BRIEFING PAPER: THE ADOBE EXTENSIBLE METADATA PLATFORM (XMP)

ALEX BALL

kim11rep015ab10.pdf

ACCESS LEVEL: 1

ISSUE DATE: 22 FEBRUARY 2007

APPROVED BY: MANSUR DARLINGTON

DATE APPROVED: 15 MARCH 2007

1 INTRODUCTION

The Adobe eXtensible Metadata Platform (XMP) was introduced to Adobe products to solve the problem of embedding a wide variety of metadata into various different file formats using a common approach [Adobe 2001]. The approach chosen involves the use of the World Wide Web Consortium's Resource Description Framework (RDF) in XML format as the vehicle for expressing metadata, coupled with some proprietary wrapper tags and custom schemata [Adobe 2005*b*].

This briefing paper looks at the nature of XMP, its possibilities and limitations.

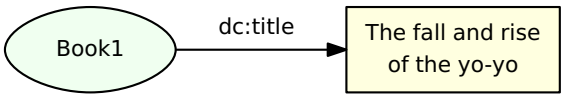
2 THE RESOURCE DESCRIPTION FRAMEWORK

This section provides a short introduction to RDF, in particular RDF/XML, as a means of expressing metadata [W3C 2004].

RDF is based on the principle of representing the properties of objects, and the relationships between objects, as simple triples of data: a *subject* (the resource being described), a *predicate* (a defined relationship or property) and an *object* (the value of the property, or the related resource). For example, a book may be described by the following triple:

Subject: Book1	Predicate: `dc:title`	Object: "The fall and rise of the yo-yo"

The predicate `dc:title` is an abbreviation of ⟨`http://purl.org/dc/elements/1.1/title`⟩, which is the URI of its definition. This triple may be represented using an RDF diagram:



where oval containers represent *resources* (things), rectangular containers represent *literals* (strings, numbers, etc.) and arrows represent predicates. The same triple can also be represented using RDF/XML:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>The fall and rise of the yo-yo</dc:title>
</rdf:Description>
...
```
(1)

RDF allows for literals to be given a type to enable disambiguation; for example, "1001" could be a decimal integer, a binary integer, a year or a string. In many cases, the type can be derived from the definition of the predicate, but in other cases it can be given explicitly as follows.

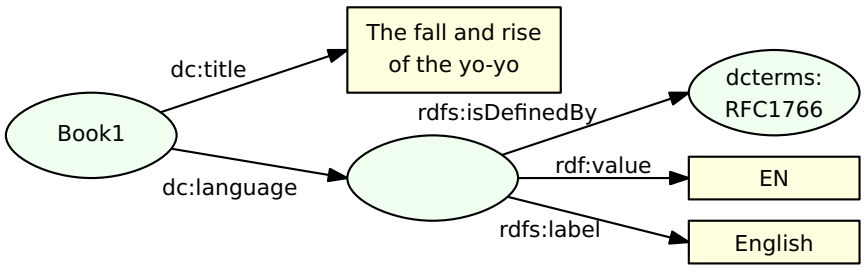
Subject: Book1	Predicate: `dc:date`	Object: "1989"^^`xsd:date`

would be written in RDF/XML as:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1989</dc:date>
</rdf:Description>
...
```
(2)

This form of expression allows the range of available data types to be fully extensible. The XML Schema namespace used here provides nineteen primitive data types (including `string`, `boolean`, `decimal`, `duration`, `time`, `date`) and twenty-five derived data types (including `integer`, `byte`, `language`). RDF itself defines only one datatype, `XMLLiteral`, although using this directly involves having to canonicalize the XML fragment according to the XML Exclusive Canonicalization recommendation [W3C 2002]. The easier way of including XML fragments is to use `rdf:parseType="Literal"` instead of the `rdf:datatype` attribute, in which case the XML can be included without being canonicalized.

