The hyperxmp package*

Scott Pakin
scott+hyxmp@pakin.org

November 22, 2020

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 [5]—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—named in the LaTeX source in the usual way: "`\author{Jack Napier \and Edward Nigma \and Harvey Dent}`". With hyperxmp, 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>
```

*This document corresponds to hyperxmp v5.9, dated 2020/11/22.

```
      <rdf:li>Harvey Dent</rdf:li>
    </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 computer applications to identify and categorize the document.

1.1 Supported metadata

hyperxmp knows how to embed all of the following types of metadata within a document:

- address of primary author (Iptc4xmpCore:CreatorContactInfo.CiAdrExtadr, Iptc4xmpCore:CreatorContactInfo.CiAdrCity, Iptc4xmpCore:CreatorContactInfo.CiAdrRegion, Iptc4xmpCore:CreatorContactInfo.CiAdrPcode, and Iptc4xmpCore:CreatorContactInfo.CiAdrCtry)
- author(s) (dc:creator)
- base URL for relative references (xmp:BaseURL)
- book edition (prism:bookEdition)
- copyright (dc:rights and xmpRights:Marked)
- date (dc:date, xmp:CreateDate, xmp:ModifyDate, and xmp:MetadataDate)
- DOI (prism:doi)
- email address(es) of primary author (Iptc4xmpCore:CreatorContactInfo.CiEmailWork)
- file format (dc:format)
- file name of main LaTeX source file (dc:source)
- file size in bytes (prism:byteCount)
- ISBN (prism:isbn)
- ISSN—both print (prism:issn) and electronic (prism:elssn)
- issue number of parent publication (prism:number)

- journal article version (jav:journal_article_version)
- keywords (pdf:Keywords and dc:subject)
- language used (dc:language)
- license URL (xmpRights:WebStatement)
- metadata writer (photoshop:CaptionWriter)
- page count (prism:pageCount)
- page range(s) (prism:pageRange)
- PDF version (pdf:PDFVersion)
- PDF-generating tool (pdf:Producer and xmp:CreatorTool)
- PDF/A version and conformance level (pdfaid:part and pdfaid:conformance)
- PDF/UA version (pdfuaid:part)
- PDF/X standard compliance (pdfxid:GTS_PDFXVersion)
- position/title of primary author (photoshop:AuthorsPosition)
- publication name of parent publication (prism:publicationName)
- publisher of the document (dc:publisher)
- rendition variation of the document (xmpMM:RenditionClass)
- summary (dc:description)
- subtitle (prism:subtitle)
- telephone number(s) of primary author (Iptc4xmpCore:CreatorContactInfo.CiTelWork)
- title (dc:title)
- trapping of colors (pdf:trapped)
- type of document (dc:type)
- type of parent publication (prism:aggregationType)
- unique identifier for the document (dc:identifier)
- URL of the document (prism:url)
- URL(s) of the primary author (Iptc4xmpCore:CreatorContactInfo.CiUrlWork)
- UUID for the document (xmpMM:DocumentID)

```
\Title{Baking through the ages}
\Author{A. Baker\sep C. Kneader}
\Language{en-GB}
\Keywords{cookies\sep muffins\sep cakes}
\Publisher{Baking International}
```

(a) pdfx (separate `.xmpdata` file)

```
\hypersetup{%
  pdftitle={Baking through the ages},
  pdfauthor={A. Baker, C. Kneader},
  pdflang={en-GB},
  pdfkeywords={cookies, muffins, cakes},
  pdfpublisher={Baking International}
}
```

(b) hyperxmp (main document)

Figure 1: Comparison of pdfx and hyperxmp

- UUID for the document instance (xmpMM:InstanceID)
- version identifier for the document (xmpMM:VersionID)
- volume number of parent publication (prism:volume)

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

1.2 Comparisons with similar packages

xmpincl In short, 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.

pdfx The main difference between hyperxmp and pdfx is that hyperxmp tries to integrate as seamlessly as possible into an existing document. It leverages hyperref's `\hypersetup` command and many of `\hypersetup`'s options and defines its own options in a compatible manner. In contrast, pdfx requires the user to create a separate `\jobname.xmpdata` file containing pdfx-defined commands for each metadata element.

Figure 1 adapts an example appearing in the pdfx manual to hyperxmp. The two are comparable line-by-line in terms of how one specifies the title, author, document language, keywords, and publisher. However, hyperxmp implicitly writes a wealth of additional metadata into the XMP packet such as the document date, creation date, creator tool, file format, PDF version, and unique document and

instance IDs. In fact, if a document omits all of the code shown in Figure 1(b), it will still store the `\title` and `\author` data in the XMP packet.

One can therefore summarize the difference between hyperxmp and pdfx as follows: pdfx requires the author to be fully explicit about the document's metadata while hyperxmp allows some metadata to be specified implicitly, automatically inferring it when possible. In general, hyperxmp tries to simplify the author's task as much as possible.

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
- pdftrapped

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

- pdfaconformance
- pdfapart
- pdfauthortitle
- pdfbookedition
- pdfbytes
- pdfcaptionwriter
- pdfcontactaddress
- pdfcontactcity
- pdfcontactcountry
- pdfcontactemail
- pdfcontactphone
- pdfcontactpostcode
- pdfcontactregion
- pdfcontacturl
- pdfcopyright
- pdfdate
- pdfdocumentid
- pdfdoi
- pdfeissn
- pdfidentifier
- pdfinstanceid
- pdfisbn
- pdfissn
- pdfissuenum
- pdflicenseurl
- pdfmetadate
- pdfmetalang
- pdfnumpages
- pdfpagerange
- pdfpublication
- pdfpublisher
- pdfpubstatus
- pdfpubtype
- pdfrendition
- pdfsource
- pdfsubtitle

- pdftype
- pdfurl
- pdfvolumenum
- pdfuapart
- pdfversionid
- pdfxstandard

2.1 Option descriptions

pdftitle

The document title is specified as normal for hyperref with pdftitle, but see Note 7 on page 16 for instructions on how to specify a title in multiple languages. If pdftitle is not specified it will inherit its value from the document's `\title`. hyperxmp introduces a complementary pdfsubtitle option:

pdfsubtitle

```
pdftitle={Frankenstein},
pdfsubtitle={The Modern Prometheus},
```

Unfortunately, the subtitle can appear in only one language. It assumed to be the same language as the document language (pdflang) but can be overridden by preceding the text with a bracketed ISO 639-1 two-letter language code and an optional ISO 3166-1 two-letter region code. See the example below for pdfpublication.

pdfauthor

hyperref's pdfauthor option specifies the document's author(s). See Note 4 on page 15 for a discussion of the correct syntax. If pdfauthor is not specified it will inherit its value from the document's `\author`. pdfauthortitle indicates the primary author's position or title. pdfcaptionwriter specifies the name of the person who added the metadata to the document.

pdfauthortitle
pdfcaptionwriter

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).

pdfcontactaddress
pdfcontactcity
pdfcontactcountry
pdfcontactemail
pdfcontactphone
pdfcontactpostcode
pdfcontactregion
pdfcontacturl

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

pdfcopyright
pdflicenseurl

pdfmetalang indicates the natural language in which certain metadata—specifically, the document's title, subject, and copyright statement—are written. The language should be specified using an IETF language tag [11], 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.

pdfmetalang
pdflang

XMP can include a universally unique identifier (UUID) for each document and for each instance of a given document. By default, hyperxmp assigns a version 4 (i.e., pseudorandom) UUID [12] for each of these. However, a document can

pdfdocumentid

alternatively specify a particular document identifier using pdfdocumentid and (not normally recommended) a particular instance identifier using pdfinstanceid. These should be of the form `uuid:`*xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*, where "*x*" is a lowercase hexadecimal number. For example, `uuid:53ab7f19-a48c-5177-8bb2-403ad907f632` is a valid argument to pdfdocumentid (or pdfinstanceid). See Leach, Mealling, and Salz's UUID specification document for details on how to produce the various forms of UUIDs [12]. A more freeform mechanism than pdfinstanceid for versioning documents is available via pdfversionid. The version specified by pdfversionid can be incremented as `1`, `2`, `3`, ...; identified with a hierarchical numbering scheme (e.g., this document is versioned `5.9` to match the package version); or labeled using any other approach. One possibility is to use a revision number or commit hash from the version-control software maintaining the document. For example, the `\gitVer` macro from the gitver package is an expandable (see Note 8 on page 17) version of the current Git hash that can suitably be passed to pdfversionid. If not specified, pdfversionid defaults to `1`.

