# The hyperxmp package*

Scott Pakin
scott+hyxmp@pakin.org

February 23, 2017

**Abstract**

hyperxmp makes it easy for an author to include XMP metadata in a PDF document produced by LaTeX. hyperxmp integrates seamlessly with hyperref and requires virtually no modifications to a document that already specifies document metadata through hyperref's mechanisms.

## 1 Introduction

Adobe Systems, Inc. has been promoting XMP [4]—eXtensible Metadata Platform—as a standard way to include metadata within a document. The idea behind XMP is that it is an XML-based description of various document attributes and is embedded as uncompressed, unencoded text within the document it describes. By storing the metadata this way it is independent of the document's file format. That is, regardless of whether a document is in PDF, JPEG, HTML, or any other format, it is trivial for a program (or human) to locate, extract, and—using any standard XML parser—process the embedded XMP metadata.

As of this writing there are few tools that actually do process XMP. However, it is easy to imagine future support existing in file browsers for displaying not only a document's filename but also its title, list of authors, description, and other metadata.

**This is too abstract! Give me an example.** Consider a LaTeX document with three authors: Jack Napier, Edward Nigma, and Harvey Dent. The generated PDF file will contain, among other information, the following stanza of XMP code embedded within it:

```
<dc:creator>
  <rdf:Seq>
    <rdf:li>Jack Napier</rdf:li>
    <rdf:li>Edward Nigma</rdf:li>
    <rdf:li>Harvey Dent</rdf:li>
```

*This document corresponds to hyperxmp v3.2, dated 2017/02/23.

```
    </rdf:Seq>
  </dc:creator>
```

In the preceding code, the dc namespace refers to the Dublin Core schema, a collection of metadata properties. The dc:creator property surrounds the list of authors. The rdf namespace is the Resource Description Framework, which defines rdf:Seq as an ordered list of values. Each author is represented by an individual list item (rdf:li), making it easy for an XML parser to separate the authors' names.

Remember that XMP code is stored as *metadata*. It does not appear when viewing or printing the PDF file. Rather, it is intended to make it easy for applications to identify and categorize the document.

**What metadata does hyperxmp process?** hyperxmp knows how to embed all of the following types of metadata within a document:

- authors (dc:creator)
- base URL (xmp:BaseURL)
- contact address (Iptc4xmpCore:CreatorContactInfo/CiAdrExtadr, Iptc4xmpCore:CreatorContactInfo/CiAdrCity, Iptc4xmpCore:CreatorContactInfo/CiAdrRegion, Iptc4xmpCore:CreatorContactInfo/CiAdrPcode, and Iptc4xmpCore:CreatorContactInfo/CiAdrCtry)
- contact email address(es) (Iptc4xmpCore:CreatorContactInfo/CiEmailWork)
- contact telephone number(s) (Iptc4xmpCore:CreatorContactInfo/CiTelWork)
- contact URL(s) (Iptc4xmpCore:CreatorContactInfo/CiUrlWork)
- copyright (dc:rights and xmpRights:Marked)
- date (dc:date, xmp:CreateDate, xmp:ModifyDate, and xmp:MetadataDate)
- document identifier (xmpMM:DocumentID)
- document instance identifier (xmpMM:InstanceID)
- document type (dc:type)
- file format (dc:format)
- keywords (pdf:Keywords and dc:subject)
- language (dc:language)
- LaTeX file name (dc:source)
- license URL (xmpRights:WebStatement)

- metadata writer (photoshop:CaptionWriter)
- PDF version (pdf:PDFVersion)
- PDF-generating tool (pdf:Producer and xmp:CreatorTool)
- PDF/A compliance level and version (pdfaid:part and pdfaid:conformance)
- primary author's position/title (photoshop:AuthorsPosition)
- summary (dc:description)
- title (dc:title)

More types of metadata may be added in a future release.

**How does hyperxmp compare to the xmpincl package?** The short answer is that xmpincl is more flexible but hyperxmp is easier to use. With xmpincl, the author manually constructs a file of arbitrary XMP data and the package merely embeds it within the generated PDF file. With hyperxmp, the author specifies values for various predefined metadata types and the package formats those values as XMP and embeds the result within the generated PDF file.

xmpincl can embed XMP only when running under pdfLaTeX and only when in PDF-generating mode. hyperxmp additionally works with a few other PDF-producing LaTeX backends.

hyperxmp and xmpincl can complement each other. An author may want to use hyperxmp to produce a basic set of XMP code, then extract the XMP code from the PDF file with a text editor, augment the XMP code with any metadata not supported by hyperxmp, and use xmpincl to include the modified XMP code in the PDF file.

# 2 Usage

hyperxmp works by postprocessing some of the package options honored by hyperref. To use hyperxmp, merely put a `\usepackage{hyperxmp}` in your document's preamble. That line can appear anywhere before the hyperref PDF options are specified (i.e., with either `\usepackage[...]{hyperref}` or `\hypersetup{...}`). hyperxmp will construct its XMP data using the following hyperref options:

- baseurl
- pdfauthor
- pdfcreationdate
- pdfkeywords
- pdflang

- pdfmoddate
- pdfproducer
- pdfsubject
- pdftitle

hyperxmp instructs hyperref also to accept the following options, which have meaning only to hyperxmp:

- pdfaconformance
- pdfapart
- pdfauthortitle
- pdfcaptionwriter
- pdfcontactaddress
- pdfcontactcity
- pdfcontactcountry
- pdfcontactemail
- pdfcontactphone
- pdfcontactpostcode
- pdfcontactregion
- pdfcontacturl
- pdfcopyright
- pdfdate
- pdflicenseurl
- pdfmetadate
- pdfmetalang
- pdftype

The two most obscure—but alphabetically first—of the above, pdfaconformance and pdfapart, are used in conjunction with hyperref's pdfa option to claim a particular PDF/A standard by which the document abides. They default to pdfapart=1 and pdfaconformance=B, indicating the PDF/A-1B standard. These can be changed (with caution) to assert that the document abides by a different standard (e.g., PDF/A-2U).

pdfauthortitle indicates the primary author's position or title. pdfcaptionwriter specifies the name of the person who added the metadata to the document. The next eight items describe how to contact the person or institution responsible for the document (the "contact"). pdfcontactaddress is the contact's street address and can include the institution name if the contact is an institution; pdfcontactcity is the contact's city. pdfcontactcountry is the contact's country; pdfcontactemail is the contact's email address (or multiple, comma-separated email addresses); pdfcontactphone is the contact's telephone number (or multiple, comma-separated telephone numbers); pdfcontactpostcode is the contact's postal code; pdfcontactregion is the contact's state or province; and pdfcontacturl is the contact's URL (or multiple, comma-separated URLs).

XMP metadata can include a number of dates (in fact, timestamps, as they include both date and time components). pdfdate specifies the document date. It is analogous to the LaTeX `\date` command, and, like `\date`, defaults to the date the document was built. It must be specified in either XMP format [4] or PDF format [3]. XMP dates are written in the form `YYYY-MM-DDThh:mm:ss+TT:tt`.[1] A W3C recommendation [12] discusses this format in more detail, but as an example, 14 hours, 15 minutes, 9 seconds past midnight U.S. Mountain Daylight Time (UTC-6) on the 23rd day of September in the year 2014 should be written as `2014-09-23T14:15:09-06:00`. This can be truncated (with loss of information) to `2014-09-23T14:15:09` or `2014-09-23T14:15` or `2014-09-23` or `2014-09` or `2014` but no other subsets. PDF dates are written in the form `D:YYYYMMDDhhmmss+TT'tt'`. The same date in the preceding example would be written as `D:20140923141509-06'00'` in PDF format.

The document's creation date, modification date, and metadata date are normally set automatically, but pdfcreationdate, pdfmoddate, and pdfmetadate can be used to override the defaults. Like pdfdate, pdfmetadate can be specified in either XMP or PDF format. However, because hyperref defines pdfcreationdate and pdfmoddate and expects these to be written as PDF dates, hyperxmp concomitantly accepts these two dates only in PDF format as well. Note that it's rare that a document would need to specify any of pdfcreationdate, pdfmoddate, or pdfmetadate.

pdfcopyright defines the copyright text, and pdflicenseurl identifies a URL that points to the document's license agreement.

pdfmetalang indicates the natural language in which the metadata is written, typically as an IETF language tag [8], for example, "`en`" for English, "`en-US`" for specifically United States English, "`de`" for German, and so forth. If pdfmetalang is not specified, hyperxmp assumes the metadata language is the same as the document language (hyperref's pdflang option). If neither pdfmetalang nor pdflang is specified, hyperxmp uses only "`x-default`" as the metadata language. Note that "`x-default`" metadata is always included in addition to the specified metadata language, as the user reading the document may not have specified a language preference.

pdftype describes the type of document being produced. This refers

[1] Although allowed by XMP, hyperxmp does not currently accept fractions of a second in timestamps.

to "the nature or genre of the resource" [4] such as "poem", "novel" or "working paper", as opposed to the file format (always "application/pdf" when generated by hyperxmp). Although pdftype can be assigned an arbitrary piece of text, the XMP specification recommends selecting types from a "controlled vocabulary" such as the DCMI Type Vocabulary [5]. The DCMI Type Vocabulary currently consists of only "Collection", "Dataset", "Event", "Image", "InteractiveResource", "MovingImage", "PhysicalObject", "Service", "Software", "Sound", "StillImage", and "Text". pdftype defaults to "Text", which refers to "books, letters, dissertations, poems, newspapers, articles, archives of mailing lists," [5] and other forms of text—all things LaTeX is commonly used to typeset.

It is usually more convenient to provide values for those options using hyperref's \hypersetup command than on the \usepackage command line. See the hyperref manual for more information. The following is a sample LaTeX document that provides values for most of the metadata options that hyperxmp recognizes:

```
\documentclass{article}
\usepackage{hyperxmp}
\usepackage{hyperref}
\title{%
  On a heuristic viewpoint concerning the production and
  transformation of light}
\author{Albert Einstein}
\date{March 17, 1905}
\hypersetup{%
  pdftitle={%
    On a heuristic viewpoint concerning the production and
    transformation of light},
  pdfauthor={Albert Einstein},
  pdfauthortitle={Technical Assistant, Level III},
  pdfdate={1905-03-17},
  pdfcopyright={Copyright (C) 1905, Albert Einstein},
  pdfsubject={photoelectric effect},
  pdfkeywords={energy quanta, Hertz effect, quantum physics},
  pdflicenseurl={http://creativecommons.org/licenses/by-nc-nd/3.0/},
  pdfcaptionwriter={Scott Pakin},
  pdfcontactaddress={Kramgasse 49},
  pdfcontactcity={Bern},
  pdfcontactpostcode={3011},
  pdfcontactcountry={Switzerland},
  pdfcontactphone={031 312 00 91},
  pdfcontactemail={aeinstein@ipi.ch},
  pdfcontacturl={%
    http://einstein.biz/,
    https://www.facebook.com/AlbertEinstein
  },
  pdflang={en},
  baseurl={http://mirror.ctan.org/macros/latex/contrib/hyperxmp/}
}
```

```
\begin{document}
\maketitle
A profound formal difference exists between the theoretical
concepts that physicists have formed about gases and other
ponderable bodies, and Maxwell's theory of electromagnetic
processes in so-called empty space\dots
\end{document}
```

Compile the document to PDF using any of the following approaches:

- pdfLaTeX
- LuaLaTeX
- LaTeX + Dvipdfm
- LaTeX + Dvips + Adobe Acrobat Distiller
- XeLaTeX

Unfortunately, the LaTeX + Dvips + Ghostscript path doesn't work. Ghostscript bug report #690066, closed with "WONTFIX" status on 2012-05-28, explains that Ghostscript doesn't honor the Metadata tag needed to inject a custom XMP packet. Instead, Ghostscript fabricates an XMP packet of its own based on the metadata it finds in the PDF file's Info dictionary (Author, Title, Subject, and Keywords).

Once the document is compiled, the resulting PDF file will contain an XMP packet that looks something like that shown in Appendix A. Figure 1 is a screenshot of the XMP metadata as it appears in Adobe Acrobat's "Advanced" metadata dialog box. Further clicking on the "Advanced" item within that dialog box displays all of the document's metadata sorted by schema as shown in Figure 2.

**Note 1: Acrobat Author bug** A bug in Adobe Acrobat—at least in versions 10.0.1 and earlier—causes that PDF reader to confuse the XMP and non-XMP author lists when displaying the document's metadata. Specifically, the first author is displayed as the concatenated list of authors from the non-XMP data (Author) while the remaining authors are displayed from the XMP data (dc:creator). For example, suppose that a document's authors are Jack Napier, Edward Nigma, and Harvey Dent. When displaying the document properties, Adobe Acrobat replaces "Jack Napier" with a single author named "Jack Napier, Edward Nigma, Harvey Dent" and leaves "Edward Nigma" and "Harvey Dent" as the second and third authors, respectively.

`\XMPTruncateList` The hyperxmp package provides a workaround for this bug in the form of the `\XMPTruncateList` macro. `\XMPTruncateList` takes the name of a list (a hyperref option name) and replaces the list with the value of its first element. Currently, the only meaningful usage is to put

```
\XMPTruncateList{pdfauthor}
```

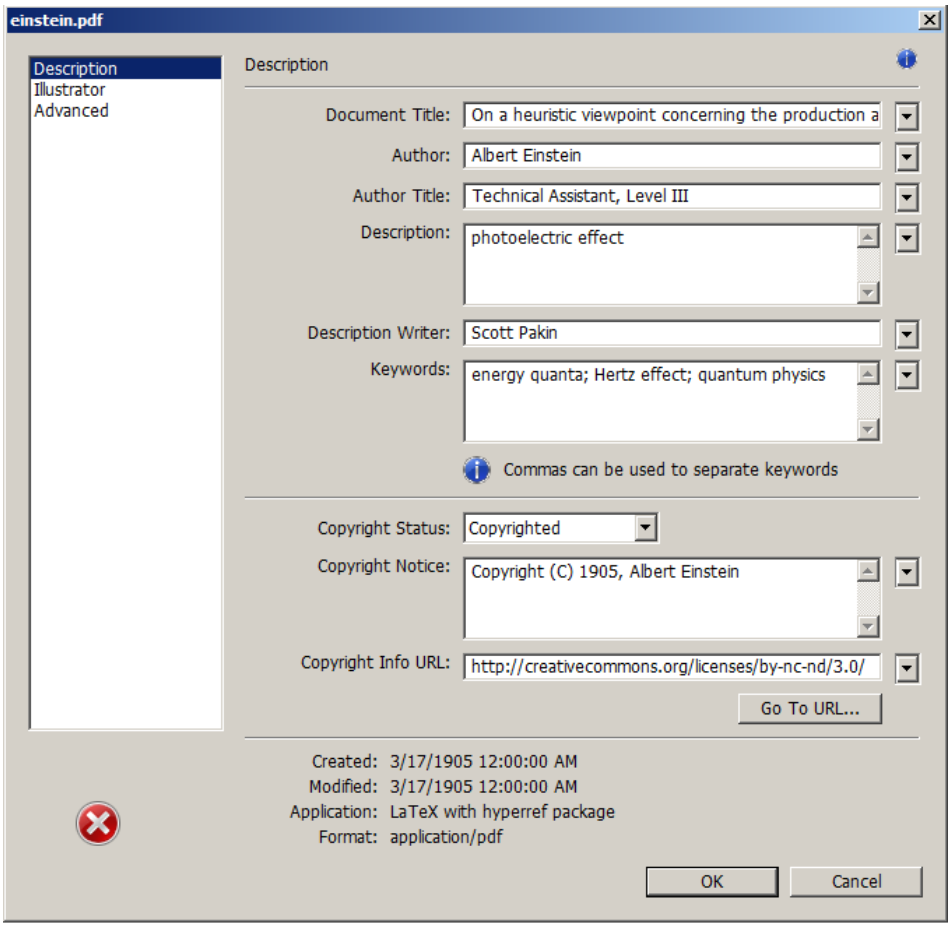


Figure 1: XMP metadata as it appears in Adobe Acrobat

in your document's preamble. This will cause Adobe Acrobat to properly display all of the authors but at the cost of other PDF readers likely displaying only the first author.