Various levels of complexity are achievable simply by allowing resources to appear in as many triples as needed, as either subject or object. RDF also permits anonymous resources — resources without URIs to identify them — to express situations where resources are identified solely by their properties; such resources are known as *blank nodes*. An example of a blank node follows:



RDF/XML provides several different ways of expressing this, for example using internal identifiers:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>The fall and rise of the yo-yo</dc:title>
  <dc:language rdf:nodeID="foo"/>
</rdf:Description>
<rdf:Description rdf:nodeID="foo" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:value>EN</rdf:value><rdfs:label>English</rdfs:label>
  <rdfs:isDefinedBy rdf:resource="http://purl.org/dc/terms/RFC1766"/>
</rdf:Description>
...
```

(3)

or nested `rdf:Description` elements:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>The fall and rise of the yo-yo</dc:title>
  <dc:language>
    <rdf:Description xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
      <rdf:value>EN</rdf:value><rdfs:label>English</rdfs:label>
      <rdfs:isDefinedBy rdf:resource="http://purl.org/dc/terms/RFC1766"/>
    </rdf:Description>
  </dc:language>
</rdf:Description>
...
```

(4)

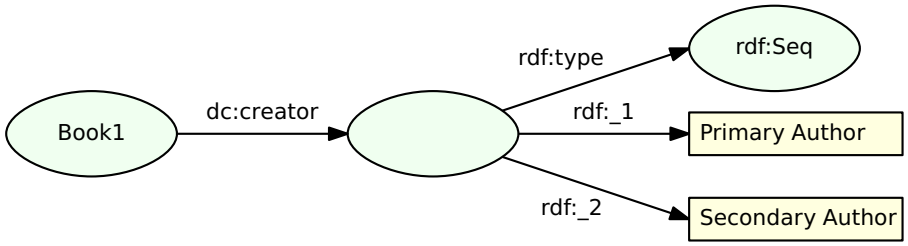
or even a property-and-node element:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/"
                 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

(5)

```
  <dc:title>The fall and rise of the yo-yo</dc:title>
  <dc:language rdf:parseType="Resource">
    <rdf:value>EN</rdf:value><rdfs:label>English</rdfs:label>
    <rdfs:isDefinedBy rdf:resource="http://purl.org/dc/terms/RFC1766"/>
  </dc:language>
</rdf:Description>
...
```

One special use of blank nodes in RDF is to handle situations where the a subject is linked to several objects by the same predicate. These objects can be collected together by a blank node that represents a list, the properties of which can be further refined. RDF provides two types of list: non-exhaustive lists, known as *containers*, and exhaustive lists, known as *collections*. The three types of container provided by RDF are `rdf:Alt`, `rdf:Bag` and `rdf:Seq`; these imply, respectively, that the list items are alternative expressions of the same thing, an unordered assortment of different things, or a sequence of different things. For example, a primary and secondary author can be listed as follows on an RDF diagram:



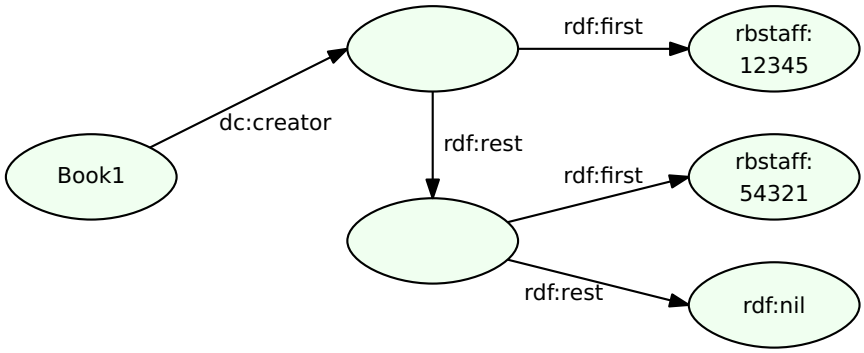
or like this in RDF/XML:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator>
    <rdf:Seq>
      <rdf:li>Primary Author</rdf:li>
      <rdf:li>Secondary Author</rdf:li>
    </rdf:Seq>
  </dc:creator>
</rdf:Description>
...
```

(6)

There are a few quirks of RDF/XML evident in this example. The first is the `rdf:Seq` element which is an example of a typed node element: an abbreviation of an `rdf:Description` element and the `rdf:type` element contained within. The second is that `rdf:li` elements have been used to generate automatically the `rdf:_1` and `rdf:_2` predicates.

To imply that no other items belong to the list, collection syntax can be used. This syntax does not distinguish between ordered and unordered lists.



This may be expressed literally in RDF/XML:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator rdf:parseType="Resource">
    <rdf:first rdf:resource="http://www.redbrick.ac.uk/staffid/12345"/>
    <rdf:rest rdf:parseType="Resource">
      <rdf:first rdf:resource="http://www.redbrick.ac.uk/staffid/54321"/>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    </rdf:rest>
  </dc:creator>
</rdf:Description>
...
```

