPSProcessingMethod
28	GPSAreaInformation

The encoding header is the first 8 bytes of the tag value, an ASCII string will NULL padding. The remainder of the tag value is the encoded text, without NULL termination. (And thus without the concept of multiple internal strings that ASCII tags have.) The encodings are:

"ASCII\0\0\0"	ISO 646, in essence 7-bit ASCII with national variants
"JIS\0\0\0\0\0"	JIS X208-1990
"UNICODE\0"	UTF-16 Unicode
"\0\0\0\0\0\0\0\0"	Undefined, implication of local OS encoding

The Exif specification does not mention whether the UTF-16 form should be big or little-endian. A reasonable presumption is to follow the ordering given in the TIFF header. The Exif specification says that UserComment should be used instead of ImageDescription for non-ASCII text. The XMP mapping does not follow this recommendation.

Most of the TIFF, Exif, and GPS tags that are mapped to XMP are considered read-only in the XMP. Edits to the XMP value should not be allowed, modified XMP values should not be written back to the native tags. The only writeback tags are:

270	TIFF ImageDescription
274	TIFF Orientation
282	TIFF XResolution
283	TIFF YResolution
296	TIFF ResolutionUnit
305	TIFF Software
306	TIFF DateTime

315	TIFF Artist
33432	TIFF Copyright
37510	Exif UserComment
40964	Exif RelatedSoundFile

## Value conversions for TIFF and Exif tag values

The value conversions for TIFF and Exif tags are mostly straightforward. Integer values are converted to decimal strings in the manner of "%d" in `C printf`. Rational numbers are converted to a *numerator/denominator* form; for example, the pair (123,456) becomes "123/456". Signed rationals are normalized to have only a leading minus, if any. The pair (123,-456) becomes "-123/456" and (-123/-456) becomes "123/456".

Tags of type ASCII are converted allowing for UTF-8 or local encoding. If a native string contains a valid UTF-8 sequence it is imported to XMP as-is. Otherwise, a local to UTF-8 conversion is performed. It is possible for a locally encoded value to look like UTF-8 and be erroneously imported as UTF-8, but this is very unlikely.

A date/time value is converted to ISO 8601 notation. The Exif date/time part is a 20-byte ASCII value formatted as "YYYY:MM:DD HH:MM:SS" with a terminating null. Any of the numeric portions can be blank if unknown. The fractional seconds are a null-terminated ASCII string with possible space padding. They are literally the fractional part, the digits that would be to the right of the decimal point.

GPS latitude and longitude values are combined with their "ref" part to produce a single XMP map coordinate.

- The native numeric portion is 3 rational numbers for degrees, minutes, and seconds. If all 3 denominators are 1, the numeric portion of the XMP is "deg,min,sec". Otherwise, floating-point arithmetic is used to normalize the coordinate to "deg,min.frac", with enough fractional minute digits to cover the largest of the 3 denominators.
- The ref portion in the native format is an ASCII letter for the English compass directions ('N', 'S', 'E', 'W') and is appended directly to the numeric portion in the XMP value. Assuming a ref of 'N', for example, the native format (12/1,34/1,56/1) becomes "12,34,56N" in XMP. The native format (12/1,34/1,56789/1000) becomes "12,34.946N".

## TIFF and Exif digests

The TIFF and Exif digests are in the XMP as `tiff:NativeDigest` and `exif:NativeDigest`. Each digest is a comma-separated list of reconciled tags, a semicolon, and an MD5 digest of the tag values. The tags should be added to the digest in the order of listing in tables 3, 4, 5, and 12 in the Exif 2.2 specification. The 16-byte digest is in the XMP as a 32-character hexadecimal string (using capital A-F).

## TIFF and Exif tag tables

Tables 14 through 16 in the Exif 2.2 specification show by implication how the various TIFF and Exif tags are divided among the 0th, Exif, and GPS IFDs. Neither the TIFF nor Exif specifications specifically state that this separation is mandatory. Nor do they state whether tags must be unique, although the definitions of almost all tags seem to imply that repeated tags should not be allowed. The TIFF specification does state that tags within an IFD must be in ascending numerical order.