**Note 2: Acrobat multiline-field bug** The IPTC Photo Metadata schema states that "the [contact] address is a multiline field" [7]. hyperxmp converts commas in pdfcontactaddress's argument to line breaks in the generated XML. Unfortunately, A bug in Adobe Acrobat—at least in versions 10.0.1 and earlier—causes that PDF reader to discard line breaks in the contact address. Interestingly, Adobe Illustrator CS5 correctly displays the contact address. If you find Adobe Acrobat's behavior bothersome, you can redefine the `\xmplinesep` macro as a string to use as an address-line separator. For example, the following replaces all

`\xmplinesep`

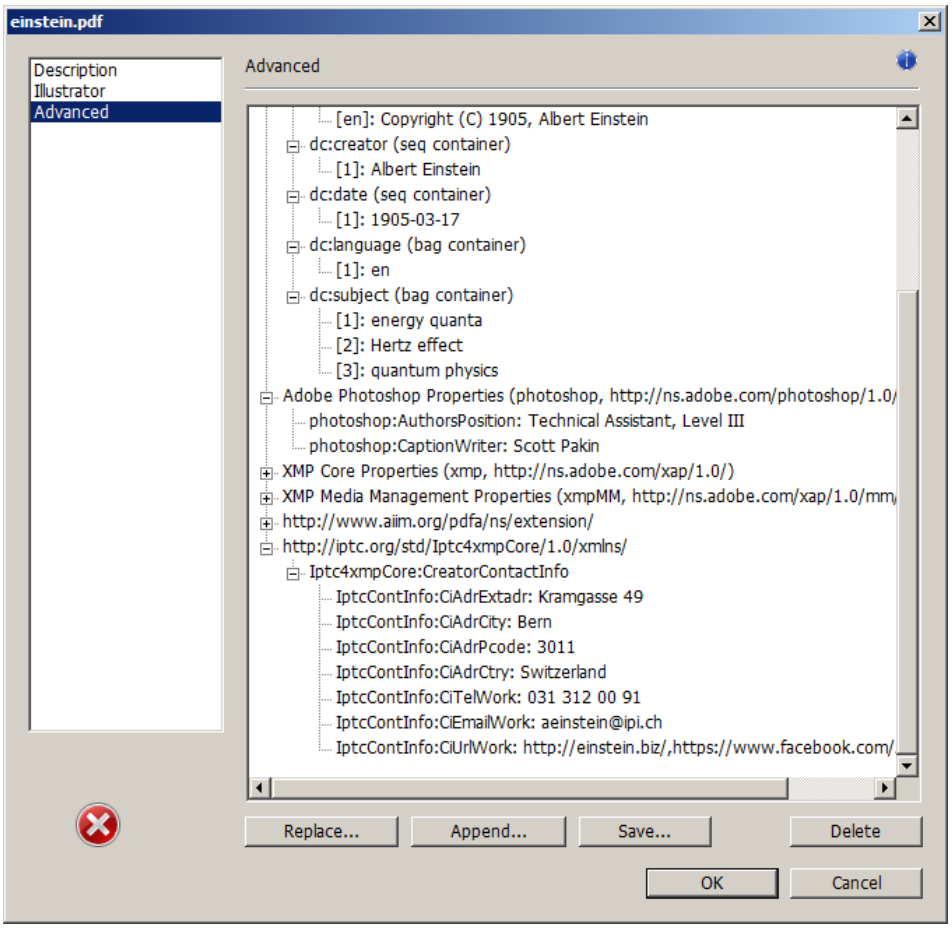


Figure 2: Additional XMP metadata as it appears in Adobe Acrobat

commas appearing in pdfcontactaddress's argument with semicolons:

```
\renewcommand*{\xmlinesep}{;}
```

**Note 3: Object compression** One intention of XMP is that metadata embedded in a file be readable even without knowledge of the file's format. That is, the metadata are expected to appear as plain text. Although hyperxmp does its best to honor that intention, it faces a few challenges:

1. When run with versions of LuaLaTeX earlier than 0.85, hyperxmp leaves all PDF objects uncompressed. This is due to LuaLaTeX treating object compression as a global parameter, unlike pdfLaTeX, which treats it as a local parameter. Hence, when hyperxmp requests that the XMP packet be left uncompressed,

LuaLaTeX in fact leaves *all* PDF streams uncompressed. Beginning with version 3.0, hyperxmp includes a workaround that correctly leaves only the XMP metadata uncompressed, but this workaround is implemented only for LuaLaTeX v0.85 onwards.

2. XeLaTeX (or, more precisely, the `xdvipdfmx` back end) exhibits the opposite problem. It compresses *all* PDF objects, including the ones containing XMP metadata. While Adobe Acrobat can still detect and utilize the XMP metadata, non-PDF-aware applications are unlikely to see the metadata. Three options to consider are to (1) use a different program (e.g., LuaLaTeX), (2) pass the `--output-driver="xdvipdfmx -z0"` option to XeLaTeX to instruct `xdvipdfmx` to turn off all compression (which will of course make the PDF file substantially larger), or (3) postprocess the generated PDF file by loading it into the commercial version of Adobe Acrobat and re-saving it with the Save As... menu option.

**Note 4: Literal commas** hyperxmp splits the pdfauthor and pdfkeywords lists at commas. Therefore, when specifying pdfauthor and pdfkeywords, you should separate items with commas. Also, omit "`and`" and other text that does not belong to any list item. The following examples should serve as clarification:

**Wrong:** `pdfauthor={Jack Napier, Edward Nigma, and Harvey Dent}`

**Wrong:** `pdfauthor={Jack Napier; Edward Nigma; Harvey Dent}`

**Right:** `pdfauthor={Jack Napier, Edward Nigma, Harvey Dent}`

`\xmpcomma` `\xmpquote`

If you need to include a literal comma within an author or keyword list (where commas normally separate list items) or a street address (where commas normally separate lines), use the `\xmpcomma` macro to represent it, and wrap the entire entry containing the comma within `\xmpquote{...}` as shown below:

```
pdfauthor={\xmpquote{Jack Napier\xmpcomma\ Jr.},
           \xmpquote{Edward Nigma\xmpcomma\ PhD},
           \xmpquote{Harvey Dent\xmpcomma\ Esq.}}

pdfcontactaddress={Office of the President,
                   \xmpquote{Wayne Enterprises\xmpcomma\ Inc.},
                   One Wayne Blvd}
```

As of version 2.2 of hyperxmp, it is acceptable to use `\xmpcomma` and `\xmpquote` within any hyperxmp option, not just in those in which a comma normally serves as a separator (i.e., lists and multiline fields). Outside of cases in which a comma serves as a separator, `\xmpcomma` is treated as an ordinary comma, and `\xmpquote` returns its argument unmodified. Hence, it is legitimate to use `\xmpcomma` and `\xmpquote` in cases like the following

```
pdfauthortitle={\xmpquote{Psychiatrist\xmpcomma\ Arkham Asylum}}
```

(Like most hyperxmp options, pdfauthortitle inserts its argument unmodified in an XMP tag.) When in doubt, use `\xmpcomma` and `\xmpquote`; it should always be safe to do so.

`\xmptilde` Version 2.4 of hyperxmp introduces a convenience macro called `\xmptilde`. `\xmptilde` expands to a literal tilde character instead of the nonbreaking space that "~" normally represents. Use it to represent URLs such as `http://www.pakin.org/~scott/` ("`http://www.pakin.org/\xmptilde scott/`") in options such as baseurl, pdfcontacturl and pdflicenseurl.

**Note 5: Unicode support** Unicode support is provided via the hyperref package. If you specify `unicode=true` either as a hyperref option or as an argument to the `\hypersetup` command, the document can include Unicode characters in its XMP fields.

**Note 6: Automatically specified metadata** pdftitle defaults to the document's title as specified by `\title{...}`. pdfauthor defaults to the document's author(s) as specified by `\author{...}`. pdfdate defaults to the current date and time. pdfmetalang defaults to the same value as pdflang if non-empty, "`x-default`" otherwise. An implication of automatic metadata specification is that an author can simply include `\usepackage{hyperxmp}` in a document's preamble and benefit from a modicum of XMP metadata with no additional effort.

# 3 Implementation

This section presents the commented LaTeX source code for hyperxmp. Read this section only if you want to learn how hyperxmp is implemented.

## 3.1 Initial preparation

`\hyxmp@dq@code` The ngerman package redefines " " as an active character, which causes problems for hyperxmp when it tries to use that character. We therefore save the double-quote character's current category code in `\hyxmp@dq@code` and mark the character as category code 12 ("other"). The original category code is restored at the end of the package code (Section 3.7).

```
\edef\hyxmp@dq@code{\the\catcode`\"}
\catcode`\"=12
```

`\hyxmp@at@end` `\hyxmp@driver` The `\hyxmp@at@end` macro includes code at the end of the document. For pdfTeX, the standard `\AtEndDocument` works well enough. For all the other backends we use `\AtEndDvi` from the atenddvi package, which is more robust but requires an addition LaTeX run.

```
\def\hyxmp@driver{hpdftex}
\ifx\hyxmp@driver\Hy@driver
  \let\hyxmp@at@end=\AtEndDocument
```

```
\else
  \RequirePackage{atenddvi}
  \let\hyxmp@at@end=\AtEndDvi
\fi
```

## 3.2 Integration with hyperref

An important design decision underlying hyperxmp is that the package should integrate seamlessly with hyperref. To that end, hyperxmp takes its XMP metadata from hyperref's baseurl, pdfauthor, pdfkeywords, pdflang, pdfproducer, pdfsubject, and pdftitle options. It also introduces a number of new options, which are listed on page 4. For consistency with hyperref's document-metadata naming conventions (which are in turn based on LaTeX's document-metadata naming conventions), we do not prefix metadata-related macro names with our package-specific `\hyxmp@` prefix. That is, we use names like `\@pdfcopyright` instead of `\hyxmp@pdfcopyright`.

We load a bunch of helper packages: kvoptions for package-option processing, pdfescape and stringenc for re-encoding Unicode strings, intcalc for performing integer calculations (division and modulo), and ifxetex for detecting XeTeX.

```
\RequirePackage{kvoptions}
\RequirePackage{pdfescape}
\RequirePackage{stringenc}
\RequirePackage{intcalc}
\RequirePackage{ifxetex}
```

`\hyxmp@pdfstringdef` `\hyxmp@textunderscore`

Because hyperxmp uses underscores to represent hard spaces, we need "`\_`" to map initially to something other than an underscore, in particular the ASCII NAK (`^^U`) character. To accomplish this, we wrap hyperref's `\pdfstringdef` macro with our own version that temporarily does the proper substitution. Later in the execution, after underscores have been replaced with spaces, we replace NAK characters with underscores.

```
\newcommand{\hyxmp@pdfstringdef}[2]{%
  \let\hyxmp@textunderscore=\textunderscore
  \let\textunderscore=\hyxmp@uscore
  \pdfstringdef{#1}{#2}%
  \let\textunderscore=\hyxmp@textunderscore
}
```

`\@pdfdatetime`

Prepare to store the document's date and (optionally) time. Whether specified by the author in XMP format or PDF format (cf. Section 3.3.2) we always store `\@pdfdatetime` as an XMP-format string.

```
\def\@pdfdatetime{}
\define@key{Hyp}{pdfdate}{%
  \begingroup
    \Hy@unicodefalse
```

`\next`

Expand pdfdate's argument and convert it to XMP format.

```
    \edef\next{%
      \noexpand\hyxmp@pdfstringdef\noexpand\@pdfdatetime{%
        \noexpand\hyxmp@as@xmp@date{#1}}%
    }%
    \next
  \endgroup
}
```

`\@pdfmetadatetime` Prepare to store the document's metadata date and (optionally) time. Whether specified by the author in XMP format or PDF format (cf. Section 3.3.2) we always store `\@pdfmetadatetime` as an XMP-format string.

```
\def\@pdfmetadatetime{}
\define@key{Hyp}{pdfmetadate}{%
  \begingroup
    \Hy@unicodefalse
```

`\next` Expand pdfmetadate's argument and convert it to XMP format.

```
    \edef\next{%
      \noexpand\hyxmp@pdfstringdef\noexpand\@pdfmetadatetime{%
        \noexpand\hyxmp@as@xmp@date{#1}}%
    }%
    \next
  \endgroup
}
```

`\@pdfcopyright` Prepare to store the document's copyright statement.

```
\def\@pdfcopyright{}
\define@key{Hyp}{pdfcopyright}{\hyxmp@pdfstringdef\@pdfcopyright{#1}}
```

`\@pdftype` Prepare to store the document's logical type, which defaults to "`Text`".

```
\def\@pdftype{Text}
\define@key{Hyp}{pdftype}{\hyxmp@pdfstringdef\@pdftype{#1}}
```

`\@pdflicenseurl` Prepare to store the URL containing the document's license agreement.

```
\def\@pdflicenseurl{}
\define@key{Hyp}{pdflicenseurl}{\hyxmp@pdfstringdef\@pdflicenseurl{#1}}
```

`\@pdfauthortitle` Prepare to store the author's position/title (e.g., Staff Writer).

```
\def\@pdfauthortitle{}
\define@key{Hyp}{pdfauthortitle}{\hyxmp@pdfstringdef\@pdfauthortitle{#1}}
```

`\@pdfcaptionwriter` Prepare to store the name of the person who inserted the hyperxmp metadata.

```
\def\@pdfcaptionwriter{}
\define@key{Hyp}{pdfcaptionwriter}{\hyxmp@pdfstringdef\@pdfcaptionwriter{#1}}
```

`\@pdfmetalang` Prepare to store the natural language of the document's metadata, typically as an ISO 639-1 two-letter abbreviation.

```
\def\@pdfmetalang{}
\define@key{Hyp}{pdfmetalang}{\hyxmp@pdfstringdef\@pdfmetalang{#1}}
```

`\@pdfapart` Prepare to store the PDF/A part ID, which defaults to "1".

```
\def\@pdfapart{1}
\define@key{Hyp}{pdfapart}{\hyxmp@pdfstringdef\@pdfapart{#1}}
```

`\@pdfaconformance` Prepare to store the PDF/A conformance ID, which defaults to "B".

```
\def\@pdfaconformance{B}
\define@key{Hyp}{pdfaconformance}{\hyxmp@pdfstringdef\@pdfaconformance{#1}}
```

The following eight macros—`\@pdfcontactaddress`, `\@pdfcontactcity`, `\@pdfcontactregion`, `\@pdfcontactpostcode`, `\@pdfcontactcountry`, `\@pdfcontactphone`, `\@pdfcontactemail`, and `\@pdfcontacturl`—together specify how to contact the person or institution responsible for the document.

`\@pdfcontactaddress` Prepare to store a street address for the document's contact person/institution. The IPTC standard defines this as follows:

> The contact information address part. Comprises an optional company name and all required information to locate the building or postbox to which mail should be sent. To that end, the address is a multiline field.