(7)

but RDF/XML defines a compact way of expressing collections:

```
...
<rdf:Description rdf:about="#Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator rdf:parseType="Collection">
    <rdf:Description rdf:about="http://www.redbrick.ac.uk/staffid/12345"/>
    <rdf:Description rdf:about="http://www.redbrick.ac.uk/staffid/54321"/>
  </dc:creator>
</rdf:Description>
...
```

(8)

One other sophisticated use of RDF is the ability to provide RDF statements about RDF statements. The long way of doing this is to reproduce the statement using special tags (`rdf:Statement`, `rdf:subject`, `rdf:predicate`, and `rdf:object`) and add in appropriate metadata:

```
...
<rdf:Description rdf:ID="Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>The fall and rise of the yo-yo</dc:title>
</rdf:Description>
<rdf:Statement rdf:about="#triple1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:subject rdf:resource="#Book1"/>
  <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/title"/>
  <rdf:object>The fall and rise of the yo-yo</rdf:object>
  <dc:creator>A Cataloguer</dc:title>
</rdf:Statement>
...
```

(9)

A more efficient and compact way of doing this to assign an `rdf:ID` directly to the statement when it first appears:

```
...
```

(10)

```
<rdf:Description rdf:ID="Book1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title rdf:ID="triple1">The fall and rise of the yo-yo</dc:title>
</rdf:Description>
<rdf:Description rdf:about="#triple1" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:creator>A Cataloguer</dc:title>
</rdf:Description>
...
```

The correct XML wrapper for RDF/XML statements is the `rdf:RDF` element. This element may be omitted if there is only one top-level node element, but it is recommended to include it anyway. Thus a full RDF/XML document would look like this:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about="#Book1">
    <dc:title>The fall and rise of the yo-yo</dc:title>
  </rdf:Description>
</rdf:RDF>
```

3 THE XMP STORAGE MODEL

XMP metadata is normally encoded as an XMP packet, to be embedded in any file format that supports embedded metadata files. Packets are demarcated using proprietary prologue and epilogue tags.