pdfinstanceid

pdfversionid

Already-published documents can be identified in a number of ways. pdfisbn specifies the ISBN. pdfissn refers to the ISSN of the *print* version of the document while pdfeissn refers to the ISSN of the *electronic* version of the document. pdfdoi specifies the DOI and should include only the DOI name without any URL prefix. For example, specify `pdfdoi={10.1145/3149526.3149532}`, *not* `pdfdoi={https://doi.org/10.1145/3149526.3149532}`. pdfurl points to the complete URL for the document. In contrast, baseurl points one level up and is used to resolve relative URLs.

pdfisbn
pdfissn
pdfeissn
pdfdoi
pdfurl
baseurl

pdfidentifier provides an alternative mechanism to uniquely identify a document. Its advantage relative to pdfisbn, pdfissn, pdfdoi, etc. is its flexibility; any of a wide variety of identification types can be used.[1] pdfidentifier's disadvantage is that it allows only a single identifier per document. For example, a document could use `pdfidentifier=urn:iso:std:32000:ed-1:v1:en` to identify itself as version 1 of English-language ISO standard 32000-1, but then this same document could not also use pdfidentifier to identify itself by DOI (`info:doi/...`), ISBN (`urn:ISSN:...`), etc. (It can still use the options described in the previous paragraph, though.) If pdfidentifier is not specified explicitly, hyperxmp will use the first non-empty value out of the DOI, electronic ISSN, print ISSN, and ISBN or skip the identifier entirely if all of those are empty.

pdfidentifier

Already-published documents can further be identified by the publication in which they appear. pdfpublication specifies the title of the journal, magazine, or other parent document. The title language is assumed to be the same as the document language (pdflang) but can be overridden by preceding the text with a bracketed ISO 639-1 two-letter language code and an optional ISO 3166-1 two-letter region code. For example, `pdfpublication={[fr]Charlie Hedbo}` indicates a French-language title. Were the language or pronunciation differences significant, `fr-FR` would indicate specifically the French spoken in France, as opposed to that spoken in, say, Canada (`fr-CA`) or Belgium (`fr-BE`). The publisher itself can be

pdfpublication

[1] See, for example, `https://www.iana.org/assignments/urn-namespaces/urn-namespaces.xhtml` for the `urn:` URI scheme and `http://info-uri.info/registry/` for the `info:` URI scheme.

Table 1: Valid arguments for pdfpubstatus

Value	Meaning
AO	Author's Original
SMUR	Submitted Manuscript Under Review
AM	Accepted Manuscript
P	Proof
VoR	Version of Record
CVoR	Corrected Version of Record
EVoR	Enhanced Version of Record

pdfpublisher named using pdfpublisher.

pdfpubtype pdfpubtype indicates the type of publication in which the document was published. This should be one of the PRISM aggregation types [9] such as `book`, `journal`, `magazine`, `manual`, `report`, or `whitepaper`.

pdfvolumenum pdfissuenum pdfpagerange For publications in journals, magazines, and similar periodicals, a document can specify the volume number with pdfvolumenum and the issue number within the volume with pdfissuenum. pdfpagerange indicates the page numbers at which the document appears within the publication. The intention is that this be a comma-separated list of dash-separated ranges, as in `pdfpagerange={1,4-5}`. See Note 9 on page 17 for advice on how to assign pdfpagerange semi-automatically.

pdfpubstatus A journal article's publication status can be indicated with pdfpubstatus. This option expects to take one of the values listed in Table 1. See the NISO/ALPSP Journal Article Versions recommendation [1] for an explanation of each of those values and when to use them.

pdfbookedition For books, pdfbookedition names the edition of the book. This is specified as text, not a number. As with pdfpublication (above), pdfbookedition accepts a bracketed language code, as in `pdfbookedition={[en]Second edition}`.

pdfdate 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 [5] or PDF format [4]. XMP dates are written in the form YYYY-MM-DDThh:mm:ss+TT:tt.[2] A W3C recommendation [15] 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`, `2014-09-23T14:15`, `2014-09-23`, `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

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

pdfcreationdate
pdfmoddate
pdfmetadate

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.

pdftype

pdftype describes the type of document being produced. This refers to "the nature or genre of the resource" [5] 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 [6]. 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," [6] and other forms of text—all things LaTeX is commonly used to typeset.

pdfrendition

Sometimes a base document is rendered in different forms. pdfrendition indicates the particular rendition the current document instance represents. The value should come from the following controlled vocabulary [5]: `default`, `draft`, `low-res`, `proof`, `screen`, and `thumbnail`. hyperxmp's default value is `default`, which indicates the master document, unless the draft option is passed to `\documentclass`, in which case hyperxmp defaults to `draft`.

pdftrapped

hyperxmp honors hyperref's pdftrapped option. A document can indicate whether it employs color trapping by specifying pdftrapped=`True` or pdftrapped=`False`. (pdftrapped=`Unknown` is also allowed.)

pdfapart
pdfaconformance

pdfapart and pdfaconformance, 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). A document that conforms to the PDF/UA standard can use pdfuapart to indicate the PDF/UA conformance level. For example, pdfuapart=`1` asserts that the document respects PDF/UA-1. pdfxstandard indicates the particular PDF/X standard by which the document abides. Unlike pdfapart and pdfaconformance, which accept a number and a letter, respectively, pdfxstandard expects a textual identification of a standard name. The following are the acceptable PDF/X standard names as of at the time of this writing.

pdfuapart
pdfxstandard

- PDF/X-1a:2001
- PDF/X-1a:2003
- PDF/X-3:2002
- PDF/X-3:2003
- PDF/X-4
- PDF/X-4p
- PDF/X-5g
- PDF/X-5n
- PDF/X-5pg

For example, one can specify pdfxstandard=`{PDF/X-4}` or pdfxstandard=`{PDF/X-3:2003}`, but specifying pdfxstandard=`{PDF/X-3}` will not pass PDF/X validation. Note that at the time of this writing the use of the PDF/X-4p, PDF/X-5n, and PDF/X-5pg standards has not been tested.

Rarely needed options

pdfsource

pdfsource overrides the name of the LaTeX source file. It defaults to `\jobname.tex` but can be replaced by any other string. If pdfsource is given an empty argument, no document source will be specified at all.

pdfnumpages

The number of pages in the published, print version of the document can be expressed with pdfnumpages. This is computed automatically when the document is built using either pdfLaTeX or LuaLaTeX.

pdfbytes

The pdfbytes option expresses the document's file size in bytes. The intention is for this to be used to display an estimate of download time to a user or to serve as a quick check on whether a file was transmitted correctly between systems. pdfbytes is computed automatically by both pdfLaTeX and LuaLaTeX, using the file size from the previous build of the document.

It is usually more convenient to provide values for all of the options presented in this section using hyperref's `\hypersetup` command than on the `\usepackage` command line. See the hyperref manual for more information.

2.2 A complete example

The following is a sample LaTeX document that provides values for most of the metadata options that hyperxmp recognizes:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{hyperxmp}
\usepackage[unicode]{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},
  pdfsubtitle={[en-US]Putting that bum Maxwell in his place},
  pdfauthor={Albert Einstein},
  pdfauthortitle={\xmpquote{Technical Assistant\xmpcomma\ 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
      },
      pdfdocumentid={uuid:6d1ac9ec-4ff2-515a-954b-648eeb4853b0},
      pdfversionid={2.998e8},
      pdfpublication={[de]Annalen der Physik},
      pdfpublisher={Wiley-VCH},
      pdfpubtype={journal},
      pdfvolumenum={322},
      pdfissuenum={6},
      pdfpagerange={132-148},
      pdfissn={0003-3804},
      pdfeissn={1521-3889},
      pdfpubstatus={VoR},
      pdflang={en},
      pdfmetalang={en},
      pdfurl={http://www.physik.uni-augsburg.de/annalen/history/einstein-
papers/1905_17_132-148.pdf},
      pdfdoi={10.1002/andp.19053220607},
      pdfidentifier={info:lccn/50013519}
    }
    \XMPLangAlt{de}{pdftitle={Über einen die Erzeugung und Verwandlung des
      Lichtes betreffenden heuristischen Gesichtspunkt}}

    \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
- XeLaTeX
- LaTeX + Dvipdfm
- LaTeX + Dvips + Adobe Acrobat Distiller

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 2 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 3.