For consistency with the rest of hyperxmp, we use commas to separate terms, in this case, lines of the address. The author can use `\xmpquote` and `\xmpcomma` to include literal commas.

```
\def\@pdfcontactaddress{}
\define@key{Hyp}{pdfcontactaddress}{%
  \let\xmpcomma=\hyxmp@comma
  \def\xmpquote##1{##1}%
  \hyxmp@pdfstringdef\@pdfcontactaddress{#1}%
  \def\xmpcomma{,}%
  \let\xmpquote=\relax
}
```

`\@pdfcontactcity` Prepare to store the city of the document's contact person/institution.

```
\def\@pdfcontactcity{}
\define@key{Hyp}{pdfcontactcity}{\hyxmp@pdfstringdef\@pdfcontactcity{#1}}
```

`\@pdfcontactregion` Prepare to store the state or province of the document's contact person/institution.

```
\def\@pdfcontactregion{}
\define@key{Hyp}{pdfcontactregion}{\hyxmp@pdfstringdef\@pdfcontactregion{#1}}
```

`\@pdfcontactpostcode` Prepare to store the postal code of the document's contact person/institution.

```
\def\@pdfcontactpostcode{}
\define@key{Hyp}{pdfcontactpostcode}{\hyxmp@pdfstringdef\@pdfcontactpostcode{#1}}
```

`\@pdfcontactcountry` Prepare to store the country of the document's contact person/institution.

```
\def\@pdfcontactcountry{}
\define@key{Hyp}{pdfcontactcountry}{\hyxmp@pdfstringdef\@pdfcontactcountry{#1}}
```

\@pdfcontactphone Prepare to store the telephone number of the document's contact person/institution.

```
\def\@pdfcontactphone{}
\define@key{Hyp}{pdfcontactphone}{\hyxmp@pdfstringdef\@pdfcontactphone{#1}}
```

\@pdfcontactemail Prepare to store the email address of the document's contact person/institution.

```
\def\@pdfcontactemail{}
\define@key{Hyp}{pdfcontactemail}{\hyxmp@pdfstringdef\@pdfcontactemail{#1}}
```

\@pdfcontacturl Prepare to store the URL of the document's contact person/institution.

```
\def\@pdfcontacturl{}
\define@key{Hyp}{pdfcontacturl}{\hyxmp@pdfstringdef\@pdfcontacturl{#1}}
```

We need to capture list arguments (viz. pdfauthor and pdfkeywords) before hyperref converts them to PDFDocEncoding. Otherwise, \xmpcomma is permanently replaced with a comma, and we lose our ability to change it to a \hyxmp@comma. We therefore need to augment hyperref's option processing with our own. Because hyperref has not yet been loaded we need to ensure that our augmentation gets loaded in the future: after the \usepackage{hyperref} but before options are passed to that package.

For lack of a better approach, hyperxmp redefines \ProcessKeyvalOptions to alter the way hyperref processes pdfauthor and pdfkeywords. This is somewhat heavy-handed as it gets executed for *every* subsequently loaded package that uses \ProcessKeyvalOptions, but at least it does what we need. hyperxmp also redefines \hypersetup to do the same thing. This is required in case hyperref is loaded before hyperxmp.

\hyxmp@pdfauthor
\hyxmp@pdfkeywords
Prepare to store the name of the author and a list of keywords.

```
\def\hyxmp@pdfauthor{}
\def\hyxmp@pdfkeywords{}
```

\hyxmp@redefine@Hyp If not already redefined, redefine hyperref's pdfauthor and pdfkeywords options to properly handle \xmpcomma and \xmpquote.

```
\newcommand*{\hyxmp@redefine@Hyp}{%
```

\hyxmp@Hyp@pdfauthor Store the old definition of \KV@Hyp@pdfauthor in \hyxmp@Hyp@pdfauthor, but only if we see that \KV@Hyp@pdfauthor is defined and \hyxmp@Hyp@pdfauthor isn't. Otherwise, we'd be defining \hyxmp@Hyp@pdfauthor in terms of itself and creating an infinite loop.

```
  \@ifundefined{KV@Hyp@pdfauthor}{}{%
    \@ifundefined{hyxmp@Hyp@pdfauthor}{%
      \expandafter\let\expandafter\hyxmp@Hyp@pdfauthor
        \csname KV@Hyp@pdfauthor\endcsname
    }{}%
  }%
```

\KV@Hyp@pdfauthor
\xmpcomma
\xmpquote
\hyxmp@and
\and
\hyxmp@pdfauthor
\@pdfauthor
Redefine \KV@Hyp@pdfauthor to process its argument twice. The first time, \xmpcomma is defined as a placeholder character (\hyxmp@comma) and \xmpquote

as the identity function. The result is stored in `\hyxmp@pdfauthor` for use in structured lists (those surrounding each entry with `<rdf:li>`). The second time, `\xmpcomma` is defined as an ordinary comma, and `\xmpquote` is defined as a macro that puts its argument within double quotes. The result is stored in `\@pdfauthor` for use in unstructured lists (those in which the entire list appears within a single pair of tags). In case pdfauthor is left unspecified and we copy `\author`'s argument to pdfauthor, we temporarily redefine `\and` as the list separator when producing a structured list and as "`and`" when producing an unstructured list.

```
\define@key{Hyp}{pdfauthor}{%
  \let\xmpcomma=\hyxmp@comma
  \def\xmpquote####1{####1}%
  \let\hyxmp@and=\and
  \def\and{,}%
  \hyxmp@Hyp@pdfauthor{##1}%
  \global\let\hyxmp@pdfauthor=\@pdfauthor
  \def\and{and\space}%
  \def\xmpcomma{,}%
  \def\xmpquote####1{"####1"}%
  \hyxmp@Hyp@pdfauthor{##1}%
  \def\xmpcomma{,}%
  \let\xmpquote=\relax
  \let\and=\hyxmp@and
}%
```

`\hyxmp@Hyp@pdfkeywords` The previous block of code now repeats for the keyword list, starting by storing the old definition of `\KV@Hyp@pdfkeywords` in `\hyxmp@Hyp@pdfkeywords`.

```
\@ifundefined{KV@Hyp@pdfkeywords}{}{%
  \@ifundefined{hyxmp@Hyp@pdfkeywords}{%
    \expandafter\let\expandafter\hyxmp@Hyp@pdfkeywords
      \csname KV@Hyp@pdfkeywords\endcsname
  }{}%
}%
```

`\KV@Hyp@pdfkeywords` Redefine `\KV@Hyp@pdfkeywords` to process its argument twice. The first time, `\xmpcomma` is defined as a placeholder character (`\hyxmp@comma`) and `\xmpquote` as the identity function. The result is stored in `\hyxmp@pdfkeywords` for use in structured lists (those surrounding each entry with `<rdf:li>`). The second time, `\xmpcomma` is defined as an ordinary comma, and `\xmpquote` is defined as a macro that puts its argument within double quotes. The result is stored in `\@pdfkeywords` for use in unstructured lists (those in which the entire list appears within a single pair of tags).

`\xmpcomma` `\xmpquote` `\hyxmp@pdfkeywords` `\@pdfkeywords`

```
\define@key{Hyp}{pdfkeywords}{%
  \let\xmpcomma=\hyxmp@comma
  \def\xmpquote####1{####1}%
  \hyxmp@Hyp@pdfkeywords{##1}%
  \global\let\hyxmp@pdfkeywords=\@pdfkeywords
  \def\xmpcomma{,}%
  \def\xmpquote####1{"####1"}%
```

```
    \hyxmp@Hyp@pdfkeywords{##1}%
    \def\xmpcomma{,}%
    \let\xmpquote=\relax
  }%
}
```

`\hyxmp@ProcessKeyvalOptions` `\ProcessKeyvalOptions`

Redefine kvoptions's `\ProcessOptions` command to invoke `\hyxmp@redefine@Hyp` before performing its normal option processing.

```
\let\hyxmp@ProcessKeyvalOptions=\ProcessKeyvalOptions
\renewcommand*{\ProcessKeyvalOptions}{%
  \hyxmp@redefine@Hyp
  \hyxmp@ProcessKeyvalOptions
}
```

`\hyxmp@hypersetup` `\hypersetup`

Redefine hyperref's `\hypersetup` command to invoke `\hyxmp@redefine@Hyp` before performing its normal option processing.

```
\let\hyxmp@hypersetup=\hypersetup
\def\hypersetup{%
  \hyxmp@redefine@Hyp
  \hyxmp@hypersetup
}
```

`\hyxmp@find@metadata` `\hyxmp@concated@metadata`

Issue a warning message if the author failed to specify any metadata at all. This excludes metadata that is included automatically such as the current timestamp. Note that we don't consider `\@pdfmetalang` as metadata as that value is meaningful only when used in conjunction with other information. We also don't examine `\@pdfapart` or `\@pdfaconformance` because those have nonempty default values.

```
\newcommand*{\hyxmp@find@metadata}{%
  \edef\hyxmp@concated@metadata{%
    \@baseurl
    \@pdfauthor
    \@pdfauthortitle
    \@pdfcaptionwriter
    \@pdfcontactaddress
    \@pdfcontactcity
    \@pdfcontactcountry
    \@pdfcontactemail
    \@pdfcontactphone
    \@pdfcontactpostcode
    \@pdfcontactregion
    \@pdfcontacturl
    \@pdfcopyright
    \@pdfcreationdate
    \@pdfdatetime
    \@pdfkeywords
    \@pdflang
    \@pdflicenseurl
    \@pdfmetadatetime
    \@pdfmoddate
```

```
    \@pdfsubject
    \@pdftitle
    \@pdftype
  }%
  \ifx\hyxmp@concated@metadata\@empty
    \PackageWarningNoLine{hyperxmp}{%
\jobname.tex did not specify any metadata to\MessageBreak
include in the XMP packet.\space\space Please see the\MessageBreak
hyperxmp documentation for instructions on how to\MessageBreak
provide metadata values to hyperxmp}%
  \fi
}
```

Rather than load hyperref ourself we let the author do it then verify he actually did. This approach gives the author the flexibility to load hyperxmp and hyperref in either order and to call `\hypersetup` anywhere in the document's preamble, not just before hyperxmp is loaded.

```
\AtBeginDocument{%
  \@ifpackageloaded{hyperref}{%
```

In older versions of hyperref, `\@pdflang` is set to `\@empty` if pdflang is not specified. In newer versions of hyperref, `\@pdflang` is set to `\relax` if pdflang is not specified. The latter is a bit problematic for hyperxmp because it makes `\@pdflang` non-expandable, which causes a literal "`\@pdflang`" to be written as XMP metadata. To avoid that situation we redefine `\@pdflang` as `\@empty` if we see it set to `\relax`.

```
    \ifx\@pdflang\relax
      \let\@pdflang=\@empty
    \fi
```

If the author explicitly specified the language to use for the document's metadata, we use that. If not, we use the document language, specified to hyperref with the pdflang option. If the author did not specify a language, we use `x-default` as the metadata language.

```
    \ifx\@pdflang\@empty
      \let\@pdfmetalang=\hyxmp@x@default
    \else
      \edef\@pdfmetalang{\@pdflang}%
    \fi
    \hyxmp@xmlify\@pdfmetalang
```

If the author explicitly specified the document date, override the compilation timestamp with the specified date.

```
    \ifx\@pdfdatetime\@empty
    \else
      \edef\hyxmp@today{\@pdfdatetime}%
    \fi
```

If the author left pdftitle blank but specified `\title`, use the title for pdftitle. Likewise, if the author left pdfauthor blank but specified `\author`, use the author for pdfauthor.

```
    \ifx\@pdftitle\@empty
      \ifx\@title\@empty
      \else
        \hypersetup{pdftitle={\@title}}%
      \fi
    \fi
    \ifx\@pdfauthor\@empty
      \ifx\@author\@empty
      \else
        \hypersetup{pdfauthor={\@author}}%
      \fi
    \fi
```

We wait until the end of the document to construct the XMP packet and write it to the PDF document catalog. This gives the author ample opportunity to provide metadata to hyperref and thereby hyperxmp.

```
    \hyxmp@at@end{%
      \hyxmp@find@metadata
      \hyxmp@embed@packet
    }%
  }{%
    \PackageWarningNoLine{hyperxmp}{%
\jobname.tex failed to include a\MessageBreak
\string\usepackage\string{hyperref\string}
in the preamble.\MessageBreak
Consequently, all hyperxmp functionality will be\MessageBreak
disabled}%
  }%
}
```

## 3.3 Manipulating author-supplied data

The author provides metadata information to hyperxmp via package options to hyperref or via hyperref's `\hypersetup` command. The functions in this section convert author-supplied lists (e.g., `pdfkeywords={foo, bar, baz}`) into LaTeX lists (e.g., `\@elt {foo} \@elt {bar} \@elt {baz}`) that can be more easily manipulated (Section 3.3.1); trim spaces off the ends of strings (Section 3.3.3); and, in Section 3.3.4, convert text to XML (e.g., from `<scott+hyxmp@pakin.org>` to `&lt;scott+hyxmp@pakin.org&gt;`).

### 3.3.1 List manipulation

We define a macro for converting a list of comma-separated elements (e.g., the list of PDF keywords) to a list of LaTeX `\@elt`-separated elements.

`\hyxmp@commas@to@list` Given a macro name (`#1`) and a comma-separated list (`#2`), define the macro name as the elements of the list, each preceded by `\@elt`. (Executing the macro therefore applies `\@elt` to each element in turn.)

```
\newcommand*{\hyxmp@commas@to@list}[2]{%
```

```
  \gdef#1{}%
  \expandafter\hyxmp@commas@to@list@i\expandafter#1#2,,%
}
```

\hyxmp@commas@to@list@i
\next

Recursively construct macro #1 from comma-separated list #2. Stop if #2 is empty.

```
\def\hyxmp@commas@to@list@i#1#2,{%
  \gdef\hyxmp@sublist{#2}%
  \ifx\hyxmp@sublist\@empty
    \let\next=\relax
  \else
    \hyxmp@trimspaces\hyxmp@sublist
    \@cons{#1}{{\hyxmp@sublist}}%
    \def\next{\hyxmp@commas@to@list@i{#1}}%
  \fi
  \next
}
```

\xmpcomma

Because hyperxmp splits lists at commas, a comma cannot normally be used within a list. We there provide an \xmpcomma macro that can expand to either a true comma or a placeholder character depending on the situation. Here, we bind it to a comma so it can be used in *any* hyperxmp option, not just those that treat commas specially.

```
\def\xmpcomma{,}%
```

\hyxmp@comma

This is what \xmpcomma maps to during list construction. We assume that documents will never otherwise use an ETX (^^C) character in their XMP metadata.

```
\bgroup
  \catcode`\^^C=11
  \gdef\hyxmp@comma{^^C}
\egroup
```

\hyxmp@uscore

This is what _ temporarily maps to during packet construction. Because underscores are replaced by spaces, we need a mechanism to preserve user-specified underscores (e.g., in email addresses). We assume that documents will never otherwise use an NAK (^^U) character in their XMP metadata.

```
\bgroup
  \catcode`\^^U=11
  \gdef\hyxmp@uscore{^^U}
\egroup
```

\xmpquote

Adobe Acrobat likes to see double quotes around list elements that contain commas when the entire list appears within a single XMP tag (e.g., <pdf:Keywords>). However, it doesn't like to see double quotes around list elements that contain commas when the list is broken up into individual components (i.e., using <rdf:li> tags). We therefore introduce an \xmpquote macro that quotes or doesn't quote its argument based on context. Here, we bind \xmpquote to \relax to prevent it from prematurely quoting or not quoting.

```
\let\xmpquote=\relax
```

\xmptilde As a convenience for the user, we define \xmptilde as a category 12 (other) "~" character.

```
\bgroup
  \catcode`\~=12%
  \gdef\xmptilde{~}%
\egroup
```

\XMPTruncateList \hyxmp@temp@str \hyxmp@temp@list \@elt As a workaround for Adobe Acrobat's inability to display author lists correctly (cf. "Acrobat Author bug" on page 7) we introduce a hack that replaces a list with its first element. One can then write "\XMPTruncateList{pdfauthor}" and have Adobe Acrobat display the author list correctly. It's sad that this is necessary, though.

```
\newcommand{\XMPTruncateList}[1]{{%
  \edef\hyxmp@temp@str{\csname hyxmp@#1\endcsname}%
  \hyxmp@commas@to@list{\hyxmp@temp@list}{\hyxmp@temp@str}%
  \def\@elt##1{%
    \expandafter\gdef\csname @#1\endcsname{##1}%
    \let\@elt=\@gobble
  }
  \hyxmp@temp@list
}}
```

### 3.3.2 Date manipulation

hyperxmp needs to manipulate two types of date (really, timestamp) formats: PDF format and XMP format. PDF timestamps are of the form "D:YYYYMMDDhhmmss+TT'tt'" (e.g., D:20170223004110-07'00') [3], while XMP timestamps are of the form "YYYY-MM-DDThh:mm:ss+TT:tt" (e.g., 2017-02-23T00:41:10-07:00) [4]. The \hyxmp@as@pdf@date and \hyxmp@as@xmp@date macros defined in this section facilitate timestamp conversions to PDF and XMP formats, respectively.