```
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
...
<?xpacket end="r"?>
```

The content of the `begin` attribute defines the encoding and byte order for the packet. If left empty, the attribute specifies UTF-8; otherwise the attribute must contain the Unicode zero-width non-breaking space character (U+FEFF), as this disambiguates the various Unicode encodings and byte orders. In UTF-8, U+FEFF is encoded as 0xEF 0xBB 0xBF, which when interpreted as ISO/IEC 8859-1 encoding becomes 'ï»¿' [ISO/IEC 10646:2003; RFC 2781; RFC 3629].

The content of the `id` attribute is a text string that may be used as a differentiation mechanism should the XMP specification be altered in future.

The content of the `end` attribute declares whether the packet can be edited in situ without changing its size (`end="w"`) or not (`end="r"`). When the packet is declared editable, sufficient whitespace padding should be included at the end of the packet to allow for the growth of the packet contents.

These packet tags are required except when the XMP metadata is to be embedded in an XML file, in which case they must be omitted. In the special case that the XML file is empty except for the XMP metadata, it is recommended that the file be given a `.xmp` extension and be served with an `application/rdf+xml` MIME type.

The XML element used for wrapping XMP metadata is the `xmpmeta` element. While this is optional, its use is recommended when inserting XMP metadata into XML streams, to mark the contents as being XMP's simplified RDF instead of pure RDF.

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

4 USING RDF IN XMP

The main body of an XMP packet or stream is written in RDF/XML, and hence begins and ends with matching `rdf:RDF` tags. In contrast to pure RDF/XML, these tags are always required.

The limitations of Adobe's XMP readers and writers have lead to simplifications in the XMP implementation of RDF/XML. The documented limitations are as follows:

- The top level node elements must be `rdf:Description` elements, not typed nodes or `rdf:Statement` elements.
- All top level `rdf:Description` elements must have the same (possibly blank) value for their `rdf:about` attributes.
- The attributes `rdf:ID` and `rdf:nodeID` are stripped out silently.
- The attributes `rdf:aboutEach` and `rdf:aboutEachPrefix` are not supported at all and cause errors.
- The attribute `rdf:parseType="Literal"` is not supported, meaning that XML snippets can only be included as the object of RDF triples in XMP using the `rdf:datatype="... XMLLiteral"` method.

Additionally, experimentation with Adobe tools shows that support for typed nodes below the top level is limited to a few common types such as `rdf:Alt`, `rdf:Bag` and `rdf:Seq`. It also appears that the attribute `rdf:parseType="Collection"` is not supported, meaning constructions like extract (8) cannot be used, although an alternative expression such as extract (7) is permitted. Use of the attribute `rdf:datatype` appears to disable XMP reading and writing in Acrobat 7, allowing an XMP packet to pass through document editing unchanged.