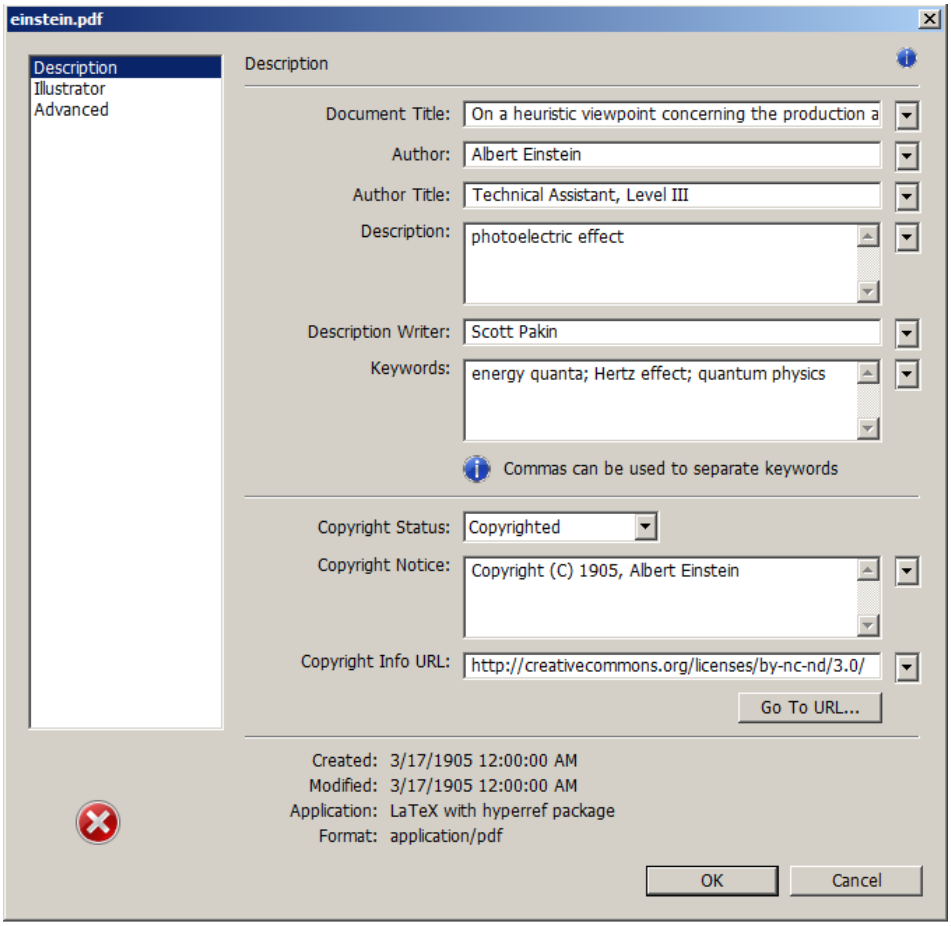

Figure 2: XMP metadata as it appears in Adobe Acrobat

2.3 Usage notes

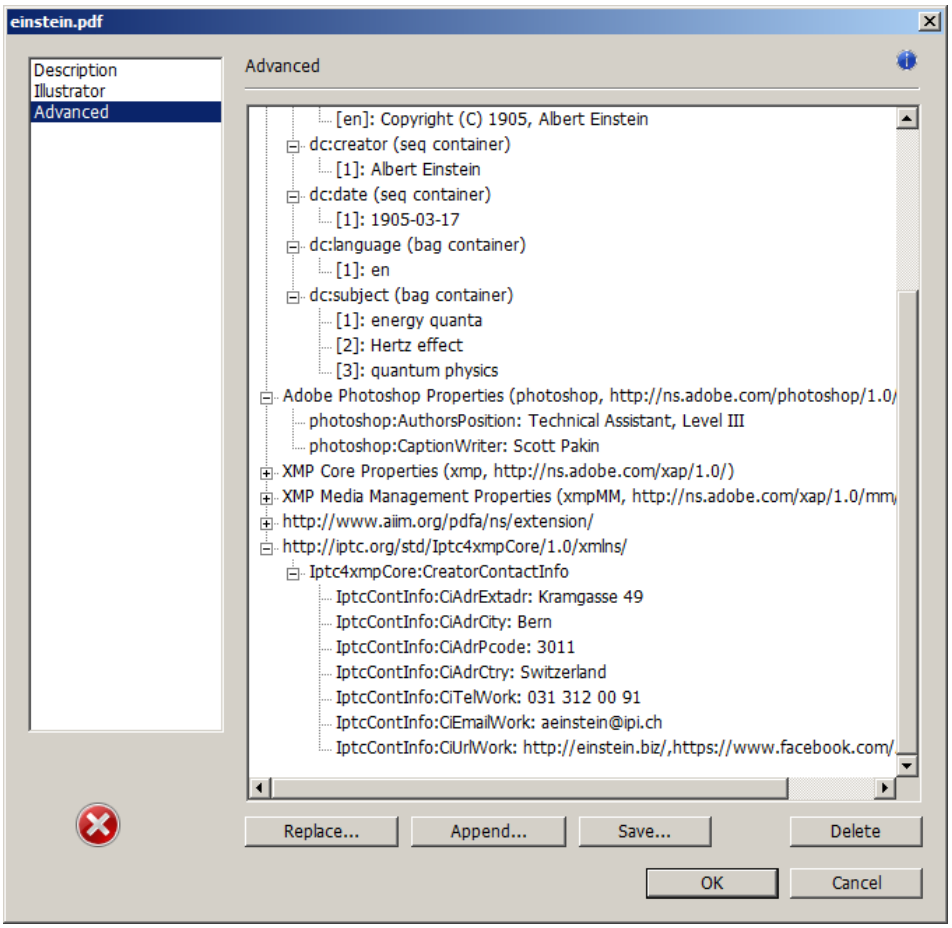

Figure 3: Additional XMP metadata as it appears in Adobe Acrobat

Note 1: Conflicting metadata in PDF/A documents A PDF file includes an Info dictionary containing Author, Title, Subject, and Keywords keys. The hyperref package's pdfauthor, pdftitle, pdfsubject, and pdfkeywords options assign values to those keys. The hyperxmp package additionally uses those options to assign values to various XMP metadata: dc:creator, dc:title, dc:description, and pdf:Keywords. The PDF/A specification indicates that values that appear in both the PDF Info dictionary and XMP packet must match. The problem is that in XMP, the author and keywords can be proper lists, as in

```
<dc:creator>
  <rdf:Seq>
    <rdf:li>Curly Howard</rdf:li>
    <rdf:li>Larry Fine</rdf:li>
    <rdf:li>Moe Howard</rdf:li>
  </rdf:Seq>
</dc:creator>
```

while in PDF, the author and keywords are specified as flat strings. Alas, there is no definition of how a list should be collapsed to a flat string: "`Curly Howard, Larry Fine, Moe Howard`" or "`Curly Howard; Larry Fine; Moe Howard`" or something else. I have not yet found a form of flat string that passes all PDF/A validators. Furthermore, when Adobe Acrobat—at least Adobe Acrobat DC (2019) and earlier versions—converts a PDF file to PDF/A format, it does so by discarding all but the first author, which is an unsatisfying solution.

Starting with version 4.0, hyperxmp's solution is to suppress writing metadata to the PDF Info dictionary and write it only to the XMP packet. (hyperxmp v5.0+ is more sophisticated. It suppresses only the author and keyword lists.) This appears to pacify PDF/A validators yet retains the author and keyword lists in their non-truncated form. If desired, the Info dictionary can be retained by passing the keeppdfinfo option to `\hypersetup`.

keeppdfinfo

Note 2: Acrobat multiline-field bug The IPTC Photo Metadata schema states that "the [contact] address is a multiline field" [10]. hyperxmp converts commas in pdfcontactaddress's argument to line breaks in the generated XML. Unfortunately, A bug in Adobe Acrobat—at least in Adobe Acrobat DC (2019) and earlier versions—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 commas appearing in pdfcontactaddress's argument with semicolons:

\xmplinesep

```
\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 hyperxmp attempts to identify certain metadata automatically. The hope is that in many cases, an author can simply include `\usepackage{hyperxmp}` in a document's preamble and benefit from a modicum of XMP metadata with no additional effort.

Currently, 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. hyperxmp recognizes some class-specific metadata as well, such as that provided via the Koma letter classes (e.g., scrlttr2) and the ACM article class (acmart).

If a document uses either the babel or polyglossia packages it is recommended that it *not* explicitly set pdflang. pdflang accepts only a single language name while hyperxmp can automatically query babel and polyglossia for a list of all languages used in the document and include this list in an XMP dc:language element.

Note 7: Multilingual metadata The pdfmetalang option specifies the language in which the document's metadata is written. It defaults to the value of pdflang,

`\XMPLangAlt` which specifies the document language. As of version 3.3 of hyperxmp, it is possible to include certain metadata—specifically, the document's title, subject, and copyright statement—in more than one language. The `\XMPLangAlt` macro provides this functionality. Usage is as follows:

`\XMPLangAlt {`⟨*language*⟩`} {` ⟨*option*⟩`=`⟨*text*⟩, ... `}`

where ⟨*language*⟩ is an ISO 639-1 two-letter country code with an optional ISO 3166-1 two-letter region code (e.g., "`en`" for English or "`en-US`" for specifically US English); ⟨*option*⟩ is one of "`pdftitle`", "`pdfsubject`", or "`pdfcopyright`"; and ⟨*text*⟩ is the text as expressed in the specified language. By way, of example, the following code provides the document title in English then specifies an alternative title to use in four other languages:

```
\hypersetup{%
```

```
  pdfmetalang={en},
  pdftitle={English title}
}
\XMPLangAlt{de}{pdftitle={Deutscher Titel}}
\XMPLangAlt{fr}{pdftitle={Titre fran\c{c}ais}}
\XMPLangAlt{it}{pdftitle={Titolo italiano}}
\XMPLangAlt{rm}{pdftitle={Titel rumantsch}}
```

Note 8: Expandable arguments All arguments passed to hyperxmp options must be expandable, in TEX terminology. This implies that any macros that are used in arguments are limited to a relatively small set of operations (such as conditionals and macro expansion) and must produce a string of text. Code (such as macro definitions and arithmetic operations) will be written to XMP as code, not as the result of executing the code.

By way of example, the macros provided by the texdate package for typesetting dates are not expandable (at least at the time of this writing). Hence, the `\printfdate{Y}` in the following code snippet is not replaced by the current year, as one might expect:

```
\usepackage{texdate}
\initcurrdate
\hypersetup{%
  pdfcopyright={Copyright \textcopyright\ \printfdate{Y}, Scott Pakin}
}
```

Rather, it generates a dc:rights tag of the form "`Copyright © =2=0=by-1by=02020, Scott Pakin`". The garbage in that line corresponds to the remnants of the `\printfdate` code after expanding all of the TEX primitives and certain other control sequences it uses to the empty string. For example, "`\global\advance\texd@yr by-1`" expands to "`by-1`".

It is not possible to determine a priori whether or not a macro is expandable. The best advice is to carefully inspect the XMP package in the output file to ensure that any macros used in arguments to hyperxmp options produced the expected output.

Note 9: Semi-automatic page ranges Although pdfpagerange is intended to refer to pages in the final, published version of a document, it would be convenient for them to be generated automatically when producing a standalone PDF file that is not intended to be incorporated into a book, journal, or other publication (or if it is known that the pages will not be renumbered for publication). One approach is to use the totpages package help generate pdfpagerange. For documents numbered from 1 to n, a simple

```
\hypersetup{%
  pdfpagerange={1-\ref*{TotPages}}
}
```

should suffice. A bit more effort is needed for documents that change numbering schemes, such as using lowercase Roman numerals for the front matter and Arabic numerals for the main matter and back matter. One approach is to use `\label` to mark the first and last page of each numbering scheme and specify pdfpagerange as in the following:

```
\hypersetup{%
  pdfpagerange={%
    \pageref*{page:begin-front}-\pageref*{page:end-front},%
    1-\pageref*{TotPages}%
  }
}
```

I don't know how unnumbered pages (e.g., blank pages and the title page) are supposed to be handled. I suppose blank pages can be omitted from pdfpagerange, and the title page can be either omitted or listed as `title`, for example.

It appears that at least with version 2.00 of totpages, the `TotPages` label is not defined until after the `\begin{document}`. Consequently, using `TotPages` within a `\hypersetup` invocation in the document's preamble will produce "`??`" as the page count in the XMP packet. The solution is either to assign pdfpagerange after the `\begin{document}` or to ask LaTeX to do that on your behalf:

```
\AtBeginDocument{%
  \hypersetup{%
    pdfpagerange={1-\ref*{TotPages}}
  }%
}
```

Note 10: Automatic computation of the PDF byte count The PRISM Basic Metadata schema [8] defines a prism:byteCount property that indicates the PDF file size in bytes. hyperxmp computes this value automatically when the document is built using LuaLaTeX but not when using any other TeX engine. Note that hyperxmp uses the file size from the *previous* run of LuaLaTeX because the new PDF file is not yet complete. Consequently, one extra compilation is needed for the byte count to converge relative to the the number of compilations that would otherwise be required.

Starting with hyperxmp v5.9, the hyperxmp distribution includes a Perl script called `hyperxmp-add-bytecount` that edits a PDF file in place, adding or replacing the prism:byteCount property with one that specifies the final file size.[3] Run the script as "`hyperxmp-add-bytecount` ⟨*filename.pdf*⟩".

[3]The script was in fact introduced with `hyperxmp` v5.8 and was then called `add_byteCount`.

```
foreach my $cmd ( "latex", "lualatex", "pdflatex", "xelatex",
                  "dvipdf", "xdvipdfmx", "ps2pdf" ) {
    ${$cmd} = "internal mycmd ${$cmd}";
}

sub mycmd {
    my $retval = system @_;
    if ( $$Pdest =~ /\.pdf$/ ) {
        system 'hyperxmp-add-bytecount', $$Pdest;
    }
    return $retval;
}
```

Figure 4: `latexmk` configuration-file code for automatically invoking `hyperxmp-add-bytecount` every time a PDF file is generated

The `latexmk` build tool can be configured to run `hyperxmp-add-bytecount` automatically every time a PDF file is generated. Simply add the code shown in Figure 4 to your `latexmk` configuration file. See the `latexmk` manual for information on configuration-file naming on different operating systems and explanations of the hook functions used in Figure 4.

Even though hyperxmp can compute the byte count automatically when run from LuaLaTeX, users of `latexmk` need to use configuration-file code like that shown in Figure 4. Otherwise, `latexmk` would compile the document one time too few for the byte count to converge. It is recommended that those who use both `latexmk` and hyperxmp configure `latexmk` to be hyperxmp-aware.

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.

One thing to bear in mind when reading the hyperxmp source code is that different actions occur at different times throughout document processing:

1. `\usepackage{hyperxmp}`: hyperxmp parses package options, defines a number of commands, loads various helper packages, and assigns default values to most XMP fields.

2. `\begin{document}`: hyperxmp loads certain packages such as hyperref and ifdraft and queries natural-language information from babel and polyglossia that becomes available only at the end of the preamble.

3. `\end{document}`: hyperxmp finalizes certain data that are known only at the end of the document, such as the page count, and writes the XMP packet to the PDF file.

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 3.8).

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

\hyxmp@at@end The \hyxmp@at@end macro includes code at the end of the document. When available (as is the case in most modern TEX backends), \AtEndDocument works well enough. Otherwise, we invoke \AtEndDvi from the atenddvi package, which is robust but requires an addition LATEX run.

```
\@ifundefined{AtEndDocument}{%
  \RequirePackage{atenddvi}
  \let\hyxmp@at@end=\AtEndDvi
}{%
  \let\hyxmp@at@end=\AtEndDocument
}
```

\hyxmp@set@jobname Given an expanded \jobname followed by \relax, invoke the \hyxmp@set@jobname@dbl macro if the job name is surrounded by double quotes and the \hyxmp@set@jobname@plain macro otherwise.

```
\def\hyxmp@set@jobname#1\relax{%
  \@ifnextchar"{\hyxmp@set@jobname@dbl}{\hyxmp@set@jobname@plain}#1\relax
}
```

\hyxmp@set@jobname@dbl Set \hyxmp@jobname to to #1, discarding the surrounding double quotes.
\hyxmp@jobname

```
\def\hyxmp@set@jobname@dbl"#1"\relax{\xdef\hyxmp@jobname{#1}}
```

\hyxmp@set@jobname@plain Set \hyxmp@jobname to to #1.
\hyxmp@jobname

```
\def\hyxmp@set@jobname@plain#1\relax{\xdef\hyxmp@jobname{#1}}
```

Define \hyxmp@jobname as a sanitized version of \jobname. The problem with using \jobname directly is that it surrounds the filename with double quotes if it contains a space character. For example, a source file named my-file.tex results in a \jobname of "my-file", but a source file named my file.tex results in a \jobname of ""my file"". Trying to access "my file".log (as is done on page 51) will fail because the filename does not in fact contain literal double quotes.

```
\expandafter\hyxmp@set@jobname\jobname\relax
```

\hyxmp@aep@toks In order for hyperxmp to be loaded safely during \AtEndPreamble we need to ensure that we perform no \AtEndPreamble actions until all top-level macro definitions have been made. The most straightforward approach would be to move all of hyperxmp's \AtEndPreamble stanzas to the end of the package. However, this degrades readability of the source code. For instance, an \AtEndPreamble stanza

related to integration with hyperref could no longer appear in the "Integration with hyperref" section (Section 3.2). Hence, we instead store in a token list, `\hyxmp@aep@toks`, each `\AtEndPreamble` stanza as we encounter it. This token list is evaluated as one of the package's final actions (Section 3.8).