\hyxmp@first@char \hyxmp@first@char@i Return the first character of a string. This macro is fully expandable.

```
\def\hyxmp@first@char#1{\hyxmp@first@char@i#1\relax}
\def\hyxmp@first@char@i#1#2\relax{#1}
```

\hyxmp@as@xmp@date If necessary, convert a timestamp to XMP format. That is, if the timestamp is in PDF format, convert it; otherwise, leave it unmodified. This macro is fully expandable.

```
\def\hyxmp@as@xmp@date#1{%
  \expandafter\ifx\hyxmp@first@char@i#1\relax D%
    \hyxmp@pdf@to@xmp@date{#1}%
  \else
    #1%
  \fi
}
```

`\hyxmp@pdf@to@xmp@date` Convert a timestamp from PDF format to XMP format. This macro is fully expandable.

```
\def\hyxmp@pdf@to@xmp@date#1:#2#3#4#5#6#7#8#9{%
  #2#3#4#5-#6#7-#8#9%
  \hyxmp@parse@time
}
```

`\hyxmp@parse@time` This is a helper function for `\hyxmp@pdf@to@xmp@date`. `\hyxmp@pdf@to@xmp@date` proper parses only the year, month, and day then calls `\hyxmp@parse@time`. `\hyxmp@parse@time` parses the hours, minutes, and seconds then calls `\hyxmp@parse@tz@char`.

```
\def\hyxmp@parse@time#1#2#3#4#5#6{%
  T#1#2:#3#4:#5#6%
  \hyxmp@parse@tz@char
}
```

`\hyxmp@parse@tz@char` This is another helper function for `\hyxmp@pdf@to@xmp@date`. So far, the date and time have been parsed. `\hyxmp@parse@tz@char` parses the first character of the timezone descriptor. This can be one of "`+`" for eastern timezones (UTC+$x$, including Asia, Oceania, and most of Europe), "`-`" for western timezones (UTC−$x$, primarily the Americas), or "`Z`" for Zulu time (UTC+0). Timezones beginning with "`+`" or "`-`" are followed by an offset in hours and minutes (parsed by `\hyxmp@parse@tz`; timezones beginning with "Z" are not.

```
\def\hyxmp@parse@tz@char#1{%
  #1%
  \ifx#1-%
    \expandafter\hyxmp@parse@tz
  \else
    \ifx#1+%
      \expandafter\hyxmp@parse@tz
    \fi
  \fi
}
```

`\hyxmp@parse@tz` This is the final helper function for `\hyxmp@pdf@to@xmp@date`. It parses the piece of the timezone comprising the offset from Coordinated Universal Time, measured in hours and minutes.

```
\def\hyxmp@parse@tz#1'#2'{%
  #1:#2%
}
```

`\hyxmp@as@pdf@date` If necessary, convert a timestamp to PDF format. That is, if the timestamp is in XMP format, convert it; otherwise, leave it unmodified. This macro is fully expandable.

```
\def\hyxmp@as@pdf@date#1{%
  \expandafter\ifx\hyxmp@first@char@i#1\relax D%
    #1%
  \else
```

```
    \hyxmp@xmp@to@pdf@date{#1}%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date` Convert a timestamp from XMP format to PDF format. This macro is fully expandable.

```
\def\hyxmp@xmp@to@pdf@date#1{%
  D:\hyxmp@xmp@to@pdf@date@i#1\relax\relax
}
```

`\hyxmp@xmp@to@pdf@date@i` Parse the year for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@i#1#2#3#4#5#6{%
  #1#2#3#4%
  \ifx#5-%
    \expandafter\hyxmp@xmp@to@pdf@date@ii\expandafter#6%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date@ii` Parse the month for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@ii#1#2#3#4{%
  #1#2%
  \ifx#3-%
    \expandafter\hyxmp@xmp@to@pdf@date@iii\expandafter#4%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date@iii` Parse the day for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@iii#1#2#3#4{%
  #1#2%
  \ifx#3T%
    \expandafter\hyxmp@xmp@to@pdf@date@iv\expandafter#4%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date@iv` Parse the hour for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@iv#1#2#3#4{%
  #1#2%
  \ifx#3:%
    \expandafter\hyxmp@xmp@to@pdf@date@v\expandafter#4%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date@v` Parse the minute for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@v#1#2#3#4{%
  #1#2%
  \ifx#3:%
    \expandafter\hyxmp@xmp@to@pdf@date@vi\expandafter#4%
  \fi
}
```

`\hyxmp@gobbletwo` This is exactly the same as LaTeX 2ε's `\@gobbletwo` but needs to be a different literal for `\hyxmp@xmp@to@pdf@date@vii`'s pattern-matching to work.

```
\let\@hyxmp@gobbletwo=\@gobbletwo
```

`\hyxmp@xmp@to@pdf@date@vi` Parse the second for `\hyxmp@xmp@to@pdf@date`. The challenge here is that we need to handle four cases for the character following the seconds—"`+`", "`-`", "`Z`", and no character—without sacrificing expandability. Our tricky solution is to insert a `\@gobbletwo` as a sentinel and let `\hyxmp@xmp@to@pdf@date@vi` discard everything up to that sentinel (i.e., all the other conditionals).

```
\def\hyxmp@xmp@to@pdf@date@vi#1#2#3#4{%
  #1#2%
  \ifx#3+%
    +\expandafter\hyxmp@xmp@to@pdf@date@vii
  \fi
  \ifx#3-%
    -\expandafter\hyxmp@xmp@to@pdf@date@vii
  \fi
  \ifx#3Z%
    Z%
  \fi
  \ifx#3\relax
    \expandafter\@hyxmp@gobbletwo
  \fi
  \@gobbletwo #4%
}
```

`\hyxmp@xmp@to@pdf@date@vii` Parse the time-zone hours for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@vii#1\@gobbletwo#2#3#4#5{%
  #2#3%
  \ifx#4:%
    \expandafter\hyxmp@xmp@to@pdf@date@viii\expandafter#5%
  \fi
}
```

`\hyxmp@xmp@to@pdf@date@viii` Parse the time-zone minutes for `\hyxmp@xmp@to@pdf@date`.

```
\def\hyxmp@xmp@to@pdf@date@viii#1#2#3#4{%
  '#1#2'%
}
```

`\hyxmp@today@define` Use TeX primitives to define a given macro as today's date in YYYY-MM-DDThh:mm format.

```
\def\hyxmp@today@define#1{%
```

The date is a straightforward representation of TeX's `\year`, `\month`, and `\day` primitives, with the latter two zero-padded to two digits apiece.

```
  \xdef#1{\the\year}%
  \ifnum\month<10
    \xdef#1{#1-0\the\month}%
  \else
```

```
    \xdef#1{#1-\the\month}%
  \fi
  \ifnum\day<10
    \xdef#1{#1-0\the\day}%
  \else
    \xdef#1{#1-\the\day}%
  \fi
```

TEX does not provide the time in terms of separate hours and minutes but rather as the total number of minutes since midnight (`\time`). There's no mechanism in TEX to query the number of seconds since midnight or the timezone so we omit those fields when defining macro `#1`.

```
  \@tempcnta=\time
  \divide\@tempcnta by 60%
  \ifnum\@tempcnta<10%
    \xdef#1{#1T0\the\@tempcnta}%
  \else
    \xdef#1{#1T\the\@tempcnta}%
  \fi
  \multiply\@tempcnta by -60%
  \advance\@tempcnta by \time
  \ifnum\@tempcnta<10%
    \xdef#1{#1:0\the\@tempcnta}%
  \else
    \xdef#1{#1:\the\@tempcnta}%
  \fi
}
```

`\hyxmp@today` Define `\hyxmp@today` as the current date and (if available) time and timezone in XMP Date format [4].

```
\@ifundefined{pdffeedback}{%
  \@ifundefined{pdfcreationdate}{%
```

Case 1: Neither `\pdffeedback` nor `\pdfcreationdate` is defined (XeLaTeX and regular LaTeX).

```
    \hyxmp@today@define\hyxmp@today
  }{%
```

Case 2: `\pdfcreationdate` is defined (pdfLaTeX and pre-0.85 LuaLaTeX).

```
    \edef\hyxmp@today{\expandafter\hyxmp@pdf@to@xmp@date\pdfcreationdate}%
  }%
}{%
```

Case 3: `\pdffeedback` is defined (LuaLaTeX 0.85+).

```
  \edef\hyxmp@today{\expandafter\hyxmp@pdf@to@xmp@date\pdffeedback creationdate}%
}
```

### 3.3.3 Trimming leading and trailing spaces

To make it easier for XMP processors to manipulate our output we define a `\hyxmp@trimspaces` macro to strip leading and trailing spaces from various data

fields.

`\hyxmp@trimspaces` Redefine a macro as its previous value but without leading or trailing spaces. This code—as well as that for its helper macros, `\hyxmp@trimb` and `\hyxmp@trimc`—was taken almost verbatim from a solution to an *Around the Bend* puzzle [6]. Inline comments are also taken from the solution text.

```
\catcode`\Q=3
```

`\hyxmp@trimspaces\x` redefines `\x` to have the same replacement text sans leading and trailing space tokens.

```
\newcommand{\hyxmp@trimspaces}[1]{%
```

Use grouping to emulate a multi-token `afterassignment` queue.

```
  \begingroup
```

Put "`\toks 0 {`" into the `afterassignment` queue.

```
  \aftergroup\toks\aftergroup0\aftergroup{%
```

Apply `\hyxmp@trimb` to the replacement text of `#1`, adding a leading `\noexpand` to prevent brace stripping and to serve another purpose later.

```
  \expandafter\hyxmp@trimb\expandafter\noexpand#1Q Q}%
```

Transfer the trimmed text back into `#1`.

```
  \edef#1{\the\toks0}%
}
```

`\hyxmp@trimb` `\hyxmp@trimb` removes a trailing space if present, then calls `\hyxmp@trimc` to clean up any leftover bizarre `Q`s, and trim a leading space. In order for `\hyxmp@trimc` to work properly we need to put back a `Q` first.

```
\def\hyxmp@trimb#1 Q{\hyxmp@trimc#1Q}
```

`\hyxmp@trimc` Execute `\vfuzz` assignment to remove leading space; the `\noexpand` will now prevent unwanted expansion of a macro or other expandable token at the beginning of the trimmed text. The `\endgroup` will feed in the `\aftergroup` tokens after the `\vfuzz` assignment is completed.

```
\def\hyxmp@trimc#1Q#2{\afterassignment\endgroup \vfuzz\the\vfuzz#1}
\catcode`\Q=11
```

### 3.3.4 Converting text to XML

The "`<`", "`>`", and "`&`" characters are significant to XML. We therefore need to escape them in any author-supplied text.

`\ifhyxmp@unicodetex` `\hyxmp@unicodetextrue` `\hyxmp@unicodetexfalse` XeTeX and LuaTeX natively support Unicode. We define the conditional `\ifhyxmp@unicodetex` to check for these so we can properly handle encoding conversions. The trick here is that Unicode TeX implementations compare decimal 64 to hexadecimal 40 (decimal 64), specified with four carets, and take the TRUE branch; non-Unicode TeX implementations compare decimal 64 to character "`^`" (decimal 94), ignore the "`^^0040`" and the rest of the TRUE branch, and take the FALSE branch.

```
\newif\ifhyxmp@unicodetex
\ifnum64=`\^^^^0040\relax
  \hyxmp@unicodetextrue
\else
  \hyxmp@unicodetexfalse
\fi
```

`\hyxmp@reencode` This is now a placeholder macro needed only for `\@pdfmetalang` in the `\begin{document}`.

```
\newcommand*{\hyxmp@reencode}[1]{}
```

`\SE->pdfdoc@03` Preserve ETX (`^^C`), which is normally an invalid character in PDFDocEncoding. We use it in hyperxmp (and specifically in `\hyxmp@xmlify` below) as a list-element separator.

```
\expandafter\def\csname SE->pdfdoc@03\endcsname{0003}
```

`\SE->pdfdoc@15` Preserve NAK (`^^U`), which is normally an invalid character in PDFDocEncoding. We use it in hyperxmp (and specifically in `\hyxmp@xmlify` below) as a placeholder for an underscore character.

```
\expandafter\def\csname SE->pdfdoc@15\endcsname{0015}
```

`\hyxmp@xmlify` `\hyxmp@xmlified` `\hyxmp@text` Given a piece of text defined using `\pdfstringdef` (i.e., with many special characters redefined to have category code 11), set `\hyxmp@xmlified` to the same text but with all occurrences of "<" replaced with `<`, all occurrences of ">" replaced with `>`, and all occurrences of "&" replaced with `&`.

```
\newcommand*{\hyxmp@xmlify}[1]{%
  \gdef\hyxmp@xmlified{}%
```

Escaped PDF string → PDFDocEncoding/Unicode

```
  \EdefUnescapeString\hyxmp@text{#1}%
  \ifhyxmp@unicodetex
```

PDFDocEncoding/Unicode → UTF-32BE

```
    \hyxmp@is@unicode\hyxmp@text{%
      \StringEncodingConvert
      \hyxmp@text\hyxmp@text{utf16be}{utf32be}%
    }{%
      \ifxetex
        \hyxmp@xetex@crap
      \else
        \StringEncodingConvert
        \hyxmp@text\hyxmp@text{pdfdoc}{utf32be}%
      \fi
    }%
```

UTF-32BE → UTF-32BE as hex string

```
    \EdefEscapeHex\hyxmp@text{\hyxmp@text}%
```

UTF-32BE → XML in ASCII

```
    \edef\hyxmp@text{%
      \expandafter
    }\expandafter\hyxmp@toxml@unicodetex\hyxmp@text
    \relax\relax\relax\relax\relax\relax\relax\relax
  \else
```

PDFDocEncoding/Unicode → UTF-8

```
    \hyxmp@is@unicode\hyxmp@text{%
      \StringEncodingConvert
      \hyxmp@text\hyxmp@text{utf16be}{utf8}%
    }{%
      \StringEncodingConvert
      \hyxmp@text\hyxmp@text{pdfdoc}{utf8}%
    }%
```

UTF-8 → UTF-8 as hex string

```
    \EdefEscapeHex\hyxmp@text{\hyxmp@text}%
```

UTF-8 as hex string → XML in UTF-8 as hex string

```
    \edef\hyxmp@text{%
      \expandafter\hyxmp@toxml\hyxmp@text\@empty\@empty
    }%
```

XML in UTF-8 as hex string → XML in UTF-8

```
    \EdefUnescapeHex\hyxmp@text{\hyxmp@text}%
  \fi
  \global\let\hyxmp@xmlified\hyxmp@text
}
```

`\hyxmp@is@unicode` `\hyxmp@@is@unicode` Given a string and two expressions, evaluate the first expression if the string is UTF-16BE-encoded and the second expression if not.

```
\begingroup
  \lccode`\<=254 %
  \lccode`\>=255 %
  \catcode254=12 %
  \catcode255=12 %
\lowercase{\endgroup
  \def\hyxmp@is@unicode#1{%
    \expandafter\hyxmp@@is@unicode#1<>\@nil
  }%
  \def\hyxmp@@is@unicode#1<>#2\@nil{%
    \ifx\\#1\\%
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi
  }%
}
```

`\hyxmp@toxml` Replace the characters "<", "&", and ">" with XML entities when using a non-native-Unicode TEX (TEX or pdfTEX).