Adobe practice is to adopt a “partially relaxed” policy towards TIFF and Exif tags. It is strict in only accepting tags from their proper IFD as defined by tables 14 through 16 in the Exif 2.2 specification. Tags in the wrong IFD are ignored. The policy allows tags within an IFD to be out of order on input, but always writes them in correct order. Repeated input tags are tolerated, the last encountered one is kept.

The outer TIFF 0th IFD may contain the following tags that are of interest as metadata. The following table does not list tags 34665 and 34853, the Exif and GPS IFD offsets. Those tags are not themselves of interest as metadata, only as a means to find their respective IFDs. Except as noted above, the tags listed below are all mapped to the XMP `tiff: namespace, "http://ns.adobe.com/tiff/1.0/":`

Tag	Hex	Type	Count	Name
256	0x100	SHORT or LONG	1	ImageWidth
257	0x101	SHORT or LONG	1	ImageLength
258	0x102	SHORT	3	BitsPerSample
259	0x103	SHORT	1	Compression
262	0x106	SHORT	1	PhotometricInterpretation
270	0x10E	ASCII	Any	ImageDescription
271	0x10F	ASCII	Any	Make
272	0x110	ASCII	Any	Model
274	0x112	SHORT	1	Orientation
282	0x11A	RATIONAL	1	XResolution
283	0x11B	RATIONAL	1	YResolution
284	0x11C	SHORT	1	PlanarConfiguration
296	0x128	SHORT	1	ResolutionUnit
301	0x12D	SHORT	3*256	TransferFunction
305	0x131	ASCII	Any	Software
306	0x132	ASCII	20	DateTime
315	0x13B	ASCII	Any	Artist
318	0x13E	RATIONAL	2	WhitePoint
319	0x13F	RATIONAL	6	PrimaryChromaticities
529	0x211	RATIONAL	3	YCbCrCoefficients
530	0x212	SHORT	2	YCbCrSubSampling
531	0x213	SHORT	1	YCbCrPositioning
532	0x214	RATIONAL	6	ReferenceBlackWhite
33432	0x8298	ASCII	Any	Copyright

The inner Exif IFD may contain the following tags that are of interest as metadata. Except as noted above, these are all mapped to the XMP `exif:` namespace, "`http://ns.adobe.com/exif/1.0/`".

Tag	Hex	Type	Count	Name
33434	0x829A	RATIONAL	1	ExposureTime
33437	0x829D	RATIONAL	1	FNumber
34850	0x8822	SHORT	1	ExposureProgram
34852	0x8824	ASCII	Any	SpectralSensitivity
34855	0x8827	SHORT	Any	ISOSpeedRatings
34856	0x8828	UNDEFINED	Any	OECF
36864	0x9000	UNDEFINED	4	ExifVersion
36867	0x9003	ASCII	20	DateTimeOriginal
36868	0x9004	ASCII	20	DateTimeDigitized
37121	0x9101	UNDEFINED	4	ComponentsConfiguration
37122	0x9102	RATIONAL	1	CompressedBitsPerPixel
37377	0x9201	SRATIONAL	1	ShutterSpeedValue
37378	0x9202	RATIONAL	1	ApertureValue
37379	0x9203	SRATIONAL	1	BrightnessValue
37380	0x9204	SRATIONAL	1	ExposureBiasValue
37381	0x9205	RATIONAL	1	MaxApertureValue
37382	0x9206	RATIONAL	1	SubjectDistance
37383	0x9207	SHORT	1	MeteringMode
37384	0x9208	SHORT	1	LightSource
37385	0x9209	SHORT	1	Flash
37386	0x920A	RATIONAL	1	FocalLength
37396	0x9214	SHORT	2..4	SubjectArea
37510	0x9286	UNDEFINED	Any	UserComment
40960	0xA000	UNDEFINED	4	FlashpixVersion
40961	0xA001	SHORT	1	ColorSpace
40962	0xA002	SHORT or LONG	1	PixelXDimension
40963	0xA003	SHORT or LONG	1	PixelYDimension
40964	0xA004	ASCII	13	RelatedSoundFile
41483	0xA20B	RATIONAL	1	FlashEnergy