In consequence, several features of RDF are disabled in XMP. Blank nodes are expressed by default in the manner of extract (5), although constructions like extract (4) are permitted; constructions like extract (3) are forbidden. It is not possible to provide any form of meta-metadata as constructions like extracts (9) and (10) are forbidden.

Further limitations are imposed for RDF schemata that Adobe itself controls, including ones implementing the EXIF metadata standard for images [JEITA CP-3451], and also for Dublin Core. While Dublin Core itself is highly flexible, the XMP specification requires that it be used in a more regimented fashion.

- The elements `dc:description`, `dc:rights` and `dc:title` must contain an `rdf:Alt` list, with each list item having an `xml:lang` attribute with a value drawn from the language codes set out in RFC 3066. The default (main) list item has the attribute `xml:lang="x-default"`, duplicating another list item if necessary.
- The elements `dc:contributor`, `dc:language`, `dc:publisher`, `dc:relation`, `dc:subject` and `dc:type` must contain an `rdf:Bag` list.
- The elements `dc:creator` and `dc:date` must contain an `rdf:Seq` list.
- Dates must be formatted according to the guidelines set out in the W3C profile of ISO 8601, e.g. `2004-10-23T18:00:00Z`.
- The element `dc:format` must contain a MIME type as defined in RFC 2046.
- The element `dc:language` must list languages using the language codes set out in RFC 3066.

5 USING XMP IN ADOBE APPLICATIONS

While Adobe applications can process XMP metadata without intervention, most also provide some form of interface allowing users to view and edit XMP metadata through the application.

Adobe Photoshop v7 provides a file information dialogue box with five pre-set panels: General, Keywords, Categories, Origins and EXIF (see figure 1). The fields in these panels interact with the entries in the XMP packet. Tools are also provided for importing, appending and exporting XMP information. The dialogue box also provides the option of preserving or removing any XMP information not displayed by the panels.

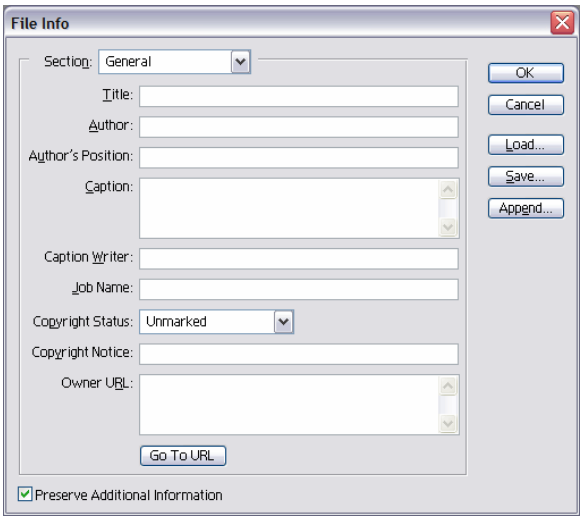


Figure 1: File information panel in Adobe Photoshop v7

Adobe Acrobat v6+ and other more recent Adobe applications use a more sophisticated interface. In Acrobat this is hidden behind the normal document properties dialogue box (see figure 2). The advanced metadata display comes with one default file information panel ('Description', displayed first) and an advanced XMP metadata display panel (see figure 3). It is also possible to add custom file information panels using XML files saved to a specific location (see figure 4).