```
\def\hyxmp@toxml#1#2{%
  \ifx#1\@empty
  \else
    \ifnum"#1#2=`\& %
      26616D703B% &amp;
    \else\ifnum"#1#2=`\< %
      266C743B% &lt;
    \else\ifnum"#1#2=`\> %
      2667743B% &gt;
    \else
```

`dvips` wraps text when generating most PostScript code but preserves line breaks within strings. Unfortunately, `dvips` fails to observe the special case in the PostScript specification that "[b]alanced pairs of parentheses in the string require no special treatment" [2]. Consequently, XMP data containing parentheses (e.g., "`Copyright (C) 1605 Miguel de Cervantes`") confuse `dvips` into thinking that the string has ended after the closing parenthesis and that line breaks can subsequently be injected safely into the document at arbitrary points for formatting purposes. This leads to erroneous display by PDF viewers, which honor line breaks within XMP tags. The solution is to insert a backslash before all parentheses when in `pdfmark`-generating mode to convince `dvips` that the entire XMP packet must be treated as a single, not-to-be-modified string.

```
      \@ifundefined{pdfmark}{%
        #1#2%
      }{%
      \ifnum"#1#2=`\( %
        5C28% \(
      \else\ifnum"#1#2=`\) %
        5C29% \)
      \else
        #1#2%
      \fi\fi
      }%
    \fi\fi\fi
    \expandafter\hyxmp@toxml
  \fi
}
```

`\hyxmp@toxml@unicodetex` `\hyxmp@text` Replace the characters "<", "&", and ">" with XML entities when using a native-Unicode TEX (XƎTEX or LuaTEX).

```
\def\hyxmp@toxml@unicodetex#1#2#3#4#5#6#7#8{%
  \ifx#1\relax
  \else
    \ifnum"#1#2#3#4#5#6#7#8>127 %
      \uccode`\*="#1#2#3#4#5#6#7#8\relax
      \uppercase{%
        \edef\hyxmp@text{\hyxmp@text *}%
```

```
      }%
    \else\ifnum"#7#8=`\< %
      \edef\hyxmp@text{\hyxmp@text &lt;}%
    \else\ifnum"#7#8=`\& %
      \edef\hyxmp@text{\hyxmp@text &amp;}%
    \else\ifnum"#7#8=`\> %
      \edef\hyxmp@text{\hyxmp@text &gt;}%
    \else\ifnum"#7#8=`\ %
      \edef\hyxmp@text{\hyxmp@text\space}%
    \else
      \uccode`\*="#7#8\relax
      \uppercase{%
        \edef\hyxmp@text{\hyxmp@text *}%
      }%
    \fi\fi\fi\fi\fi
    \expandafter\hyxmp@toxml@unicodetex
  \fi
}
```

`\hyxmp@skipzeros` Skip over leading zeroes in the input argument.

```
\def\hyxmp@skipzeros#1{%
  \ifx#10%
    \expandafter\hyxmp@skipzeros
  \fi
}
```

`\x` `\hyxmp@xetex@crap` `\hyxmp@try` `\hyxmp@crap@result` `\hyxmp@text` In the case of XeTeX, the strings defined by `\pdfstringdef` can contain big characters. In this case, the string is treated as Unicode.

```
\begingroup
\def\x#1{\endgroup
  \def\hyxmp@xetex@crap{%
    \edef\hyxmp@try{%
      \expandafter\hyxmp@SpaceOther\hyxmp@text#1\@nil
    }%
    \let\hyxmp@crap@result=N%
    \expandafter\hyxmp@crap@test\hyxmp@try\relax
    \ifx\hyxmp@crap@result Y%
      \let\hyxmp@text\@empty
      \expandafter\hyxmp@crap@convert\hyxmp@try\relax
    \else
      \StringEncodingConvert\hyxmp@text\hyxmp@text{pdfdoc}{utf32be}%
    \fi
  }%
}
\x{ }
```

`\hyxmp@SpaceOther` Re-encode all spaces in a string with category code 12 (“other”).

```
\begingroup
  \catcode`\~=12 %
```

```
  \lccode`\~=`\ %
\lowercase{\endgroup
  \def\hyxmp@SpaceOther#1 #2\@nil{%
    #1%
    \ifx\relax#2\relax
      \expandafter\@gobble
    \else
      ~%
      \expandafter\@firstofone
    \fi
    {\hyxmp@SpaceOther#2\@nil}%
  }%
}
```

`\hyxmp@crap@test` Determine if we need to treat a string as Unicode.

```
\def\hyxmp@crap@test#1{%
  \ifx#1\relax
  \else
    \ifnum`#1>127 %
      \let\hyxmp@crap@result=Y%
      \expandafter\expandafter\expandafter\hyxmp@skiptorelax
    \else
      \expandafter\expandafter\expandafter\hyxmp@crap@test
    \fi
  \fi
}
```

`\hyxmp@skiptorelax` Discard all tokens up to and including the first `\relax`.

```
\def\hyxmp@skiptorelax#1\relax{}
```

`\hyxmp@crap@convert` Convert a hexadecimal string to a number.
`\hyxmp@num`
`\hyxmp@text`

```
\def\hyxmp@crap@convert#1{%
  \ifx#1\relax
  \else
    \edef\hyxmp@num{\number`#1}%
    \ifnum\hyxmp@num>"FFFFFF %
      \lccode`\!=\intcalcDiv{\hyxmp@num}{\number"1000000}\relax
      \lowercase{\edef\hyxmp@text{\hyxmp@text!}}%
      \edef\hyxmp@num{\intcalcMod{\hyxmp@num}{\number"1000000}}%
    \else
      \edef\hyxmp@text{\hyxmp@text\hyxmp@zero}%
    \fi
    \ifnum\hyxmp@num>"FFFF %
      \lccode`\!=\intcalcDiv{\hyxmp@num}{\number"10000}\relax
      \lowercase{\edef\hyxmp@text{\hyxmp@text!}}%
      \edef\hyxmp@num{\intcalcMod{\hyxmp@num}{\number"10000}}%
    \else
      \edef\hyxmp@text{\hyxmp@text\hyxmp@zero}%
    \fi
```

```
    \ifnum\hyxmp@num>"FF %
      \lccode`\!=\intcalcDiv{\hyxmp@num}{\number"100}\relax
      \lowercase{\edef\hyxmp@text{\hyxmp@text!}}%
      \edef\hyxmp@num{\intcalcMod{\hyxmp@num}{\number"100}}%
    \else
      \edef\hyxmp@text{\hyxmp@text\hyxmp@zero}%
    \fi
    \ifnum\hyxmp@num>0 %
      \lccode`\!=\hyxmp@num\relax
      \lowercase{\edef\hyxmp@text{\hyxmp@text!}}%
    \else
      \edef\hyxmp@text{\hyxmp@text\hyxmp@zero}%
    \fi
    \expandafter\hyxmp@crap@convert
  \fi
}
```

`\hyxmp@zero` Define a null character with category code 12 ("other").

```
\begingroup
  \catcode0=12 %
  \gdef\hyxmp@zero{^^00}%
\endgroup
```

## 3.4 UUID generation

We use a linear congruential generator to produce pseudorandom version 4 UUIDs [9]. True, this method has its flaws but it's simple to implement in TEX and is good enough for producing the XMP xmpMM:DocumentID and xmpMM:InstanceID fields.

`\hyxmp@modulo@a` Replace the contents of `\@tempcnta` with the contents modulo `#1`. Note that `\@tempcntb` is overwritten in the process.

```
\def\hyxmp@modulo@a#1{%
  \@tempcntb=\@tempcnta
  \divide\@tempcntb by #1
  \multiply\@tempcntb by #1
  \advance\@tempcnta by -\@tempcntb
}
```

`\hyxmp@big@prime` `\hyxmp@big@prime@ii` Define a couple of large prime numbers that can still be stored in a TEX counter.

```
\def\hyxmp@big@prime{536870923}
\def\hyxmp@big@prime@ii{536870027}
```

`\hyxmp@seed@rng` `\hyxmp@one@token` Seed hyperxmp's random-number generator from a given piece of text.

```
\def\hyxmp@seed@rng#1{%
  \@tempcnta=\hyxmp@big@prime
  \futurelet\hyxmp@one@token\hyxmp@seed@rng@i#1\@empty
}
```

\hyxmp@seed@rng@i
\hyxmp@one@token
\next

Do all of the work for \hyxmp@seed@rng. For each character code $c$ of the input text, assign \@tempcnta $\leftarrow 3 \cdot$ \@tempcnta $+ c \pmod{\text{\hyxmp@big@prime}}$.

```
\def\hyxmp@seed@rng@i{%
  \ifx\hyxmp@one@token\@empty
    \let\next=\relax
  \else
    \def\next##1{%
      \multiply\@tempcnta by 3
      \advance\@tempcnta by `##1
      \hyxmp@modulo@a{\hyxmp@big@prime}%
      \futurelet\hyxmp@one@token\hyxmp@seed@rng@i
    }%
  \fi
  \next
}
```

\hyxmp@set@rand@num
\hyxmp@rand@num

Advance \hyxmp@rand@num to the next pseudorandom number in the sequence. Specifically, we assign \hyxmp@rand@num $\leftarrow 3 \cdot$ \hyxmp@rand@num $+$ \hyxmp@big@prime@ii $\pmod{\text{\hyxmp@big@prime}}$. Note that both \@tempcnta and \@tempcntb are overwritten in the process.

```
\def\hyxmp@set@rand@num{%
  \@tempcnta=\hyxmp@rand@num
  \multiply\@tempcnta by 3
  \advance\@tempcnta by \hyxmp@big@prime@ii
  \hyxmp@modulo@a{\hyxmp@big@prime}%
  \xdef\hyxmp@rand@num{\the\@tempcnta}%
}
```

\hyxmp@append@hex

Append a randomly selected hexadecimal digit to macro #1. Note that both \@tempcnta and \@tempcntb are overwritten in the process.

```
\def\hyxmp@append@hex#1{%
  \hyxmp@set@rand@num
  \@tempcnta=\hyxmp@rand@num
  \hyxmp@modulo@a{16}%
  \ifnum\@tempcnta<10
    \xdef#1{#1\the\@tempcnta}%
  \else
```

There *must* be a better way to handle the numbers 10–15 than with \ifcase.

```
    \advance\@tempcnta by -10
    \ifcase\@tempcnta
      \xdef#1{#1a}%
      \or\xdef#1{#1b}%
      \or\xdef#1{#1c}%
      \or\xdef#1{#1d}%
      \or\xdef#1{#1e}%
      \or\xdef#1{#1f}%
    \fi
  \fi
}
```

`\hyxmp@append@hex@iii` Invoke `\hyxmp@append@hex` three times.

```
\def\hyxmp@append@hex@iii#1{%
  \hyxmp@append@hex#1%
  \hyxmp@append@hex#1%
  \hyxmp@append@hex#1%
}
```

`\hyxmp@append@hex@iv` Invoke `\hyxmp@append@hex` four times.

```
\def\hyxmp@append@hex@iv#1{%
  \hyxmp@append@hex@iii#1%
  \hyxmp@append@hex#1%
}
```

`\hyxmp@create@uuid` As per the definition of a version 4 UUID [9], define macro `#1` as a UUID of the form "`uuid:`*xxxxxxxx*`-`*xxxx*`-`*4xxx*`-`*yxxx*`-`*xxxxxxxxxxxx*" in which each "*x*" is a lowercase hexadecimal digit and "*y*" is one of "`8`", "`9`", "`a`", or "`b`". We assume that the random-number generator is already seeded. Note that `\hyxmp@create@uuid` overwrites both `\@tempcnta` and `\@tempcntb`.

```
\def\hyxmp@create@uuid#1{%
  \def#1{uuid:}%
  \hyxmp@append@hex@iv#1%
  \hyxmp@append@hex@iv#1%
  \g@addto@macro#1{-}%
  \hyxmp@append@hex@iv#1%
  \g@addto@macro#1{-4}%
  \hyxmp@append@hex@iii#1%
  \g@addto@macro#1{-}%
```

Randomly select one of "`8`", "`9`", "`a`", or "`b`".

```
  \hyxmp@set@rand@num
  \@tempcnta=\hyxmp@rand@num
  \hyxmp@modulo@a{4}%
  \ifcase\@tempcnta
    \g@addto@macro#1{8}%
    \or\g@addto@macro#1{9}%
    \or\g@addto@macro#1{a}%
    \or\g@addto@macro#1{b}%
  \fi
  \hyxmp@append@hex@iii#1%
  \g@addto@macro#1{-}%
  \hyxmp@append@hex@iv#1%
  \hyxmp@append@hex@iv#1%
  \hyxmp@append@hex@iv#1%
}
```

`\hyxmp@def@DocumentID` `\hyxmp@DocumentID` Seed the random-number generator with a function of the current filename, PDF document title, and PDF author, then invoke `\hyxmp@create@uuid` to define `\hyxmp@DocumentID` as a random UUID.

```
\newcommand*{\hyxmp@def@DocumentID}{%
```

```
  \edef\hyxmp@seed@string{\jobname:\@pdftitle:\@pdfauthor}%
  \expandafter\hyxmp@seed@rng\expandafter{\hyxmp@seed@string}%
  \edef\hyxmp@rand@num{\the\@tempcnta}%
  \hyxmp@create@uuid\hyxmp@DocumentID
}
```

`\hyxmp@def@InstanceID` `\hyxmp@InstanceID` Seed the random-number generator with a function of the current filename, PDF document title, PDF author, and the current timestamp, then invoke `\hyxmp@create@uuid` to define `\hyxmp@InstanceID` as a random UUID.

```
\newcommand*{\hyxmp@def@InstanceID}{%
  \edef\hyxmp@seed@string{\jobname:\@pdftitle:\@pdfauthor:\hyxmp@today}%
  \expandafter\hyxmp@seed@rng\expandafter{\hyxmp@seed@string}%
  \edef\hyxmp@rand@num{\the\@tempcnta}%
  \hyxmp@create@uuid\hyxmp@InstanceID
}
```

## 3.5 Constructing the XMP packet

An XMP packet "shall consist of the following, in order: a header PI, the serialized XMP data model (the XMP packet) with optional white-space padding, and a trailer PI" [4]. ("PI" is an abbreviation for "processing instructions"). The serialized XMP includes blocks of XML for various XMP schemata: Adobe PDF (Section 3.5.2), Dublin Core (Section 3.5.3), XMP Rights Management (Section 3.5.4), XMP Media Management (Section 3.5.5), XMP Basic (Section 3.5.6), Photoshop (Section 3.5.7), IPTC Photo Metadata (Section 3.5.8), and PDF/A Identification (Section 3.5.9). The `\hyxmp@construct@packet` macro (Section 3.5.10) constructs the XMP packet into `\hyxmp@xml`. It first writes the appropriate XML header, then calls the various schema-writing macros, then injects `\hyxmp@padding` as padding, and finally writes the appropriate XML trailer.

### 3.5.1 XMP utility functions

`\hyxmp@add@to@xml` Given a piece of text, replace all underscores with category-code 11 ("other") spaces and all `^C` characters with commas, then append the result to the `\hyxmp@xml` macro.

```
\newcommand*{\hyxmp@add@to@xml}[1]{%
  \bgroup
    \@tempcnta=0
    \ifhyxmp@unicodetex
      \@tempcntb=65536%
    \else
      \@tempcntb=256%
    \fi
    \loop
      \lccode\@tempcnta=\@tempcnta
      \advance\@tempcnta by 1
      \ifnum\@tempcnta<\@tempcntb
    \repeat
```

```
    \lccode`\_=`\ \relax
    \lccode`\^^C=`\,\relax
    \lccode`\^^U=`\_\relax
    \lowercase{\xdef\hyxmp@new@xml{#1}}%
    \xdef\hyxmp@xml{\hyxmp@xml\hyxmp@new@xml}%
  \egroup
}
```

`\hyxmp@hash` Define a category-code 11 ("other") version of the "`#`" character.