```
\newtoks{\hyxmp@aep@toks}
```

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 XMP metadata from hyperref's baseurl, pdfauthor, pdfkeywords, pdflang, pdfproducer, pdfsubject, pdftrapped, and pdftitle options. It also introduces a number of new options, which are listed on pages 5–6. 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), iftex for determining which TeX engine is being used, ifmtarg for testing if a macro argument is empty or all spaces, etoolbox for dynamically patching existing commands (specifically, hyperref's `\PDF@FinishDoc`), and ifthen for convenient string comparisons.

```
\RequirePackage{kvoptions}
\RequirePackage{pdfescape}
\RequirePackage{stringenc}
\RequirePackage{intcalc}
\RequirePackage{iftex}
\RequirePackage{ifmtarg}
\RequirePackage{etoolbox}
\RequirePackage{ifthen}
```

There are a few places where hyperxmp can take advantage of LuaTeX features. To simplify the use of LuaTeX we load the luacode package.

```
\ifLuaTeX
  \RequirePackage{luacode}
\fi
```

`\@ifmtargexp` `\@ifnotmtargexp`

`\@ifmtarg` and `\@ifnotmtarg` do not expand their first argument. Define `\@ifmtargexp` and `\@ifnotmtargexp` as expanding versions of those macros.

```
\def\@ifmtargexp#1{\expandafter\@ifmtarg\expandafter{#1}}
\def\@ifnotmtargexp#1{\expandafter\@ifnotmtarg\expandafter{#1}}
```

`\@if@def@and@nonempty`

This macro combines `\@ifundefined` and `\@ifmtargexp`. If the macro named `#1` is both defined and non-empty, evaluate `#2`. Otherwise, evaluate `#3`.

```
\newcommand*{\@if@def@and@nonempty}[3]{%
```

```
  \@ifundefined{#1}{#3}{%
    \expandafter\@ifmtargexp\expandafter{\csname#1\endcsname}{#3}{#2}%
  }%
}
```

`\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 (see Section 3.4.2) 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 (see Section 3.4.2) 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}}
```

`\hyxmp@no@bad@parts` Complain about a bad pdfapart or pdfuapart if given trailing non-digits after a part number.

```
\def\hyxmp@no@bad@parts#1\relax{%
  \@ifnotmtarg{#1}{%
    \PackageWarning{hyperxmp}{pdfapart and pdfuapart must be numeric}%
  }%
}
```

`\@pdfapart` Prepare to store the PDF/A part ID, which defaults to "1" if pdfa is passed to hyperref.

```
\def\@pdfapart{}
\define@key{Hyp}{pdfapart}{%
  \afterassignment\hyxmp@no@bad@parts\@tempcnta=0#1\relax
  \hyxmp@pdfstringdef\@pdfapart{\the\@tempcnta}%
}
```

`\@pdfaconformance` Prepare to store the PDF/A conformance ID, which defaults to "b" if pdfa is passed to hyperref and `\@pdfapart` is empty.

```
\def\@pdfaconformance{}
```

```
\define@key{Hyp}{pdfaconformance}{%
  \uppercase{\hyxmp@pdfstringdef\@pdfaconformance{#1}}%
}
```

`\@pdfuapart` Prepare to store the PDF/UA part ID.

```
\def\@pdfuapart{}
\define@key{Hyp}{pdfuapart}{%
  \afterassignment\hyxmp@no@bad@parts\@tempcnta=0#1\relax
  \hyxmp@pdfstringdef\@pdfuapart{\the\@tempcnta}%
}
```

`\hyxmp@set@pdfx@major` Parse pdfxstandard as "PDF/X-⟨*major*⟩⟨*other*⟩", setting `\hyxmp@pdfx@major` to ⟨*major*⟩.

```
\newcommand*{\hyxmp@set@pdfx@major}[1]{\hyxmp@set@pdfx@major@i#1!}
```

`\hyxmp@set@pdfx@major@i` This is the first helper macro for `\hyxmp@set@pdfx@major`. It stores the PDF/X major version in `\@tempcnta`.

```
\def\hyxmp@set@pdfx@major@i PDF/X-{%
  \afterassignment\hyxmp@set@pdfx@major@ii
  \@tempcnta=%
}
```

`\hyxmp@set@pdfx@major@ii` `\hyxmp@pdfx@major` This is the second helper macro for `\hyxmp@set@pdfx@major`. It copies the PDF/X major version from `\@tempcnta` to `\@hyxmp@pdfx@major` and discards the rest of the PDF/X standard string.

```
\def\hyxmp@set@pdfx@major@ii#1!{%
  \edef\hyxmp@pdfx@major{\the\@tempcnta}%
}
```

`\hyxmp@check@std` Compare a user-provided string to a fixed string. (Assumption: Both are names of PDF/X standard versions.) If they match, undefine `\next`, which we assume was previously defined to issue an "unrecognized standard" warning message.

```
\newcommand*\hyxmp@check@std[2]{%
  \ifthenelse{\equal{#1}{#2}}%
             {\global\let\next=\relax}%
             {}%
}%
```

`\@pdfxstandard` Prepare to store the PDF/X standard.

```
\def\@pdfxstandard{}
\def\hyxmp@pdfx@major{}
\define@key{Hyp}{pdfxstandard}{%
  \hyxmp@pdfstringdef\@pdfxstandard{#1}%
```

`\next` Issue a warning message if the PDF/X standard named by the user does not appear in a list of known PDF/X standards. This is to caution the user that hyperxmp generates standard-specific XMP metadata and it can only guess at the correct

format for new standard versions. (See the comments on page 74 above the definition of \hyxmp@pdfx@id@schema, for example.)

```
  \gdef\next{%
    \PackageWarning{hyperxmp}{Unrecognized PDF/X standard `#1'}%
  }%
  \hyxmp@check@std{#1}{PDF/X-1a:2001}%
  \hyxmp@check@std{#1}{PDF/X-1a:2003}%
  \hyxmp@check@std{#1}{PDF/X-3:2002}%
  \hyxmp@check@std{#1}{PDF/X-3:2003}%
  \hyxmp@check@std{#1}{PDF/X-4}%
  \hyxmp@check@std{#1}{PDF/X-4p}%
  \hyxmp@check@std{#1}{PDF/X-5g}%
  \hyxmp@check@std{#1}{PDF/X-5n}%
  \hyxmp@check@std{#1}{PDF/X-5pg}%
  \next
```

\hyxmp@pdfx@major Parse the PDF/X major version number from pdfxstandard and assign it to \hyxmp@pdfx@major.

```
  \hyxmp@set@pdfx@major{#1}%
}
```

\@pdfsource Prepare to store the document's source, which defaults to the value of \jobname.

```
\edef\@pdfsource{\hyxmp@jobname.tex}
\define@key{Hyp}{pdfsource}{\hyxmp@pdfstringdef\@pdfsource{#1}}
```

\hyxmp@DocumentID Prepare to store a UUID that represents the document.

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

\hyxmp@InstanceID Prepare to store a UUID that represents the current instance of the document.

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

\@pdfversionid Prepare to store a string that represents the current version of the document. It defaults to "1".

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

\ifdraft \next Use the ifdraft package to determine if this is a draft or final document. The challenge here is that we want to use ifdraft if it's already loaded, load it if not, and not break any incompatible, author-defined \ifdraft macros that may occur either before or after the \usepackage{hyperxmp}. Our solution begins by defining a new group. Then, if ifdraft is not yet loaded, we locally undefine \ifdraft and load the package. In this case, we later "unload" the package by setting \ver@ifdraft.sty to \relax.

```
\begingroup
  \@ifpackageloaded{ifdraft}{%
    \let\next=\relax
```

```
  }{%
    \let\ifdraft=\relax
    \RequirePackage{ifdraft}%
    \def\next{%
      \expandafter\global\expandafter\let\csname ver@ifdraft.sty\endcsname=\relax
    }%
  }%
```

`\@pdfrendition` Prepare to store a tag describing how this rendition of the document differs from the master. The default value is `default`, which indicates the master document, except in the case of `\documentclass[draft]`, for which `\@pdfrendition` defaults to `draft`.

```
  \ifdraft{%
    \gdef\@pdfrendition{draft}%
  }{%
    \gdef\@pdfrendition{default}%
  }
  \next
\endgroup
\define@key{Hyp}{pdfrendition}{\hyxmp@pdfstringdef\@pdfrendition{#1}}
```

`\@pdfpublication` Prepare to store the name of the publication in which the document was published.