Tag	Hex	Type	Count	Name
41484	0xA20C	UNDEFINED	Any	SpatialFrequencyResponse
41486	0xA20E	RATIONAL	1	FocalPlaneXResolution
41487	0xA20F	RATIONAL	1	FocalPlaneYResolution
41488	0xA210	SHORT	1	FocalPlaneResolutionUnit
41492	0xA214	SHORT	2	SubjectLocation
41493	0xA215	RATIONAL	1	ExposureIndex
41495	0xA217	SHORT	1	SensingMethod
41728	0xA300	UNDEFINED	1	FileSource
41729	0xA301	UNDEFINED	1	SceneType
41730	0xA302	UNDEFINED	Any	CFAPattern
41985	0xA401	SHORT	1	CustomRendered
41986	0xA402	SHORT	1	ExposureMode
41987	0xA403	SHORT	1	WhiteBalance
41988	0xA404	RATIONAL	1	DigitalZoomRatio
41989	0xA405	SHORT	1	FocalLengthIn35mmFilm
41990	0xA406	SHORT	1	SceneCaptureType
41991	0xA407	RATIONAL	1	GainControl
41992	0xA408	SHORT	1	Contrast
41993	0xA409	SHORT	1	Saturation
41994	0xA40A	SHORT	1	Sharpness
41995	0xA40B	UNDEFINED	Any	DeviceSettingDescription
41996	0xA40C	SHORT	1	SubjectDistanceRange
42016	0xA420	ASCII	33	ImageUniqueID

The inner GPS IFD may contain the following tags that are of interest as metadata. Except as noted above, these are all mapped to the XMP `exif:` namespace, "`http://ns.adobe.com/exif/1.0/`".

Tag	Hex	Type	Count	Name
0	0x0	BYTE	4	GPSVersionID
2	0x2	RATIONAL	3	GPSLatitude
4	0x4	RATIONAL	3	GPSLongitude
5	0x5	BYTE	1	GPSAltitudeRef

Tag	Hex	Type	Count	Name
6	0x6	RATIONAL	1	GPSAltitude
7	0x7	RATIONAL	3	GPSTimeStamp
8	0x8	ASCII	Any	GPSSatellites
9	0x9	ASCII	2	GPSStatus
10	0xA	ASCII	2	GPSMeasureMode
11	0xB	RATIONAL	1	GPSDOP
12	0xC	ASCII	2	GPSSpeedRef
13	0xD	RATIONAL	1	GPSSpeed
14	0xE	ASCII	2	GPSTrackRef
15	0xF	RATIONAL	1	GPSTrack
16	0x10	ASCII	2	GPSTrackRef
17	0x11	RATIONAL	1	GPSTrack
18	0x12	ASCII	Any	GPSTrackRef
20	0x14	RATIONAL	3	GPSDestLatitude
22	0x16	RATIONAL	3	GPSDestLongitude
23	0x17	ASCII	2	GPSDestBearingRef
24	0x18	RATIONAL	1	GPSDestBearing
25	0x19	ASCII	2	GPSDestDistanceRef
26	0x1A	RATIONAL	1	GPSDestDistance
27	0x1B	UNDEFINED	Any	GPSProcessingMethod
28	0x1C	UNDEFINED	Any	GPSAreaInformation
30	0x1E	SHORT	1	GPSDifferential

## Metadata storage

The intersection of file formats and metadata forms has confusing inversions, with portions of metadata being wrapped and stored in a variety of ways in the file formats. For example, a TIFF file contains a block of Photoshop image resources as the value of a TIFF tag, while a Photoshop file contains a TIFF header plus IFDs as the value of an image resource. This section will outline metadata storage organized by file type and by metadata type.

### Storage by metadata form

TIFF tags for individual properties, including TIFF/Exif, can be found in these locations:

File format	Storage location
Photoshop	TIFF within image resource 1058
TIFF	Native IFDs
JPEG	TIFF within Exif APP1 marker segment

**PHOTOSHOP NOTE:** Photoshop 6 split the tags in a TIFF file between proper native tags and “buried” tags in image resource 1058 within tag 34377. See [“Photoshop 6 and TIFF” on page 84](#).