```
\bgroup
\catcode`\#=11
\gdef\hyxmp@hash{#}
\egroup
```

`\hyxmp@padding` `\hyxmp@xml` The XMP specification recommends leaving approximately 2000 bytes of whitespace at the end of each XMP packet to facilitate editing the packet in place [4]. `\hyxmp@padding` is defined to contain 32 lines of 63 spaces and a newline apiece for a total of 2048 characters of whitespace.

```
\bgroup
  \xdef\hyxmp@xml{}%
  \hyxmp@add@to@xml{%
_______________________________________________________________^^J%
  }
  \xdef\hyxmp@padding{\hyxmp@xml}%
\egroup
\xdef\hyxmp@padding{\hyxmp@padding\hyxmp@padding}
\xdef\hyxmp@padding{\hyxmp@padding\hyxmp@padding}
\xdef\hyxmp@padding{\hyxmp@padding\hyxmp@padding}
\xdef\hyxmp@padding{\hyxmp@padding\hyxmp@padding}
\xdef\hyxmp@padding{\hyxmp@padding\hyxmp@padding}
```

`\hyxmp@x@default` Define an `x-default` string that we can use in comparisons with `\@pdfmetalang`.

```
\newcommand*{\hyxmp@x@default}{x-default}
```

### 3.5.2 The Adobe PDF schema

`\hyxmp@pdf@schema` Add properties defined by the Adobe PDF schema to the `\hyxmp@xml` macro.

```
\newcommand*{\hyxmp@pdf@schema}{%
```

Add a block of XML to `\hyxmp@xml` that lists the document's keywords (the pdf:Keywords property), the tools used to produce the PDF file (the pdf:Producer property), and the version of the PDF standard adhered to (the pdf:PDFVersion property). Unlike most of the other schemata that hyperxmp supports, the Adobe PDF schema is *always* included in the document, even if all of its keys are empty. This is because PDF/A-1b requires the keywords and producer to be the same in the XMP metadata and the PDF metadata. Because hyperref always specifies the Keywords and Producer fields, even when they're empty, hyperxmp has to follow suit and define pdf:Keywords and pdf:Producer in the XMP packet.

```
  \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
____________xmlns:pdf="http://ns.adobe.com/pdf/1.3/">^^J%
  }%
  \hyxmp@add@simple@var{pdf:Keywords}{@pdfkeywords}%
  \hyxmp@add@simple@var{pdf:Producer}{@pdfproducer}%
  \@ifundefined{pdfvariable}{%
    \@ifundefined{pdfminorversion}{%
```

Case 1: Neither `\pdfvariable` nor `\pdfminorversion` is defined (XeLaTeX and regular LaTeX).

```
    }{%
```

Case 2: `\pdfminorversion` is defined (pdfLaTeX and pre-0.85 LuaLaTeX).

```
      \hyxmp@add@simple{pdf:PDFVersion}{1.\the\pdfminorversion}%
    }%
  }{%
```

Case 3: `\pdfvariable` is defined (LuaLaTeX 0.85+).

```
    \hyxmp@add@simple{pdf:PDFVersion}{1.\the\pdfvariable minorversion}%
  }%
  \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
  }%
}
```

`\hyxmp@add@simple` `\hyxmp@string`

Given an XMP tag (`#1`) and a string (`#2`), if the string is nonempty, add a begin tag, the string, and an end tag to the packet. The "`simple`" in the macro name indicates that the string is output without variations for different languages.

```
\newcommand*{\hyxmp@add@simple}[2]{%
  \edef\hyxmp@string{#2}%
  \ifx\hyxmp@string\@empty
  \else
    \hyxmp@xmlify{\hyxmp@string}%
    \hyxmp@add@to@xml{%
_________<#1>\hyxmp@xmlified</#1>^^J%
    }%
  \fi
}
```

`\hyxmp@add@simple@var`

Given an XMP tag (`#1`) and a variable name (`#2`), if the string is defined, add a begin tag, the string, and an end tag to the packet. The "`simple`" in the macro name indicates that the string is output without variations for different languages. `\hyxmp@add@simple@var` differs from `\hyxmp@add@simple` in that the former includes defined but empty values in the XMP packet while the latter excludes both undefined and defined but empty values.

```
\newcommand*{\hyxmp@add@simple@var}[2]{%
  \expandafter\ifx\csname#2\endcsname\relax
  \else
    \hyxmp@xmlify{\csname#2\endcsname}%
```

```
    \hyxmp@add@to@xml{%
_________<#1>\hyxmp@xmlified</#1>^^J%
    }%
  \fi
}
```

### 3.5.3 The Dublin Core schema

`\hyxmp@rdf@dc` Given an optional `\if`⟨*something*⟩ statement (`#1`), a Dublin Core property (`#2`) and a macro containing some `\pdfstringdef`-defined text (`#3`), append the appropriate block of XML to the `\hyxmp@xml` macro.

```
\newcommand*{\hyxmp@rdf@dc}[3][\iffalse]{%
```

Set `\@tempswatrue` only if the given text is nonempty or the provided conditional evaluates to TRUE.

```
  \@tempswafalse
  \ifx#3\@empty
  \else
    \@tempswatrue
  \fi
  #1
    \@tempswatrue
  \fi
```

Append the corresponding XML only if `\@tempswatrue`.

```
  \if@tempswa
    \hyxmp@xmlify{#3}%
    \hyxmp@add@to@xml{%
_________<dc:#2>^^J%
____________<rdf:Alt>^^J%
    }%
    \ifx\@pdfmetalang\hyxmp@x@default
    \else
      \hyxmp@add@to@xml{%
_________________<rdf:li xml:lang="\@pdfmetalang">\hyxmp@xmlified</rdf:li>^^J%
      }%
    \fi
    \hyxmp@add@to@xml{%
_________________<rdf:li xml:lang="\hyxmp@x@default">\hyxmp@xmlified</rdf:li>^^J%
____________</rdf:Alt>^^J%
_________</dc:#2>^^J%
    }%
  \fi
}%
```

`\hyxmp@list@to@xml` Given an optional `\if`⟨*something*⟩ statement (`#1`), a Dublin Core property (`#2`), an RDF array (`#3`), and a macro containing a comma-separated list (`#4`), append the appropriate block of XML to the `\hyxmp@xml` macro.

```
\newcommand*{\hyxmp@list@to@xml}[4][\iffalse]{%
```

Set \@tempswatrue only if the given list is nonempty or the provided conditional evaluates to TRUE.

```
  \@tempswafalse
  \ifx#4\@empty
  \else
    \@tempswatrue
  \fi
  #1
    \@tempswatrue
  \fi
```

Append the corresponding XML only if \@tempswatrue.

```
  \if@tempswa
    \hyxmp@add@to@xml{%
_________<dc:#2>^^J%
____________<rdf:#3>^^J%
    }%
    \bgroup
```

\@elt Re-encode the text from Unicode if necessary. Then redefine \@elt to XML-ify each element of the list and append it to \hyxmp@xmlified.

```
      \hyxmp@xmlify{#4}%
      \hyxmp@commas@to@list\hyxmp@list{\hyxmp@xmlified}%
      \def\@elt##1{%
        \hyxmp@add@to@xml{%
_____________<rdf:li>##1</rdf:li>^^J%
        }%
      }%
      \hyxmp@list
    \egroup
    \hyxmp@add@to@xml{%
____________</rdf:#3>^^J%
_________</dc:#2>^^J%
    }%
  \fi
}
```

\hyxmp@dc@schema Add properties defined by the Dublin Core schema to the \hyxmp@xml macro. Specifically, we add entries for the dc:title property if the author specified a pdftitle, the dc:description property if the author specified a pdfsubject, the dc:rights property if the author specified a pdfcopyright, the dc:creator property if the author specified a pdfauthor, the dc:subject property if the author specified pdfkeywords, and the dc:language property if the author specified pdflang. We also specify the dc:date property using the date the document was run through LaTeX and the dc:source property using the base name of the source file with .tex appended.

```
\newcommand*{\hyxmp@dc@schema}{%
  \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
____________xmlns:dc="http://purl.org/dc/elements/1.1/">^^J%
```

```
818 __________<dc:format>application/pdf</dc:format>^^J%
819   }%
820   \hyxmp@rdf@dc[\ifHy@pdfa]{title}{\@pdftitle}%
821   \hyxmp@rdf@dc[\ifHy@pdfa]{description}{\@pdfsubject}%
822   \hyxmp@rdf@dc{rights}{\@pdfcopyright}%
823   \hyxmp@list@to@xml[\ifHy@pdfa]{creator}{Seq}{\hyxmp@pdfauthor}%
824   \hyxmp@list@to@xml{subject}{Bag}{\hyxmp@pdfkeywords}%
825   \hyxmp@list@to@xml{date}{Seq}{\hyxmp@today}%
826   \hyxmp@list@to@xml{language}{Bag}{\@pdflang}%
827   \hyxmp@list@to@xml{type}{Bag}{\@pdftype}%
828   \hyxmp@add@simple{dc:source}{\jobname.tex}%
829   \hyxmp@add@to@xml{%
830 ______</rdf:Description>^^J%
831   }%
832 }
```

### 3.5.4 The XMP Rights Management schema

`\hyxmp@xmpRights@schema` Add properties defined by the XMP Rights Management schema to the `\hyxmp@xml` macro. Currently, these are only the xmpRights:Marked property and the xmpRights:WebStatement property. If the author specified a copyright statement we mark the document as copyrighted. If the author specified a license statement we include the URL in the metadata.

```
833 \newcommand*{\hyxmp@xmpRights@schema}{%
```

`\hyxmp@legal` Set `\hyxmp@rights` to `YES` if either pdfcopyright or pdflicenseurl was specified.

```
834   \let\hyxmp@rights=\@empty
835   \ifx\@pdflicenseurl\@empty
836   \else
837     \def\hyxmp@rights{YES}%
838   \fi
839   \ifx\@pdfcopyright\@empty
840   \else
841     \def\hyxmp@rights{YES}%
842   \fi
```

Include the license-statement URL and/or the copyright indication. The copyright statement itself is included by `\hyxmp@dc@schema` in Section 3.5.3.

```
843   \ifx\hyxmp@rights\@empty
844   \else
```

Header

```
845     \hyxmp@add@to@xml{%
846 ______<rdf:Description rdf:about=""^^J%
847 ____________xmlns:xmpRights="http://ns.adobe.com/xap/1.0/rights/">^^J%
848     }%
```

Copyright indication

```
849     \ifx\@pdfcopyright\@empty
850     \else
```

```
      \hyxmp@add@to@xml{%
__________<xmpRights:Marked>True</xmpRights:Marked>^^J%
      }%
    \fi
```

License URL

```
    \hyxmp@add@simple{xmpRights:WebStatement}{\@pdflicenseurl}%
```

Trailer

```
    \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
    }%
  \fi
}
```

### 3.5.5 The XMP Media Management schema

`\hyxmp@mm@schema` Add properties defined by the XMP Media Management schema to the `\hyxmp@xml` macro. According to the XMP specification, the xmpMM:DocumentID property is supposed to uniquely identify a document, and the xmpMM:InstanceID property is supposed to change with each save operation [4]. As seen in Section 3.4, we do what we can to honor this intention from within a TEX-based workflow.

```
\gdef\hyxmp@mm@schema{%
  \hyxmp@def@DocumentID
  \hyxmp@def@InstanceID
  \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
_____________xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/">^^J%
__________<xmpMM:DocumentID>\hyxmp@DocumentID</xmpMM:DocumentID>^^J%
__________<xmpMM:InstanceID>\hyxmp@InstanceID</xmpMM:InstanceID>^^J%
______</rdf:Description>^^J%
  }%
}
```

### 3.5.6 The XMP Basic schema

`\hyxmp@define@createdate` `\hyxmp@createdate` Define `\hyxmp@createdate` as the document's creation date but in XMP date format, not PDF date format. We use `\hyxmp@createdate` for the CreateDate, ModifyDate, and MetadataDate fields.

```
\newcommand*{\hyxmp@define@createdate}{%
  \@ifundefined{pdffeedback}{%
    \@ifundefined{pdfcreationdate}{%
```

Case 1: Neither `\pdffeedback` nor `\pdfcreationdate` is defined (XeLaTeX and regular LaTeX).

```
      \hyxmp@today@define\hyxmp@createdate
    }{%
```

Case 2: `\pdfcreationdate` is defined (pdfLaTeX and pre-0.85 LuaLaTeX).

```
      \edef\hyxmp@createdate{\expandafter\hyxmp@pdf@to@xmp@date\pdfcreationdate}%
```

```
    }%
  }{%
```

Case 3: `\pdffeedback` is defined (LuaLaTeX 0.85+).

```
    \edef\hyxmp@createdate{\expandafter\hyxmp@pdf@to@xmp@date\pdffeedback creationdate}%
  }%
}
```

`\hyxmp@xmp@basic@schema` Add properties defined by the XMP Basic schema to the `\hyxmp@xml` macro. These include a bunch of dates (all set to the same value) and the base URL for the document if specified with baseurl.

```
\newcommand*{\hyxmp@xmp@basic@schema}{%
  \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
____________xmlns:xmp="http://ns.adobe.com/xap/1.0/">^^J%
  }%
  \hyxmp@define@createdate
```

For the document's creation date, use the user-specified `\@pdfcreationdate` if defined and non-empty. Otherwise use our fabricated `\hyxmp@createdate`.

```
  \@ifundefined{@pdfcreationdate}{%
    \hyxmp@add@simple{xmp:CreateDate}{\hyxmp@createdate}%
  }{%
    \ifx\@pdfcreationdate\@empty
      \hyxmp@add@simple{xmp:CreateDate}{\hyxmp@createdate}%
    \else
      \hyxmp@add@simple{xmp:CreateDate}{%
        \expandafter\hyxmp@as@xmp@date\expandafter{\@pdfcreationdate}}%
    \fi
  }%
```

For the document's modification date, use the user-specified `\@pdfmoddate` if defined and non-empty. Otherwise use our fabricated `\hyxmp@createdate`.

```
  \@ifundefined{@pdfmoddate}{%
    \hyxmp@add@simple{xmp:ModifyDate}{\hyxmp@createdate}%
  }{%
    \ifx\@pdfmoddate\@empty
      \hyxmp@add@simple{xmp:ModifyDate}{\hyxmp@createdate}%
    \else
      \hyxmp@add@simple{xmp:ModifyDate}{%
        \expandafter\hyxmp@as@xmp@date\expandafter{\@pdfmoddate}}%
    \fi
  }%
```

For the document's metadata date, use the user-specified `\@pdfmetadatetime` if defined and non-empty. Otherwise use our fabricated `\hyxmp@createdate`.

```
  \ifx\@pdfmetadatetime\@empty
    \hyxmp@add@simple{xmp:MetadataDate}{\hyxmp@createdate}%
  \else
    \hyxmp@add@simple{xmp:MetadataDate}{\@pdfmetadatetime}%
  \fi
```

Define the creation tool and the base URL.

```
  \hyxmp@add@simple{xmp:CreatorTool}{\@pdfcreator}%
  \hyxmp@add@simple{xmp:BaseURL}{\@baseurl}%
  \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
  }%
}
```

### 3.5.7 The Photoshop schema

`\hyxmp@photoshop@schema` `\hyxmp@photoshop@data`

Add properties defined by the Photoshop schema to the `\hyxmp@xml` macro. We currently support only the photoshop:AuthorsPosition and photoshop:CaptionWriter properties.

```
\gdef\hyxmp@photoshop@schema{%
  \edef\hyxmp@photoshop@data{\@pdfauthortitle\@pdfcaptionwriter}%
  \ifx\hyxmp@photoshop@data\@empty
  \else
    \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
____________xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/">^^J%
    }%
  \fi
  \hyxmp@add@simple{photoshop:AuthorsPosition}{\@pdfauthortitle}%
  \hyxmp@add@simple{photoshop:CaptionWriter}{\@pdfcaptionwriter}%
  \ifx\hyxmp@photoshop@data\@empty
  \else
    \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
    }%
  \fi
}
```

### 3.5.8 The IPTC Photo Metadata schema

`\xmplinesep`

Lines in multiline fields are separated by `\xmplinesep` in the generated XML. This defaults to an LF (`^^J`) character but written as an XML character entity for consistency across operating systems.