Unfortunately, the syntax used for designing file information panels places additional restrictions on complexity of metadata that can be expressed [Adobe 2005*a*]. Panel fields can only read or write RDF triples that have the document in question as the subject and a plain (untyped) literal, resource or RDF collection of list items as the object. Thus of the RDF extracts shown in this document, only extracts (1) and (6) can be created.

The default file information panel, 'Description', has a number of important fields on it that contribute to six different XMP schemata. Table 1 shows which panel fields map on to which statements in these six schemata. Thus, if any of the RDF statements read by that panel contain complex RDF constructs, they will be ignored and overwritten in memory by empty RDF statements, even if the complex RDF constructs are supported by the XMP specification. The consequence of this being the default panel is that *any* attempt to view the XMP information through the user interface, let alone edit it, will corrupt it if any of the panel's fields map onto a complex RDF statement. Of course, this corruption is not committed to the file unless the interface is exited with the OK buttons and the file is saved.

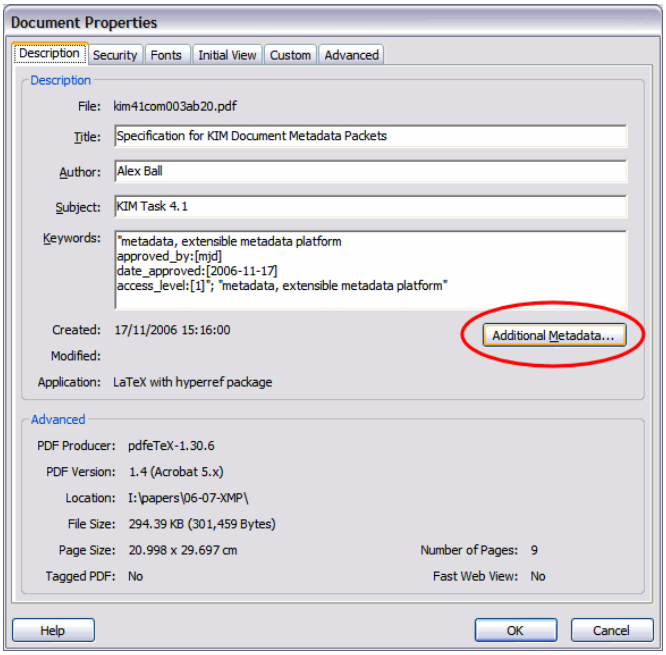


Figure 2: Document properties dialogue box in Adobe Acrobat v7

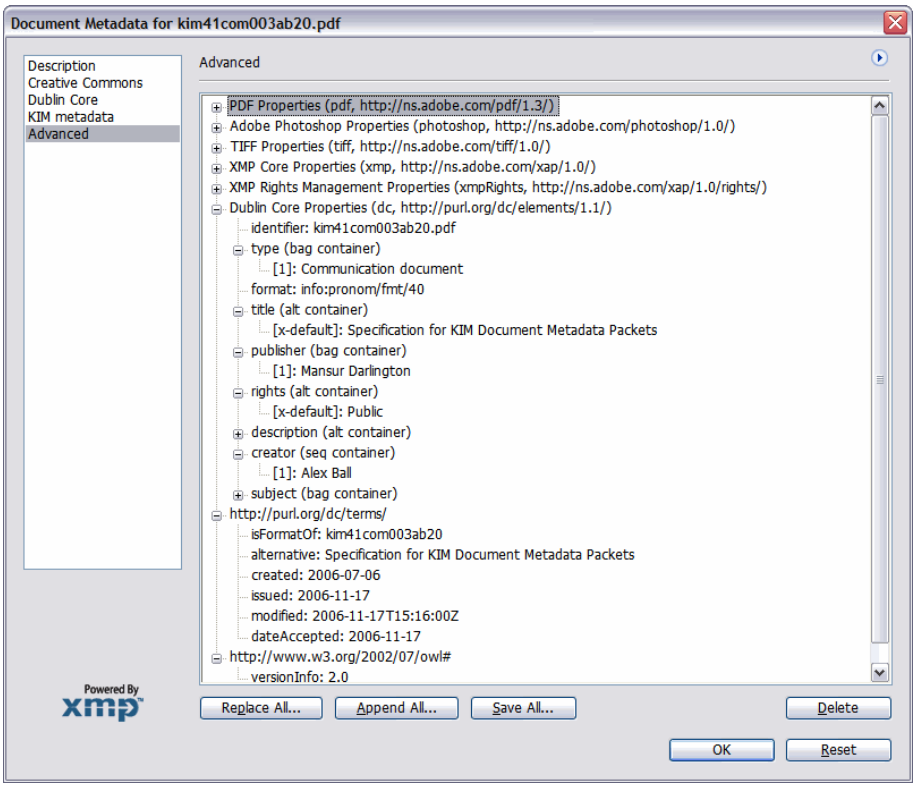


Figure 3: Advanced XMP metadata display panel in Adobe Acrobat v7

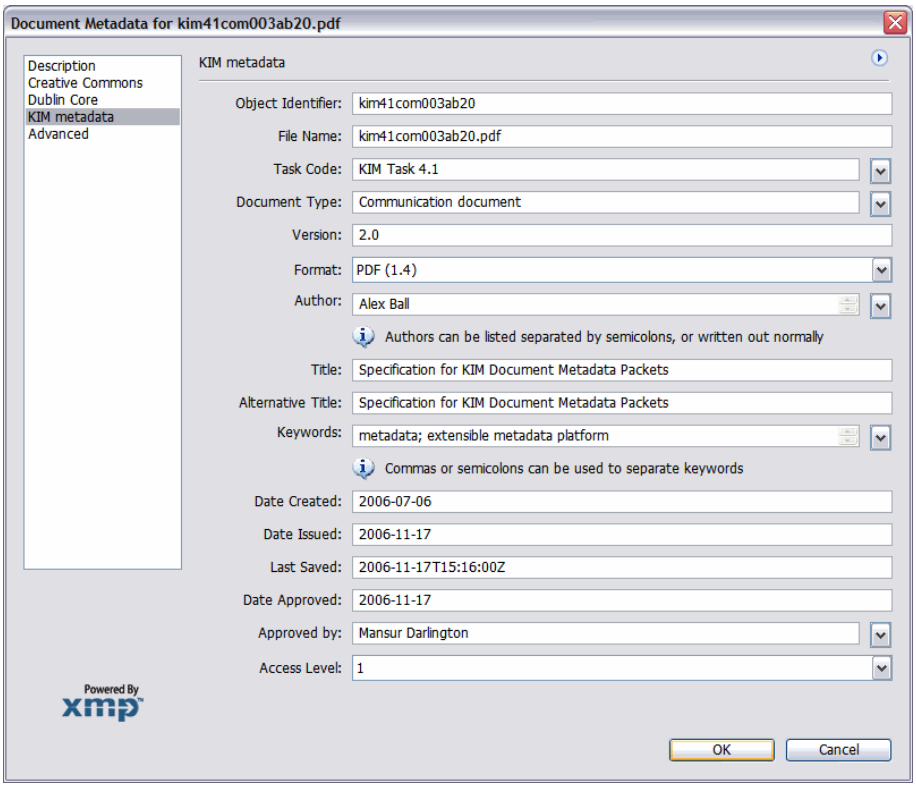


Figure 4: Custom file information panel in Adobe Acrobat v7

Table 1: Mappings between the default Description panel and built-in XMP schemata

Panel field	PDF	Photoshop	TIFF	XMP Core	XMP Rights	Dublin Core
Document Title	Title	Title		Title *(alt)*		Title *(alt)*
Author	Author	Author	Artist	Authors *(seq)*, Author		Creator *(seq)*
Description	Subject	Caption	Image Description	Description *(alt)*		Description *(alt)*
Description Writer		CaptionWriter				
Keywords		Keywords *(bag)*		Keywords *(bag)*		Subject *(bag)*
Copyright Status		Marked			Marked	
Copyright Notice		Copyright	Copyright *(alt)*		Copyright *(alt)*	Rights *(alt)*
Copyright Info URL		WebStatement			WebStatement	
Created	CreationDate			CreateDate		
Modified	ModDate		DateTime	ModifyDate		
Application	Creator		Software	CreatorTool		
Format				Format		Format

The advanced XMP metadata display panel (figure 3) is more expressive, as it can handle arbitrary blank nodes, and thus can be used to view the full range of RDF statements that XMP can handle. Of course, this is only of benefit for RDF statements not already rewritten by the Description panel and any other panels viewed.

6 USING XMP IN OTHER APPLICATIONS