Photoshop image resources for individual properties can be found in these locations:

File format	Storage location
Photoshop	Native image resource section
TIFF	Tag 34377 in the 0th IFD
JPEG	Photoshop APP13 marker segment

IPTC DataSets can be found in these locations:

File format	Storage location
Photoshop	Photoshop image resource 1028
TIFF	Tag 33723 in the 0th IFD; Photoshop image resource 1028
JPEG	Photoshop image resource 1028
Any Mac	Mac OS file resource 'ANPA' 10000 (prior to Photoshop 9)

XMP can be found in these locations:

File format	Storage location
Photoshop	Photoshop image resource 1060

File format	Storage location
TIFF	TIFF tag 700
JPEG	XMP APP1 marker segment

Additionally, keywords and description can be found in Mac OS file resources:

File format	Storage location
Any Mac OS	Mac OS file resources from 'pnot' 0 items KeyW and Desc

Reading and writing of the Mac OS ANPA 10000 resource and the pnot 0 resource's KeyW and Desc items was dropped in Photoshop 9.

## Metadata storage in JPEG files

JPEG files, including Exif JPEG, use a mixture of native marker segments, TIFF tags, and Photoshop image resources. Metadata can be found in 3 APPn marker segments:

Marker	Signature, including NULLs	Usage
APP1	"Exif\0\0"	TIFF and Exif (2 NULLs)
APP1	"http://ns.adobe.com/xap/1.0/\0"	XMP
APP13	"Photoshop 3.0\0"	Photoshop image resources

The TIFF in the Exif APP1 marker segment should not contain tag 700 (XMP), tag 33723 (IPTC), or tag 34377 (Photoshop image resources). There should be only one copy of the IPTC in a JPEG file, in Photoshop image resource 1028 (ignoring the possible Mac OS ANPA 10000 resource).

The Photoshop image resources for metadata that can be used in JPEG are shown below. Other non-metadata image resources might also be present. Image resources 1058 (TIFF) and 1060 (XMP) should not be used in JPEG files.

ID	Hex	Description
1028	0x404	IPTC DataSets
1034	0x40A	Copyright flag as a 0/1 Boolean
1035	0x40B	Text string for a copyright information URL
1061	0x425	MD5 digest of the IPTC data

## Metadata storage in Photoshop files

Photoshop files use Photoshop image resources as the metadata storage mechanism. The metadata image resources were listed earlier, and are repeated below. Other non-metadata image resources might also be present.

ID	Hex	Description
1028	0x404	IPTC DataSets
1034	0x40A	Copyright flag as a 0/1 Boolean
1035	0x40B	Text string for a copyright information URL
1058	0x422	Exif metadata
1060	0x424	XMP
1061	0x425	MD5 digest of the IPTC DataSets

The TIFF in image resource 1058 can contain a full set of the TIFF metadata tags for individual properties, including use of the Exif and GPS subsidiary IFDs. The TIFF in a Photoshop file should not contain XMP or IPTC (TIFF tags 700 or 33723). Or of course nested copies of image resources (TIFF tag 34377).

Some very old versions of Photoshop wrote image resource 1008 or 1020, but not both. These are now considered obsolete, both were dropped prior to Photoshop 6. The current Photoshop SDK does not even contain type information for 1020.

ID	Hex	Description
1008	0x3F0	Caption as a Pascal string
1020	0x3FC	Caption as a plain string

## Metadata storage in TIFF files

As mentioned earlier, the IFDs used in TIFF and Exif contain a large number of tags that are of interest as metadata. All but 5 of these tags are for individual metadata properties. The other 5 are for contained blocks of metadata:

Tag	Hex	Usage
700	0x2BC	XMP
33723	0x83BB	IPTC DataSets
34377	0x8649	Photoshop image resources
34665	0x8769	Exif IFD offset
34853	0x8825	GPS IFD offset

The TIFF physical type (BYTE, SHORT, LONG, etc.) must be taken into account when processing the XMP, IPTC, and image resources. There may be appended 0x00 bytes to make the total size be a multiple of the

physical unit. Software loops processing the data must be prepared to terminate 1 to 3 bytes early, depending on the physical unit.

The Photoshop image resources for metadata that can be used in TIFF are shown below. A TIFF file should not contain image resource 1060 (XMP). A TIFF file might contain image resource 1058 (TIFF) because of the Photoshop 6 feature/bug mentioned below. Other non-metadata image resources might also be present.

ID	Hex	Description
1028	0x404	IPTC DataSets
1034	0x40A	A Copyright flag as a 0/1 Boolean
1035	0x40B	Text string for a copyright information URL
1061	0x425	MD5 digest of the IPTC DataSets

**NOTE:** TIFF files contain 2 copies of the IPTC: in the top level TIFF tag 33723, and in image resource 1028 within TIFF tag 34377. Photoshop 6, Photoshop 7, and Photoshop 9 all do this. There is also a possible third copy of the IPTC in the Mac OS ANPA 10000 resource.