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

`\@pdfpubtype` Prepare to store the type of the publication in which the document was published.

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

`\@pdfbytes` Prepare to store the size of the file in bytes.

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

`\@pdfnumpages` Prepare to store the number of pages in the file.

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

`\@pdfissn` Prepare to store the ISSN of the publication in which the document was published.

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

`\@pdfeissn` Prepare to store the ISSN of the electronic version of the publication in which the document was published.

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

`\@pdfisbn` Prepare to store the ISBN of the publication in which the document was published.

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

`\@pdfbookedition` Prepare to store the edition of the book in which the document was published.

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

`\@pdfpublisher` Prepare to store the name of the document's publisher.

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

`\@pdfvolumenum` Prepare to store the volume identifier of the publication in which the document was published.

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

`\@pdfissuenum` Prepare to store the identifier of the issue within a volume of the publication in which the document was published.

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

`\@pdfpagerange` Prepare to store the document's range of pages within the publication in which the document was published.

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

`\@pdfdoi` Prepare to store a DOI that represents the current instance of the document.

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

`\@pdfurl` Prepare to store a URL that represents where the document can be found. Note that we do not prepend baseurl to the value provided.

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

`\@pdfidentifier` Prepare to store an identifier that uniquely represents the document.

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

`\@pdfsubtitle` Prepare to store the document's subtitle.

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

`\@pdfpubstatus` Prepare to store the document's journal article version.

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

`\hyxmp@no@info@lists` Suppress hyperref from writing Author and Keywords into the Info dictionary. This prevents conflicts between the PDF metadata and the XMP metadata that cause PDF/A validation to fail. The PDF metadata can be restored by passing the keeppdfinfo option to `\hypersetup`.

```
\def\hyxmp@no@info@lists{%
```

`\hyxmp@suppress@pdf@info` `\next` If `\patchcmd` fails for any reason—most likely, a modification to the hyperref package—our fallback is to prevent hyperref from writing *any* data to the PDF Info dictionary.

```
  \def\hyxmp@suppress@pdf@info{%
    \global\let\PDF@FinishDoc=\@empty
    \PackageWarningNoLine{hyperxmp}{%
      Suppressing the _entire_ PDF Info dictionary.\MessageBreak
      Please notify the hyperxmp maintainer%
    }%
  }%
  \let\next=\relax
  \patchcmd
    {\PDF@FinishDoc}%
    {/Author(\@pdfauthor)}%
    {}%
    {}%
    {\let\next=\hyxmp@suppress@pdf@info}%
  \patchcmd
    {\PDF@FinishDoc}%
    {/Keywords(\@pdfkeywords)}%
    {}%
    {}%
    {\let\next=\hyxmp@suppress@pdf@info}%
  \next
}

\define@key{Hyp}{keeppdfinfo}[true]{%
  \gdef\hyxmp@no@info@lists{}%
}
```

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
\xmpcomma
\xmpquote
\hyxmp@pdfkeywords
\@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).

```
\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@concated@metadata
\hyxmp@aep@toks

Assume that if the document loaded either babel or polyglossia it will eventually define one or more languages that hyperxmp can list within a dc:language element. As explained in Section 3.1, we defer the invocation of `\AtEndPreamble` to the end of the file.

```
\edef\hyxmp@concated@metadata{}
\expandafter\hyxmp@aep@toks\expandafter=\expandafter{%
  \the\hyxmp@aep@toks
  \AtEndPreamble{%
    \@ifpackageloaded{babel}{%
      \edef\hyxmp@concated@metadata{babel}%
    }{%
      \@ifpackageloaded{polyglossia}{%
        \edef\hyxmp@concated@metadata{polyglossia}%
      }{%
      }%
    }%
  }%
}
```

`\hyxmp@warn@if@no@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@warn@if@no@metadata}{%
  \edef\hyxmp@concated@metadata{%
    \hyxmp@concated@metadata
    \@baseurl
    \@pdfauthor
    \@pdfauthortitle
    \@pdfbookedition
    \@pdfbytes
    \@pdfcaptionwriter
    \@pdfcontactaddress
    \@pdfcontactcity
    \@pdfcontactcountry
    \@pdfcontactemail
    \@pdfcontactphone
    \@pdfcontactpostcode
    \@pdfcontactregion
    \@pdfcontacturl
    \@pdfcopyright
    \@pdfcreationdate
    \@pdfdatetime
    \@pdfdoi
    \@pdfeissn
    \@pdfidentifier
    \@pdfisbn
    \@pdfissn
    \@pdfissuenum
    \@pdfkeywords
    \@pdflang
    \@pdflicenseurl
```

```
    \@pdfmetadatetime
    \@pdfmoddate
    \@pdfnumpages
    \@pdfpagerange
    \@pdfpublication
    \@pdfpubtype
    \@pdfsubject
    \@pdfsubtitle
    \@pdftitle
    \@pdfuapart
    \@pdfurl
    \@pdfvolumenum
    \@pdfxstandard
  }%
  \ifx\hyxmp@concated@metadata\@empty
    \PackageWarningNoLine{hyperxmp}{%
      \hyxmp@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
}
```

`\hyxmp@check@standards` Most PDF standards require that certain metadata be present. If compliance with a PDF standard is claimed but any of the metadata it requires are absent, issue a warning message.

```
\newcommand*{\hyxmp@check@standards}{%
```

If the pdfa option was passed to hyperref but `\@pdfapart` is not set, set it to `1` and `\@pdfaconformance` to `B`.

```
  \ifHy@pdfa
    \@ifmtargexp{\@pdfapart}{%
      \PackageWarningNoLine{hyperxmp}{%
        'pdfa' was passed to hyperref, but 'pdfapart' was\MessageBreak
        not specified.\space\space  Setting pdfapart to '1' and\MessageBreak
        pdfaconformance to 'B'%
      }%
      \gdef\@pdfapart{1}%
      \gdef\@pdfaconformance{B}%
    }%
    {}%
  \fi
```

`\hyxmp@standards` We define `\hyxmp@standards` to be non-empty if *any* PDF standard is claimed (currently, PDF/A, PDF/X, or PDF/UA.

```
  \edef\hyxmp@standards{%
    \@pdfapart
    \@pdfxstandard
    \@pdfuapart
  }%
```

Check that a document title was provided and is non-empty.

```
  \@ifnotmtargexp{\hyxmp@standards}{%
    \@ifmtargexp{\@pdftitle}{%
      \PackageWarningNoLine{hyperxmp}{%
        Missing pdftitle (required for PDF standards\MessageBreak
        compliance)%
      }%
    }%
    {}%
  }%
}
```

`\hyxmp@aep@toks` Right before we reach the `\begin{document}` we check if hyperref was loaded. In normal usage, the document will already have done a `\usepackage{hyperref}` because otherwise, `\hypersetup` will not have been defined, and only a limited amount of metadata will be included. However, in case the author is relying exclusively on hyperxmp's automatically detected metadata, we'll need to load hyperref now. As explained in Section 3.1, we defer the invocation of `\AtEndPreamble` to the end of the file.

```
\expandafter\hyxmp@aep@toks\expandafter=\expandafter{%
  \the\hyxmp@aep@toks
  \AtEndPreamble{%
    \RequirePackage{hyperref}%
```

Older versions of hyperref write the Info dictionary to the PDF file at the end of the document. Newer versions of hyperref write the Info dictionary to the PDF file at the *beginning* of the document. For compatibility with both old and new hyperref implementations we suppress writing the Info dictionary here, at the beginning of the document.

```
    \hyxmp@no@info@lists
```

If pdftitle is undefined but the author invoked `\title`, we copy the latter to the former. This addresses two problems: (1) handling LaTeX classes in which `\maketitle` clears `\title` and (2) ensuring that hyperref writes the same title to the PDF Info dictionary that hyperxmp writes to the XMP packet. We do likewise for `\author` → pdfauthor.