```
\begingroup
  \catcode`\&=12
  \catcode`\#=12
  \gdef\xmplinesep{&#xA;}
\endgroup
```

`\hyxmp@list@to@lines`

Given a property (`#1`) and a macro containing a comma-separated list (`#2`), replace commas with `\xmplinesep`. Do nothing it the list is empty.

```
\newcommand*{\hyxmp@list@to@lines}[2]{%
  \ifx#2\@empty
  \else
```

```
    \bgroup
      \hyxmp@add@to@xml{%
__________<#1>%
      }%
```

`\@elt@first` The first element of the list is output as is.

```
      \def\@elt@first##1{%
        \hyxmp@add@to@xml{##1}%
        \let\@elt=\@elt@rest
      }%
```

`\@elt@rest` The remaining elements of the list are output with a preceding line separator (`\xmplinesep`).

```
      \def\@elt@rest##1{%
        \hyxmp@add@to@xml{\xmplinesep##1}%
      }%
```

`\@elt` Re-encode the text from Unicode if necessary. Then redefine `\@elt` to insert a line separator between terms.

```
      \let\@elt=\@elt@first
      \hyxmp@xmlify{#2}%
      \hyxmp@commas@to@list\hyxmp@list{\hyxmp@xmlified}%
      \hyxmp@list
      \hyxmp@add@to@xml{</#1>^^J}%
    \egroup
  \fi
}
```

`\hyxmp@photometa@schema` `\hyxmp@photometa@data` Add properties defined by the IPTC Photo Metadata schema [7] to the `\hyxmp@xml` macro. We currently support only the contact-information details structure, viz. the Iptc4xmpCore:CreatorContactInfo/CiAdrExtadr, Iptc4xmpCore:CreatorContactInfo/CiAdrCity, Iptc4xmpCore:CreatorContactInfo/CiAdrRegion, Iptc4xmpCore:CreatorContactInfo/CiAdrPcode, Iptc4xmpCore:CreatorContactInfo/CiAdrCtry, Iptc4xmpCore:CreatorContactInfo/CiTelWork, Iptc4xmpCore:CreatorContactInfo/CiEmailWork, and Iptc4xmpCore:CreatorContactInfo/CiUrlWork properties.

```
\gdef\hyxmp@photometa@schema{%
  \edef\hyxmp@photometa@data{%
    \@pdfcontactaddress
    \@pdfcontactcity
    \@pdfcontactregion
    \@pdfcontactpostcode
    \@pdfcontactcountry
    \@pdfcontactphone
    \@pdfcontactemail
    \@pdfcontacturl
  }%
  \ifx\hyxmp@photometa@data\@empty
```

```
  \else
    \hyxmp@iptc@extensions
    \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
__________xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/">^^J%
_______<Iptc4xmpCore:CreatorContactInfo rdf:parseType="Resource">^^J%
    }%
  \fi
  \hyxmp@list@to@lines{Iptc4xmpCore:CiAdrExtadr}{\@pdfcontactaddress}%
  \hyxmp@add@simple{Iptc4xmpCore:CiAdrCity}{\@pdfcontactcity}%
  \hyxmp@add@simple{Iptc4xmpCore:CiAdrRegion}{\@pdfcontactregion}%
  \hyxmp@add@simple{Iptc4xmpCore:CiAdrPcode}{\@pdfcontactpostcode}%
  \hyxmp@add@simple{Iptc4xmpCore:CiAdrCtry}{\@pdfcontactcountry}%
```

`\xmplinesep` The IPTC standard states that sets of telephone numbers, email addresses, and URLs for the contact person or institution, "[m]ay have to be separated by a comma in the user interface" [7]. This is rather ambiguous: Does the comma appear *only* in the user interface or also in the generated XML? Here we assume the latter interpretation and temporarily redefine `\xmplinesep` as a comma and use `\hyxmp@list@to@lines` to insert the data. Unlike `\hyxmp@add@simple`, this approach trims all spaces surrounding commas.

```
  \bgroup
    \def\xmplinesep{,}%
    \hyxmp@list@to@lines{Iptc4xmpCore:CiTelWork}{\@pdfcontactphone}%
    \hyxmp@list@to@lines{Iptc4xmpCore:CiEmailWork}{\@pdfcontactemail}%
    \hyxmp@list@to@lines{Iptc4xmpCore:CiUrlWork}{\@pdfcontacturl}%
  \egroup
  \ifx\hyxmp@photometa@data\@empty
  \else
    \hyxmp@add@to@xml{%
_______</Iptc4xmpCore:CreatorContactInfo>^^J%
______</rdf:Description>^^J%
    }%
  \fi
}
```

`\hyxmp@iptc@extensions` Because IPTC metadata are not recognized by the PDF/A standard, PDF/A conversion would normally fail for documents that utilize `\pdfcontactaddress`, `\pdfcontactcity`, etc. However, there exists a technique, described in a PDF Association technical note [11], for describing nonstandard XMP metadata within the XMP packet itself. We use that technique here to describe all of the metadata that `\hyxmp@photometa@schema` can produce. Doing so enables the document to be converted to PDF/A format.

```
\newcommand*{\hyxmp@iptc@extensions}{%
  \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
__________xmlns:pdfaExtension="http://www.aiim.org/pdfa/ns/extension/"^^J%
__________xmlns:pdfaSchema="http://www.aiim.org/pdfa/ns/schema\hyxmp@hash"^^J%
__________xmlns:pdfaProperty="http://www.aiim.org/pdfa/ns/property\hyxmp@hash"^^J%
```

```
__________xmlns:pdfaType="http://www.aiim.org/pdfa/ns/type\hyxmp@hash"^^J%
__________xmlns:pdfaField="http://www.aiim.org/pdfa/ns/field\hyxmp@hash">^^J%
________<pdfaExtension:schemas>^^J%
__________<rdf:Bag>^^J%
____________<rdf:li rdf:parseType="Resource">^^J%
______________<pdfaSchema:schema>IPTC Core Schema</pdfaSchema:schema>^^J%
______________<pdfaSchema:namespaceURI>http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/</pdfaSchema:
______________<pdfaSchema:prefix>Iptc4xmpCore</pdfaSchema:prefix>^^J%
______________<pdfaSchema:property>^^J%
________________<rdf:Seq>^^J%
__________________<rdf:li rdf:parseType="Resource">^^J%
____________________<pdfaProperty:name>CreatorContactInfo</pdfaProperty:name>^^J%
____________________<pdfaProperty:valueType>contactinfo</pdfaProperty:valueType>^^J%
____________________<pdfaProperty:category>external</pdfaProperty:category>^^J%
____________________<pdfaProperty:description>contact information for the document's creator</p
__________________</rdf:li>^^J%
________________</rdf:Seq>^^J%
______________</pdfaSchema:property>^^J%
______________<pdfaSchema:valueType>^^J%
________________<rdf:Seq>^^J%
__________________<rdf:li rdf:parseType="Resource">^^J%
____________________<pdfaType:type>contactinfo</pdfaType:type>^^J%
____________________<pdfaType:namespaceURI>http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/</pdfaTyp
____________________<pdfaType:prefix>Iptc4xmpCore</pdfaType:prefix>^^J%
____________________<pdfaType:description>contact information</pdfaType:description>^^J%
____________________<pdfaType:field>^^J%
______________________<rdf:Seq>^^J%
  }%
  \hyxmp@text@resource{CiAdrExtadr}{contact address}%
  \hyxmp@text@resource{CiAdrCity}{contact city}%
  \hyxmp@text@resource{CiAdrRegion}{contact region}%
  \hyxmp@text@resource{CiAdrPcode}{contact postal code}%
  \hyxmp@text@resource{CiAdrCtry}{contact country}%
  \hyxmp@text@resource{CiTelWork}{contact telephone number}%
  \hyxmp@text@resource{CiEmailWork}{contact email address}%
  \hyxmp@text@resource{CiUrlWork}{contact url}%
  \hyxmp@add@to@xml{%
______________________</rdf:Seq>^^J%
____________________</pdfaType:field>^^J%
__________________</rdf:li>^^J%
________________</rdf:Seq>^^J%
______________</pdfaSchema:valueType>^^J%
____________</rdf:li>^^J%
__________</rdf:Bag>^^J%
________</pdfaExtension:schemas>^^J%
______</rdf:Description>^^J%
  }%
}
```

`\hyxmp@text@resource` Output a single Text resource given its name and description.

```
\newcommand*{\hyxmp@text@resource}[2]{%
  \hyxmp@add@to@xml{%
__________________________<rdf:li rdf:parseType="Resource">^^J%
_____________________________<pdfaField:name>#1</pdfaField:name>^^J%
_____________________________<pdfaField:valueType>Text</pdfaField:valueType>^^J%
_____________________________<pdfaField:description>#2</pdfaField:description>^^J%
__________________________</rdf:li>^^J%
  }
}
```

### 3.5.9 The PDF/A Identification schema

`\hyxmp@pdfa@id@schema` Add properties defined by the PDF/A Identification schema [10] to the `\hyxmp@xml` macro. These properties identify a document as conforming to a particular PDF/A standard. We default to PDF/A-1b if any PDF/A compliance is detected but let the author override the "1" with pdfapart and the "B" with pdfaconformance.

```
\newcommand*{\hyxmp@pdfa@id@schema}{%
  \ifHy@pdfa
    \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
_____________xmlns:pdfaid="http://www.aiim.org/pdfa/ns/id/">^^J%
    }%
    \hyxmp@add@simple{pdfaid:part}{\@pdfapart}%
    \hyxmp@add@simple{pdfaid:conformance}{\@pdfaconformance}%
    \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
    }%
  \fi
}
```

### 3.5.10 Combining schemata into an XMP packet

`\hyxmp@bom` Define a macro for the Unicode byte-order marker (BOM).

```
\begingroup
  \ifhyxmp@unicodetex
    \lccode`\!="FEFF %
    \lowercase{%
      \gdef\hyxmp@bom{!}
    }%
  \else
    \catcode`\^^ef=12
    \catcode`\^^bb=12
    \catcode`\^^bf=12
    \gdef\hyxmp@bom{^^ef^^bb^^bf}%
  \fi
\endgroup
```

\hyxmp@construct@packet
\hyxmp@xml

Successively add XML data to \hyxmp@xml until we have something we can insert into the document's PDF catalog.

```
\def\hyxmp@construct@packet{%
  \gdef\hyxmp@xml{}%
  \hyxmp@add@to@xml{<?xpacket begin="\hyxmp@bom" %
id="W5M0MpCehiHzreSzNTczkc9d"?>^^J%
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="3.1-702">^^J%
___<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\hyxmp@hash">^^J%
  }%
  \hyxmp@pdf@schema
  \hyxmp@xmpRights@schema
  \hyxmp@dc@schema
  \hyxmp@photoshop@schema
  \hyxmp@photometa@schema
  \hyxmp@xmp@basic@schema
  \hyxmp@pdfa@id@schema
  \hyxmp@mm@schema
  \hyxmp@add@to@xml{%
___</rdf:RDF>^^J%
</x:xmpmeta>^^J%
\hyxmp@padding
<?xpacket end="w"?>^^J%
  }%
}
```

## 3.6 Embedding the XMP packet

The PDF specification says that "a metadata stream may be attached to a document through the Metadata entry in the document catalogue" [3] so that's what we do here.

\hyxmp@embed@packet
\hyxmp@driver

Determine which hyperref driver is in use and invoke the appropriate embedding function.

```
\newcommand*{\hyxmp@embed@packet}{%
  \hyxmp@construct@packet
  \def\hyxmp@driver{hpdftex}%
  \ifx\hyxmp@driver\Hy@driver
    \hyxmp@embed@packet@pdftex
  \else
    \def\hyxmp@driver{hluatex}%
    \ifx\hyxmp@driver\Hy@driver
      \hyxmp@embed@packet@luatex
    \else
      \def\hyxmp@driver{hdvipdfm}%
      \ifx\hyxmp@driver\Hy@driver
        \hyxmp@embed@packet@dvipdfm
      \else
        \def\hyxmp@driver{hxetex}%
```

```
        \ifx\hyxmp@driver\Hy@driver
          \hyxmp@embed@packet@xetex
        \else
          \@ifundefined{pdfmark}{%
            \PackageWarningNoLine{hyperxmp}{%
              Unrecognized hyperref driver `\Hy@driver'.\MessageBreak
              \jobname.tex's XMP metadata will *not* be\MessageBreak
              embedded in the resulting file}%
          }{%
            \hyxmp@embed@packet@pdfmark
          }%
        \fi
      \fi
    \fi
  \fi
}
```

### 3.6.1 Embedding using pdfTeX

Up to version 0.85, LuaTeX supported the pdfTeX primitives, and hyperref didn't distinguish the two backends. However, from hyperxmp's perspective there is one key difference: the effect of `\pdfcompresslevel` is local to a group in pdfTeX but is global in LuaTeX.

The PDF object representing the XMP packet is supposed to include an uncompressed stream so it can be read by non-PDF-aware tools. However, we don't want to unnecessarily uncompress *every* PDF stream. The solution, provided by Hans Hagen on the `luatex` mailing list (thread: "Leaving a single PDF object uncompressed", 6 Jul 2016), is to provide the `uncompressed` flag to `\pdfobj`. Our definition of `\hyxmp@embed@packet@pdftex` uses the ifluatex package to distinguish the pdfTeX case from the pre-0.85 LuaTeX case.

```
\RequirePackage{ifluatex}
```

`\hyxmp@embed@packet@pdftex` Embed the XMP packet using pdfTeX primitives, which are supported by both pdfTeX and pre-0.85 LuaTeX. The only difference is that in the former case we locally specify `\pdfcompresslevel=0` to leave the PDF object uncompressed while in the latter case we pass the `uncompressed` flag to `\pdfobj` to achieve the same effect.

```
\newcommand*{\hyxmp@embed@packet@pdftex}{%
  \bgroup
    \ifluatex
    \else
      \pdfcompresslevel=0
    \fi
    \immediate\pdfobj \ifluatex uncompressed\fi stream attr {%
      /Type /Metadata
      /Subtype /XML
    }{\hyxmp@xml}%
    \pdfcatalog {/Metadata \the\pdflastobj\space 0 R}%
```

```
  \egroup
}
```

### 3.6.2 Embedding using LuaTeX 0.85+

`\hyxmp@embed@packet@luatex` Embed the XMP packet using LuaTeX 0.85+ primitives.

```
\newcommand*{\hyxmp@embed@packet@luatex}{%
  \immediate\pdfextension obj uncompressed stream attr {%
    /Type /Metadata
    /Subtype /XML
  }{\hyxmp@xml}%
  \pdfextension catalog {/Metadata \the\numexpr\pdffeedback lastobj\relax\space 0 R}%
}
```

### 3.6.3 Embedding using any pdfmark-based backend

`\hyxmp@embed@packet@pdfmark` Embed the XMP packet using hyperref's `\pdfmark` command. I believe `\pdfmark` is used by the dvipdf, dvipsone, dvips, dviwindo, nativepdf, pdfmark, ps2pdf, textures, and vtexpdfmark options to hyperref, but I've tested only a few of those.