## Photoshop 6 and TIFF

Photoshop 6 wrote TIFF in an awkward and nonintuitive manner. Photoshop 6 placed most of the TIFF/Exif information within image resource 1058 instead of natively in the "real" TIFF file IFDs. This includes the Exif subsidiary IFD, tag 34665 is only written in the 0th IFD within image resource 1058. In addition, tags 259, 270, 282, and 296 were written in both the outer native 0th IFD and in the 0th IFD within image resource 1058. Several tags are also duplicated between the 0th and 1st IFDs image resource 1058. But since that is duplication between the main image and thumbnail and the 1st IFD contains a proper subset of the tags in the 0th IFD, those in the 1st IFD can be ignored.

Photoshop 7 and Photoshop 9 do not write image resource 1058 in a TIFF file.

Here is a sketch of the content of a TIFF example from various versions of Photoshop:

- Photoshop 6 - big-endian TIFF
  - IFD 0 - 256, 257, 258, 259, 262, 270, 282, 283, 284, 296, 33723 (IPTC), 34377 (PSIR)
    - Tag 34377 PSIR - 1028 (IPTC), 1034, 1035, 1058 (Exif)
    - 1058 - Exif as little-endian TIFF
      - IFD 0 - 270, 271, 272, 274, 282, 283, 296, 306, 531, 34665 (Exif)
      - IFD 1 - 259, 271, 272, 274, 282, 283, 296, 306
      - Exif IFD - 27 typical tags
- Photoshop 7 - big-endian TIFF
  - IFD 0 - 256, 257, 258, 259, 262, 270, 271, 272, 282, 283, 284, 296, 305, 306, 315, 700 (XMP), 33723 (IPTC), 34377 (PSIR), 34665 (Exif)
  - Exif IFD - 25 typical tags

- Tag 34377 PSIR - 1028 (IPTC), 1034, 1035, 1061
- Photoshop 9 - big-endian TIFF
  - IFD 0 - 256, 257, 258, 259, 262, 270, 271, 272, 282, 283, 284, 296, 305, 306, 315, 700 (XMP), 33432, 33723 (IPTC), 34377 (PSIR), 34665 (Exif)
  - Exif IFD - 26 typical tags
  - Tag 34377 PSIR - 1028 (IPTC), 1034, 1035, 1061

## Metadata storage in Mac OS file resources

Mac OS versions of Photoshop through Photoshop CS also wrote some metadata in Mac OS file resources. These resources are not written by Photoshop CS2 or later. The resources are:

ANPA	10000	IPTC metadata (this takes priority for reconciliation of multiply-defined values; see <a href="#">"Reconciliation issues" on page 64</a> )
pnot	0	Keywords and description

The value of the ANPA 10000 resource is a sequence of IPTC DataSets, as previously described.

The `pnot` resource was created by Apple for annotation information as part of QuickTime. Its structure is shown below. Only the `KeyW` and `Desc` additional items are of interest for metadata.

Offset	Length	Description
0	4	Modification timestamp as a 32-bit big-endian Mac OS time
4	2	Version number, 16-bit big-endian integer
6	4	Resource type for preview
10	2	Resource ID for preview, 16-bit big-endian integer
12	2	Number of additional items, 16-bit big-endian integer
14	-	Sequence of additional items

Each additional item has the structure shown below. Only the associated resource for the `KeyW` and `Desc` items is of interest for metadata.

Offset	Length	Description
0	4	Modification timestamp as a 32-bit big-endian Mac OS time
4	4	OSType defining the usage of this item
8	4	Resource type for the associated data
12	2	Resource ID for the associated data, 16-bit big-endian integer

Offset	Length	Description
14	2	Region code for this item, 16-bit big-endian integer
16	4	Reserved

The items whose usage is `KeyW` or `Desc` are of interest for metadata. The `KeyW` item should have an associated `STR#` resource that contains the keywords. A `STR#` resource contains a 16-bit big-endian count of strings followed by a sequence of Pascal strings. The `Desc` item should have an associated `TEXT` resource that contains the description. A `TEXT` resource simply contains text.