One tricky bit is that the standard LaTeX classes do not define `\@title` and `\@author` as empty strings but rather as calls to `\@latex@warning@no@line` that complain about a missing title/author. Hence, we can't simply test if the title and author are empty because they're not. Instead, we first locally redefine `\@latex@warning@no@line` to discard its argument then test if any text remains.

```
    \begingroup
      \let\@latex@warning@no@line=\@gobble
      \hyxmp@use@first@valid{pdftitle}{\@pdftitle}{%
        \scr@subject@var,%
        \@title
      }%
      \hyxmp@use@first@valid{pdfauthor}{\@pdfauthor}{%
```

```
        \scr@fromname@var,%
        \@author
      }%
    \endgroup
  }%
}
```

When we reach the `\end{document}` we need to gather up the metadata specified explicitly by the user, infer additional metadata where possible, and write the XMP packet to the PDF file.

```
\hyxmp@at@end{%
```

Fill in any missing metadata we can using values provided by the author via mechanisms other than the `\hypersetup` command.

```
  \hyxmp@auto@assign@data
```

If the document claims to comply with one or more PDF standards, check that all of the requisite metadata are present.

```
  \hyxmp@check@standards
```

We can finally construct the XMP packet and write it to the PDF document catalog.

```
  \hyxmp@warn@if@no@metadata
  \hyxmp@embed@packet
}
```

3.3 Advanced metadata detection

hyperxmp strives to be as convenient and user-friendly as possible. To that end, we try to automatically detect as much metadata as possible. The author can of course augment or override autodetected metadata by explicitly providing values to `\hypersetup`, but the hope is that we can save the author some effort in many cases.

In this section, we identify additional metadata we can use. Most of the functionality is class- or package-specific. For example, we check for phone numbers provided to the Koma letter classes via `\setkomavar{fromphone}{...}` and/or `\setkomavar{frommobilephone}{...}`, street addresses provided to the ACM article class via `\affiliation`, and languages the polyglossia package is instructed to load via `\setdefaultlanguage` and `\setotherlanguage`.

`\hyxmp@set@koma@phones` `\hyxmp@koma@phones`

Define `\hyxmp@koma@phones` as a comma-separated list of the phone numbers provided to a Koma letter class (mobile and landline).

```
\newcommand*{\hyxmp@set@koma@phones}{%
  \begingroup
    \Hy@unicodefalse
    \@if@def@and@nonempty{scr@frommobilephone@var}{%
      \@if@def@and@nonempty{scr@fromphone@var}{%
        \hyxmp@pdfstringdef\hyxmp@koma@phones{\scr@frommobilephone@var,\scr@fromphone@var}%
      }{%
        \hyxmp@pdfstringdef\hyxmp@koma@phones{\scr@frommobilephone@var}%
```

```
      }%
    }{%
      \@if@def@and@nonempty{scr@fromphone@var}{%
        \hyxmp@pdfstringdef\hyxmp@koma@phones{\scr@fromphone@var}%
      }{%
      }%
    }%
  \endgroup
}
```

`\hyxmp@use@first@valid` Given a hyperxmp option (`#1`), its current value (`#2`), and a comma-separated list of option names (`#3`), if the current value is empy, invoke `\hypersetup` to set the option to the first non-empty item in the list. If all items in the list are empty, do nothing.

```
\newcommand*{\hyxmp@use@first@valid}[3]{%
  \@ifmtargexp{#2}{%
    \hyxmp@use@first@valid@i{#1}#3,!,%
  }%
  {}%
}
```

`\hyxmp@use@first@valid@i` This macro performs all the work for `\hyxmp@use@first@valid`. It loops over a comma-separated list of macros (`#2`), stopping when it encounters an end-of-list marker ("`!`"). The first list element that is neither undefined nor empty is assigned to a given option name (`#1`) using `\hypersetup`.

```
\def\hyxmp@use@first@valid@i#1#2,{%
  \def\next{\hyxmp@use@first@valid@i{#1}}%
  \ifx#2!%
    \let\next=\relax
  \else
    \ifx#2\undefined
    \else
      \@ifnotmtargexp{#2}{%
        \hypersetup{#1={#2}}%
        \def\next##1!,{}%
      }%
    \fi
  \fi
  \next
}
```

`\hyxmp@auto@assign@data` If certain metadata are unspecified, try to specify meaningful values using data provided by author via other means (e.g., `\title` for the document's title).

```
\newcommand*{\hyxmp@auto@assign@data}{%
```

If `\@pdflang` is not set, see if we can detect the document language via either the babel or polyglossia packages.

```
  \@if@def@and@nonempty{@pdflang}{%
    \let\hyxmp@dc@lang=\@pdflang
```

```
}{%
  \hyxmp@detect@langs
}%
```

Replace an empty `\@pdfmetalang`. If `\@pdflang` is defined, use that as the metadata language. Otherwise, use x-default.

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

Identify various author-provided information that can be co-opted for use as XMP metadata.

```
\hyxmp@use@first@valid{pdfcontactemail}{\@pdfcontactemail}{%
  \scr@fromemail@var
}%
\hyxmp@set@koma@phones
\hyxmp@use@first@valid{pdfcontactphone}{\@pdfcontactphone}{%
  \hyxmp@koma@phones
}%
\hyxmp@use@first@valid{pdfcontacturl}{\@pdfcontacturl}{%
  \scr@fromurl@var
}%
\hyxmp@use@first@valid{pdfsubtitle}{\@pdfsubtitle}{%
  \@subtitle
}%
\hyxmp@use@first@valid{pdfpublisher}{\@pdfpublisher}{%
  \@publishers
}%
```

We handle the acmart class specially. acmart stores author-provided contact information in a structured format that we can process fairly easily. Note that if the author is not using the acmart class, `\hyxmp@parse@acmart` will have been redefined to do nothing.

```
\hyxmp@parse@acmart
```

Most PDF standards dictate that if the same metadata appear in both the XMP packet and the PDF Info dictionary, the metadata must match. This requirement poses a problem for a user-unspecified pdfcreationdate in the context of XeLaTeX. In this case we explicitly define `\@pdfcreationdate` as `\hyxmp@today@pdf` to prevent the `xdvipdfmx` back-end processor from detecting a missing CreationDate in the Info dictionary and adding its own—typically a few seconds after hyperxmp has constructed an xmp:CreateDate for the XMP metadata and leading to a metadata mismatch.

```
\@ifundefined{XeTeXversion}{}{%
  \@ifmtargexp{\@pdfcreationdate}{%
    \let\@pdfcreationdate=\hyxmp@today@pdf
```

```
    }%
    {}%
  }%
```

Query the document currently being built for page and byte counts.

```
  \hyxmp@query@self
}
```

hyperxmp can directly query the page count using LuaTeX features. When any other TeX engine is used, hyperxmp employs the totpages package to help tally the total number of pages.

```
\ifLuaTeX
\else
  \RequirePackage{totpages}
\fi
```

Determine the size of the output file from the *previous* run of LuaLaTeX. This action has to be performed before the `\begin{document}` because at that point the size of the output file is reset to zero. We use `\jobname.pdf` as the name of the output file because `status.output_file_name` is not defined at this point.

It's possible to use pdfLaTeX's `\pdffilesize` primitive to query the size of `\jobname.pdf` under pdfLaTeX. Unfortunately, doing so has a side effect of making `latexmk` view the PDF file as an input file, which puts `latexmk` in an infinite build loop. (This was the case for hyperxmp v5.5 and v5.6.) See the discussion at `https://github.com/borisveytsman/acmart/issues/413` for more information.

```
\ifLuaTeX
```

Now that we know we're running LuaLaTeX we define a Lua function, `get_pdf_size`, that takes the base name of the output file and returns the number of bytes in the corresponding PDF file. One difficulty is that, at the time of this writing, LuaTeX lacks a mechanism for querying the full name of the output file. Our workaround is a tad kludgy but seems to work. We walk the list of command-line arguments for "`--output-directory=`⟨*dir*⟩". (We in fact accept either one or two initial dashes and abbreviations as terse as "`-output-d`".) Then, we concatenate the output directory (or "`.`" if unspecified), a path separator, the given base name of the job, and a "`.pdf`" extension. Alas, different operating systems use different path separators so we have to query the operating-system type to select an appropriate separator: "`\`" on Windows/DOS and "`/`" on everything else.