```
\newcommand*{\hyxmp@embed@packet@pdfmark}{%
  \pdfmark{%
    pdfmark=/NamespacePush
  }%
  \pdfmark{%
    pdfmark=/OBJ,
    Raw={/_objdef \string{hyxmp@Metadata\string} /type /stream}%
  }%
  \pdfmark{%
    pdfmark=/PUT,
    Raw={\string{hyxmp@Metadata\string}
      2 dict begin
        /Type /Metadata def
        /Subtype /XML def
        currentdict
      end
    }%
  }%
  \pdfmark{%
    pdfmark=/PUT,
    Raw={\string{hyxmp@Metadata\string} (\hyxmp@xml)}%
  }%
  \pdfmark{%
    pdfmark=/Metadata,
    Raw={\string{Catalog\string} \string{hyxmp@Metadata\string}}%
  }%
  \pdfmark{%
    pdfmark=/NamespacePop
  }%
}
```

### 3.6.4 Embedding using `dvipdfm`

`\hyxmp@embed@packet@dvipdfm` Embed the XMP packet using `dvipdfm`-specific `\special` commands. Note that `dvipdfm` rather irritatingly requires us to count the number of characters in the `\hyxmp@xml` stream ourselves.

```
\newcommand*{\hyxmp@embed@packet@dvipdfm}{%
  \hyxmp@string@len{\hyxmp@xml}%
  \special{pdf: object @hyxmp@Metadata
    <<
      /Type /Metadata
      /Subtype /XML
      /Length \the\@tempcnta
    >>
    stream^^J\hyxmp@xml endstream%
  }%
  \special{pdf: docview
    <<
      /Metadata @hyxmp@Metadata
    >>
  }%
}
```

`\hyxmp@string@len` Set `\@tempcnta` to the number of characters in a given string (`#1`). The approach is first to tally the number of space characters then to tally the number of non-space characters. While this is rather sloppy I haven't found a better way to achieve the same effect, especially given that all of the characters in `#1` have already been assigned their category codes.

```
\newcommand*{\hyxmp@string@len}[1]{%
  \@tempcnta=0
  \expandafter\hyxmp@count@spaces#1 {} %
  \expandafter\hyxmp@count@non@spaces#1{}%
}
```

`\hyxmp@count@spaces` Count the number of spaces in a given string. We rely on the built-in pattern matching of TEX's `\def` primitive to pry one word at a time off the head of the input string.

```
\def\hyxmp@count@spaces#1 {%
  \def\hyxmp@one@token{#1}%
  \ifx\hyxmp@one@token\@empty
    \advance\@tempcnta by -1
  \else
    \advance\@tempcnta by 1
    \expandafter\hyxmp@count@spaces
  \fi
}
```

`\hyxmp@count@non@spaces` Count the number of non-spaces in a given string. Ideally, we'd count both spaces and non-spaces but TEX won't bind `#1` to a space character (category code 10). Hence, in each iteration, `#1` is bound to the next non-space character only.

```
\newcommand*{\hyxmp@count@non@spaces}[1]{%
  \def\hyxmp@one@token{#1}%
  \ifx\hyxmp@one@token\@empty
  \else
    \advance\@tempcnta by 1
    \expandafter\hyxmp@count@non@spaces
  \fi
}
```

### 3.6.5 Embedding using XeTeX

`\hyxmp@embed@packet@xetex`

Embed the XMP packet using `xdvipdfmx`-specific `\special` commands. I don't know how to tell `xdvipdfmx` always to leave the Metadata stream uncompressed, so the XMP metadata is likely to be missed by non-PDF-aware XMP viewers.

```
\newcommand*{\hyxmp@embed@packet@xetex}{%
  \special{pdf:stream @hyxmp@Metadata (\hyxmp@xml)
    <<
      /Type /Metadata
      /Subtype /XML
    >>
  }%
  \special{pdf:put @catalog
    <<
      /Metadata @hyxmp@Metadata
    >>
  }%
}
```

## 3.7 Final clean-up

Having saved the category code of " " at the start of the package code (Section 3.1), we now restore that character's original category code.

```
\catcode`\"=\hyxmp@dq@code
```

# 4 Future Work

**Help wanted** Ideally, `\xmpquote` should automatically replace all commas with `\xmpcomma`. Unfortunately, my TeX skills are insufficient to pull that off. If you know a way to make `\xmpquote{Hello, world}` work with both Unicode and non-Unicode encodings and with all TeX engines (pdfTeX, LuaTeX, XeTeX, etc.), please send me a code patch.

# A Sample XMP packet

The following is an example of a complete XMP packet as may be produced by hyperxmp. This packet corresponds to the metadata included in the sample LaTeX

document presented on pages 6–7. For clarity, metadata values, either specified explicitly by the document or introduced automatically by hyperxmp, are colored blue.

```
<?xpacket begin="\357\273\277" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="3.1-702">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
            <pdf:Keywords>
              energy quanta, Hertz effect, quantum physics
            </pdf:Keywords>
            <pdf:Producer>pdfTeX-1.40.10</pdf:Producer>
      </rdf:Description>
      <rdf:Description rdf:about=""
            xmlns:xmpRights="http://ns.adobe.com/xap/1.0/rights/">
         <xmpRights:Marked>True</xmpRights:Marked>
         <xmpRights:WebStatement>
           http://creativecommons.org/licenses/by-nc-nd/3.0/
         </xmpRights:WebStatement>
      </rdf:Description>
      <rdf:Description rdf:about=""
            xmlns:dc="http://purl.org/dc/elements/1.1/">
         <dc:format>application/pdf</dc:format>
         <dc:title>
            <rdf:Alt>
               <rdf:li xml:lang="en">
                 On a heuristic viewpoint concerning the production and
                 transformation of light
               </rdf:li>
               <rdf:li xml:lang="x-default">
                 On a heuristic viewpoint concerning the production and
                 transformation of light
               </rdf:li>
            </rdf:Alt>
         </dc:title>
         <dc:description>
            <rdf:Alt>
               <rdf:li xml:lang="en">photoelectric effect</rdf:li>
               <rdf:li xml:lang="x-default">photoelectric effect</rdf:li>
            </rdf:Alt>
         </dc:description>
         <dc:rights>
            <rdf:Alt>
               <rdf:li xml:lang="en">
                 Copyright (C) 1905, Albert Einstein
```

```
            </rdf:li>
            <rdf:li xml:lang="x-default">
              Copyright (C) 1905, Albert Einstein
            </rdf:li>
         </rdf:Alt>
      </dc:rights>
      <dc:creator>
         <rdf:Seq>
            <rdf:li>Albert Einstein</rdf:li>
         </rdf:Seq>
      </dc:creator>
      <dc:subject>
         <rdf:Bag>
            <rdf:li>energy quanta</rdf:li>
            <rdf:li>Hertz effect</rdf:li>
            <rdf:li>quantum physics</rdf:li>
         </rdf:Bag>
      </dc:subject>
      <dc:date>
         <rdf:Seq>
            <rdf:li>1905-03-17</rdf:li>
         </rdf:Seq>
      </dc:date>
      <dc:language>
         <rdf:Bag>
            <rdf:li>en</rdf:li>
         </rdf:Bag>
       </dc:language>
       <dc:type>
          <rdf:Bag>
             <rdf:li>Text</rdf:li>
          </rdf:Bag>
       </dc:type>
      <dc:source>einstein.tex</dc:source>
   </rdf:Description>
   <rdf:Description rdf:about=""
         xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/">
      <photoshop:AuthorsPosition>
        Technical Assistant, Level III
      </photoshop:AuthorsPosition>
      <photoshop:CaptionWriter>Scott Pakin</photoshop:CaptionWriter>
   </rdf:Description>
   <rdf:Description rdf:about=""
       xmlns:pdfaExtension="http://www.aiim.org/pdfa/ns/extension/"
       xmlns:pdfaSchema="http://www.aiim.org/pdfa/ns/schema#"
       xmlns:pdfaProperty="http://www.aiim.org/pdfa/ns/property#"
```

```
  xmlns:pdfaType="http://www.aiim.org/pdfa/ns/type#"
  xmlns:pdfaField="http://www.aiim.org/pdfa/ns/field#">
<pdfaExtension:schemas>
  <rdf:Bag>
    <rdf:li rdf:parseType="Resource">
      <pdfaSchema:schema>IPTC Core Schema</pdfaSchema:schema>
      <pdfaSchema:namespaceURI>http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/</
      <pdfaSchema:prefix>Iptc4xmpCore</pdfaSchema:prefix>
      <pdfaSchema:property>
        <rdf:Seq>
          <rdf:li rdf:parseType="Resource">
            <pdfaProperty:name>CreatorContactInfo</pdfaProperty:name>
            <pdfaProperty:valueType>contactinfo</pdfaProperty:valueType>
            <pdfaProperty:category>external</pdfaProperty:category>
            <pdfaProperty:description>contact information for the document's
          </rdf:li>
        </rdf:Seq>
      </pdfaSchema:property>
      <pdfaSchema:valueType>
        <rdf:Seq>
          <rdf:li rdf:parseType="Resource">
            <pdfaType:type>contactinfo</pdfaType:type>
            <pdfaType:namespaceURI>http://iptc.org/std/Iptc4xmpCore/1.0/xmln
            <pdfaType:prefix>Iptc4xmpCore</pdfaType:prefix>
            <pdfaType:description>contact information</pdfaType:description>
            <pdfaType:field>
              <rdf:Seq>
                <rdf:li rdf:parseType="Resource">
                  <pdfaField:name>CiAdrExtadr</pdfaField:name>
                  <pdfaField:valueType>Text</pdfaField:valueType>
                  <pdfaField:description>contact address</pdfaField:descript
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                  <pdfaField:name>CiAdrCity</pdfaField:name>
                  <pdfaField:valueType>Text</pdfaField:valueType>
                  <pdfaField:description>contact city</pdfaField:description
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                  <pdfaField:name>CiAdrRegion</pdfaField:name>
                  <pdfaField:valueType>Text</pdfaField:valueType>
                  <pdfaField:description>contact region</pdfaField:descripti
                </rdf:li>
                <rdf:li rdf:parseType="Resource">
                  <pdfaField:name>CiAdrPcode</pdfaField:name>
                  <pdfaField:valueType>Text</pdfaField:valueType>
                  <pdfaField:description>contact postal code</pdfaField:desc
```

```
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                      <pdfaField:name>CiAdrCtry</pdfaField:name>
                      <pdfaField:valueType>Text</pdfaField:valueType>
                      <pdfaField:description>contact country</pdfaField:descript
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                      <pdfaField:name>CiTelWork</pdfaField:name>
                      <pdfaField:valueType>Text</pdfaField:valueType>
                      <pdfaField:description>contact telephone number</pdfaFiel
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                      <pdfaField:name>CiEmailWork</pdfaField:name>
                      <pdfaField:valueType>Text</pdfaField:valueType>
                      <pdfaField:description>contact email address</pdfaField:de
                    </rdf:li>
                    <rdf:li rdf:parseType="Resource">
                      <pdfaField:name>CiUrlWork</pdfaField:name>
                      <pdfaField:valueType>Text</pdfaField:valueType>
                      <pdfaField:description>contact url</pdfaField:description>
                    </rdf:li>
                  </rdf:Seq>
                </pdfaType:field>
              </rdf:li>
            </rdf:Seq>
          </pdfaSchema:valueType>
        </rdf:li>
      </rdf:Bag>
    </pdfaExtension:schemas>
  </rdf:Description>
  <rdf:Description rdf:about=""
       xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/">
   <Iptc4xmpCore:CreatorContactInfo rdf:parseType="Resource">
     <Iptc4xmpCore:CiAdrExtadr>Kramgasse 49</Iptc4xmpCore:CiAdrExtadr>
     <Iptc4xmpCore:CiAdrCity>Bern</Iptc4xmpCore:CiAdrCity>
     <Iptc4xmpCore:CiAdrPcode>3011</Iptc4xmpCore:CiAdrPcode>
     <Iptc4xmpCore:CiAdrCtry>Switzerland</Iptc4xmpCore:CiAdrCtry>
     <Iptc4xmpCore:CiTelWork>031 312 00 91</Iptc4xmpCore:CiTelWork>
     <Iptc4xmpCore:CiEmailWork>aeinstein@ipi.ch</Iptc4xmpCore:CiEmailWork>
     <Iptc4xmpCore:CiUrlWork>
       http://einstein.biz/,
       https://www.facebook.com/AlbertEinstein
     </Iptc4xmpCore:CiUrlWork>
   </Iptc4xmpCore:CreatorContactInfo>
  </rdf:Description>
  <rdf:Description rdf:about=""
```

```
            xmlns:xmp="http://ns.adobe.com/xap/1.0/">
         <xmp:CreateDate>2017-02-23T00:41:10-07:00</xmp:CreateDate>
         <xmp:ModifyDate>2017-02-23T00:41:10-07:00</xmp:ModifyDate>
         <xmp:MetadataDate>2017-02-23T00:41:10-07:00</xmp:MetadataDate>
         <xmp:CreatorTool>LaTeX with hyperref package</xmp:CreatorTool>
         <xmp:BaseURL>
           http://mirror.ctan.org/macros/latex/contrib/hyperxmp/
         </xmp:BaseURL>
       </rdf:Description>
       <rdf:Description rdf:about=""
            xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/">
         <xmpMM:DocumentID>
           uuid:0595fdce-41dc-e4c4-6c418dc4ce46
         </xmpMM:DocumentID>
         <xmpMM:InstanceID>
           uuid:efd754c4-1d7f-200a-ef754ce413ea
         </xmpMM:InstanceID>
       </rdf:Description>
    </rdf:RDF>
  </x:xmpmeta>
  <?xpacket end="w"?>
```

# References


[1] Adobe Systems, Inc., San Jose, California. *Adobe Acrobat X SDK Help, pdfmark Reference*. Available from `http://www.adobe.com/devnet/acrobat/documentation.html`.

[2] Adobe Systems, Inc. *PostScript Language Reference Manual*. Addison-Wesley, 2nd edition, January 1996, ISBN: 0-201-18127-4.

[3] Adobe Systems, Inc., San Jose, California. *Document Management—Portable Document Format—Part 1: PDF 1.7*, July 2008. ISO 32000-1 standard document. Available from `http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf`.

[4] Adobe Systems, Inc., San Jose, California. *XMP Specification Part 1: Data model, Serialization, and Core Properties*, April 2012. Available from `http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/cc-201306/XMPSpecificationPart1.pdf`.

[5] DCMI Usage Board *DCMI Metadata Terms*, June 14, 2012. Available from `http://dublincore.org/documents/dcmi-terms/`.

[6] Michael Downes. Around the bend #15, answers, 4th (last) installment. `comp.text.tex` newsgroup posting, January 3, 1994.

Archived by Google at http://groups.google.com/group/comp.text.tex/msg/7da7643b9e8f3b48.

[7] International Press Telecommunications Council. *IPTC Photo Metadata: Core 1.1/Extension 1.1*, July 2010. Revision 1. Available from http://www.iptc.org/std/photometadata/specification/IPTC-PhotoMetadata-201007_1.pdf.

[8] Internet Assigned Numbers Authority. Language subtag registry, January 11, 2011. Available from http://www.iana.org/assignments/language-subtag-registry.

[9] Paul J. Leach, Michael Mealling, and Rich Salz. A Universally Unique IDentifier (UUID) URN namespace. Request for Comments 4122, Internet Engineering Task Force, Network Working Group, July 2005. Category: Standards Track. Available from http://www.ietf.org/rfc/rfc4122.txt.

[10] PDF/A Competence Center, Berlin, Germany. *TechNote 0008: Predefined XMP Properties in PDF/A-1*, March 20, 2008. Available from http://www.pdfa.org/wp-content/uploads/2011/08/tn0008_predefined_xmp_properties_in_pdfa-1_2008-03-20.pdf.

[11] PDF/A Competence Center, Berlin, Germany. *TechNote 0009: XMP Extension Schemas in PDF/A-1*, March 20, 2008. Available from http://www.pdfa.org/wp-content/uploads/2011/08/tn0009_xmp_extension_schemas_in_pdfa-1_2008-03-20.pdf.

[12] Misha Wolf and Charles Wicksteed. Date and time formats. Note NOTE-datetime, World Wide Web Consortium (W3C), September 15, 1997. Available from http://www.w3.org/TR/NOTE-datetime.

# Change History

# Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

## I



## J



## K