Support for XMP outside of Adobe applications is most obvious in areas where there is a distinct advantage to hiding metadata within files, as opposed to displaying metadata in the rendered file or keeping separate registers of metadata. A number of different content management systems and digital asset management systems now use XMP, for example EMC Documentum, IBM DB2 and MediaBeacon R3volution [Adobe 2007]. Support is particularly strong for digital images, though mainly in dedicated image management tools rather than image editors. Of particular note are the free Microsoft Photo Info plugin for Windows,[1] and the Imagero Java library for reading image files.[2] Support for using XMP directly in text documents is still poor outside Adobe tools, with no explicit XMP support for Microsoft Office or OpenOffice.org.

One of the few ways of using XMP with text documents is through LaTeX. LaTeX is a document typesetting language built on the TeX typesetting engine. There are currently two LaTeX packages available on the Comprehensive TeX Archive Network (CTAN) that allow XMP packets to be embedded in PDF documents: xmpincl and hyperxmp. The xmpincl package[3] allows a previously composed XMP packet to be embedded in a PDF using pdfLaTeX. The hyperxmp package,[4] on the other hand, collects information about the author, title, subject, keywords, copyright, and licence URL of a document through a mixture of its own commands and those provided by the package hyperref — commonly used for enhancing PDF documents created using LaTeX — and both constructs and embeds an appropriate XMP packet. This latter package not only works with pdfLaTeX but also with standard LaTeX coupled with the utility dvipdfm.

Ghostscript is popular utility for interpreting and writing PostScript and PDF files.[5] When converting PostScript documents to PDFs of version 1.4+, Ghostscript automatically creates an XMP packet that records its involvement and any suitable information (e.g. title, author) provided in the metadata at the start of the PostScript file.

While editing and embedding XMP is poorly supported in the main, reading XMP is a much simpler case. One of the key requirements for embedding XMP information in files is that it must be included as text rather than in a compressed binary format, even when embedded in a binary file format. Thus it is in principle easy for any application to extract XMP information; it is certainly possible to do this manually using a text editor.

7 CONCLUSIONS

The primary usefulness of XMP is that it provides a way of embedding RDF/XML in arbitrary documents, whether text-based or binary. Part of the reason for its success in the area of digital imagery is that, through being extensible and XML-based, it can encode the same

1. URL: ⟨http://www.microsoft.com/windowsxp/using/digitalphotography/prophoto/photoinfo.mspx⟩.
2. URL: ⟨http://reader.imagero.com/⟩.
3. TeX Catalogue entry: ⟨http://www.tex.ac.uk/tex-archive/help/Catalogue/entries/xmpincl.html⟩
4. TeX Catalogue entry: ⟨http://www.tex.ac.uk/tex-archive/help/Catalogue/entries/hyperxmp.html⟩
5. URL: ⟨http://www.cs.wisc.edu/~ghost/⟩

information as rival embedded metadata schemes — EXIF and IPTC in particular — in an easily processable form, while allowing additional flexibility.

On the other hand, XMP's highly conservative profile of RDF/XML and (more importantly) its unduly prescriptive handling of the Dublin Core metadata schema limit its usefulness as a method of encoding document metadata. The lack of tools for editing, embedding and viewing XMP, and the deficiencies of those provided by Adobe, are a further discouragement to authors wishing to take advantage of what could otherwise be a powerful method of providing rich metadata sets with documents.

8 ACKNOWLEDGEMENTS

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) and the Economic and Social Research Council (ESRC) under Grant Numbers EP/C534220/1 and RES-331-27-0006.

REFERENCES

Adobe. 2001. 'A manager's introduction to Adobe eXtensible Metadata Platform, the Adobe XML metadata framework.' Adobe Systems Incorporated.

———. 2005*a*. *Custom Panels for XMP File Info*. San Jose, CA: Adobe Systems. URL: ⟨http://www.adobe.com/products/xmp/downloads/XMP_CustomPanels.zip⟩.

———. 2005*b*. 'XMP specification.' Adobe Systems Incorporated. URL: ⟨http://partners.adobe.com/public/developer/en/xmp/sdk/xmpspecification.pdf⟩.

———. 2007. 'Adobe XMP partners.' Web page. Accessed 22 Feb. 2007. URL: ⟨http://www.adobe.com/products/xmp/partners.html⟩.

ISO 8601:2004. 'Data elements and interchange formats — Information interchange — Representation of dates and times.'

ISO/IEC 10646:2003. 'Information technology — Universal Multiple-Octet Coded Character Set (UCS).'

ISO/IEC 8859-1:1998. 'Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet no. 1.'

JEITA CP-3451. 2002. 'Exchangeable image file format for digital still cameras: EXIF version 2.2.' URL: ⟨http://www.exif.org/Exif2-2.PDF⟩.

RFC 2046. 1996. 'Multipurpose Internet mail extensions (MIME) part two: Media types.' URL: ⟨http://tools.ietf.org/html/rfc2046⟩.

RFC 2781. 2000. 'UTF-16, a transformation format of ISO 10646.' URL: ⟨http://tools.ietf.org/html/rfc2781⟩.

RFC 3066. 2001. 'Tags for the identification of languages.' URL: ⟨http://tools.ietf.org/html/rfc3066⟩.

RFC 3629. 2003. 'UTF-8, a transformation format of ISO 10646.' URL: ⟨http://tools.ietf.org/html/rfc3629⟩.

W3C. 1997. 'Date and time formats.' A profile of ISO 8601. URL: ⟨http://www.w3.org/TR/NOTE-datetime⟩.

———. 2002. 'Exclusive XML canonicalization version 1.0.' World Wide Web Consortium Recommendation. URL: ⟨http://www.w3.org/TR/xml-exc-c14n/⟩.

———. 2004. 'RDF primer.' W3C Recommendation, World Wide Web Consortium. URL: ⟨http://www.w3.org/TR/rdf-primer/⟩.

All links were correct on 22 February 2007.