`get_pdf_size` is called regardless of whether we're producing PDF or DVI output. We assume that even if the user specified `--output-format=dvi`, the user's intention is eventually to convert the document to PDF.

```
  \begin{luacode*}
function get_pdf_size (bname)
```

Search the list of command-line arguments for the output directory.

```
  local odir = ""
  for _, opt in ipairs(arg) do
    local m = string.match(opt, "%-output%-d.-=(.*)")
```

```
      if m then
         odir = m
      end
   end
```

Set the path separator to either "/" or "\", depending on the operating system.

```
   local sep = "/"
   if os.type == "windows" or os.type == "msdos" then
      sep = "\\\\"
   end
```

Concatenate the output directory, path separator, base name, and `.pdf` extension. Do not insert a path separator if either (1) no output directory was specified, (2) the output directory already ends with the path separator, or (3) the output directory ends in a colon (and is therefore a relative directory) on Windows/DOS. As a few examples,

- "" + "/" + "myfile" + ".pdf" = "myfile.pdf"
- "/docs" + "/" + "myfile" + ".pdf" = "/docs/myfile.pdf"
- "/docs/" + "/" + "myfile" + ".pdf" = "/docs/myfile.pdf"
- "C:\docs" + "\" + "myfile" + ".pdf" = "C:\docs\myfile.pdf"
- "C:\docs\" + "\" + "myfile" + ".pdf" = "C:\docs\myfile.pdf"
- "C:\" + "\" + "myfile" + ".pdf" = "C:\myfile.pdf"
- "C:" + "\" + "myfile" + ".pdf" = "C:myfile.pdf"

```
   local dlast = string.sub(odir, -1)
   if odir == "" or dlast == sep or (dlast == ":" and sep == "\\\\") then
      sep = ""
   end
   local fname = odir .. sep .. bname .. ".pdf"
```

Query the file size and return it.

```
   local nbytes = lfs.attributes(fname, "size")
   return nbytes
end
  \end{luacode*}
```

Now that we've defined `get_pdf_size` we invoke it, passing it `\hyxmp@jobname` as the base name of the job. (Recall that `\hyxmp@jobname` is the same as `\jobname` but with any surrounding double quotes removed.) We store `get_pdf_size`'s output—which will be empty if the PDF file doesn't yet exist—in `\hyxmp@prev@pdf@size`.

```
  \xdef\hyxmp@prev@pdf@size{%
    \luadirect{
nbytes = get_pdf_size("\hyxmp@jobname")
if nbytes then
    tex.write(nbytes)
end
    }%
  }%
\fi
```

\hyxmp@query@self Query the document currently being built to acquire page and byte counts.

```
\newcommand*{\hyxmp@query@self}{%
```

LuaTeX exposes via `status.total_pages` the number of pages written. We use this mechanism when available to assign `\@pdfnumpages`. To finalize the page count we first issue a `\clearpage`.

```
  \ifLuaTeX
    \@if@def@and@nonempty{@pdfnumpages}{%
    }{%
      \clearpage
      \xdef\@pdfnumpages{\luadirect{tex.print(status.total_pages)}}%
    }%
  \else
```

Without LuaTeX we rely on the totpages package to help count the number of pages. This may require an additional run of LaTeX, but the user will be notified in that case.

```
    \pdfstringdef\@pdfnumpages{\ref*{TotPages}}%
  \fi
```

If pdfbytes hasn't been set, set it to the output file's size from the previous run.

```
  \hyxmp@use@first@valid{pdfbytes}{\@pdfbytes}{%
    \hyxmp@prev@pdf@size
  }%
}
```

\hyxmp@parse@acmart The acmart class stores a rich set of author metadata in its `\addresses` macro. `\hyxmp@parse@acmart` extracts the contact information for the first author and converts that to XMP metadata.

```
\newcommand*{\hyxmp@parse@acmart}{%
  \begingroup
```

\@author acmart has already invoked `\hypersetup{pdfauthor=...}` to specify the complete list of authors. At this point, `\@author` is defined to produce a warning message. We locally redefine it to do nothing.

```
    \let\@author=\@gobble
```

\email \hyxmp@address@val Within `\addresses`, `\email` is defined to accept two arguments, the second of which is the author's email address.

```
    \def\email##1##2{%
      \def\hyxmp@address@val{##2}%
      \hyxmp@use@first@valid{pdfcontactemail}{\@pdfcontactemail}{%
        \hyxmp@address@val
      }%
    }%
```

\streetaddress \hyxmp@address@val `\streetaddress` wraps the author's street address.

```
    \def\streetaddress##1{%
      \def\hyxmp@address@val{##1}%
```

```
      \hyxmp@use@first@valid{pdfcontactaddress}{\@pdfcontactaddress}{%
        \hyxmp@address@val
      }%
    }%
```

\city \hyxmp@address@val

`\city` wraps the author's city name.

```
    \def\city##1{%
      \def\hyxmp@address@val{##1}%
      \hyxmp@use@first@valid{pdfcontactcity}{\@pdfcontactcity}{%
        \hyxmp@address@val
      }%
    }%
```

\state \hyxmp@address@val

`\state` wraps the author's state or region name.

```
    \def\state##1{%
      \def\hyxmp@address@val{##1}%
      \hyxmp@use@first@valid{pdfcontactregion}{\@pdfcontactregion}{%
        \hyxmp@address@val
      }%
    }%
```

\country \hyxmp@address@val

`\country` wraps the author's country name.

```
    \def\country##1{%
      \def\hyxmp@address@val{##1}%
      \hyxmp@use@first@valid{pdfcontactcountry}{\@pdfcontactcountry}{%
        \hyxmp@address@val
      }%
    }%
```

\postcode \hyxmp@address@val

`\postcode` wraps the author's postal code.

```
    \def\postcode##1{%
      \def\hyxmp@address@val{##1}%
      \hyxmp@use@first@valid{pdfcontactpostcode}{\@pdfcontactpostcode}{%
        \hyxmp@address@val
      }%
    }%
```

\affiliation

We want to produce XMP metadata for only a single affiliation. Although `\hyxmp@use@first@valid` will ensure that only the first email, city, country, etc. encountered is considered, we run the first of one affiliation defining, say, a city and state but no country and a subsequent affiliation defining a country. In that case, the XMP would include the first author's city and state and the subsequent author's country. Hence, we define `\affiliation` to "self destruct" after its first use, discarding all further affiliations.

```
    \def\affiliation##1##2{%
      ##2%
      \let\affiliation=\@gobbletwo
    }%
```

We want to evaluate `\addresses` with the preceding local definitions in effect, but we don't want to typeset any text appearing in the string. Hence, we "typeset" `\addresses` within a box that is subsequently discarded.

```
    \setbox0=\hbox{\addresses}%
  \endgroup
```

acmart supports other relevant metadata in addition to the authors' mailing addresses. For instance, papers accepted for publication indicate their DOI number. However, papers under review will contain either a placeholder DOI, "10.1145/nnnnnnn.nnnnnnn", or the example DOI specified in the acmart example document, "10.1145/1122445.1122456". We ignore both of those DOIs.

```
  \@if@def@and@nonempty{@acmDOI}{%
    \IfSubStr{\@acmDOI}{10.1145/1122445.1122456}{}{%
      \IfSubStr{\@acmDOI}{10.1145/nnnnnnn.nnnnnnn}{}{%
        \hyxmp@use@first@valid{pdfdoi}{\@pdfdoi}{%
          \@acmDOI
        }%
      }%
    }%
  }%
  {}%
```

`\hyxmp@strip@isbn@date` `\hyxmp@acm@isbn` Papers appearing in conference proceedings specify the proceedings' ISBN. As with `\@acmDOI` above, we ignore both the placeholder ISBN, "978-x-xxxx-xxxx-x/YY/MM", and the example ISBN, "978-1-4503-XXXX-X/18/06". We also strip off the "`/`⟨*year*⟩`/`⟨*month*⟩" suffix so as to include a true ISBN in the XMP metadata.

```
  \@if@def@and@nonempty{@acmISBN}{%
    \IfSubStr{\@acmISBN}{XXXX}{}{%
      \IfSubStr{\@acmISBN}{xxxx}{}{%
        \def\hyxmp@strip@isbn@date##1/##2!{##1}%
        \edef\hyxmp@acm@isbn{%
          \expandafter\hyxmp@strip@isbn@date\@acmISBN/!%
        }%
        \hyxmp@use@first@valid{pdfisbn}{\@pdfisbn}{%
          \hyxmp@acm@isbn
        }%
      }%
    }%
  }%
  {}%
```

`\hyxmp@acm@publisher` The publisher is of course ACM.

```
  \def\hyxmp@acm@publisher{Association for Computing Machinery}%
  \hyxmp@use@first@valid{pdfpublisher}{\@pdfpublisher}{%
    \hyxmp@acm@publisher
  }%
```

Use the journal name if defined, otherwise the book name (for conference proceedings).

```
  \hyxmp@use@first@valid{pdfpublication}{\@pdfpublication}{%
    \@journalName,%
    \@acmBooktitle,%
    \@acmConference
  }%
```

`\hyxmp@acm@pubtype` acmart makes clear whether it's typesetting a journal article. If it's not a journal, we assume it's a book (conference proceedings).

```
  \if@ACM@journal
    \def\hyxmp@acm@pubtype{journal}%
  \else
    \def\hyxmp@acm@pubtype{book}%
  \fi
  \hyxmp@use@first@valid{pdfpubtype}{\@pdfpubtype}{%
    \hyxmp@acm@pubtype
  }%
```

Journal articles have a volume and issue number.

```
  \hyxmp@use@first@valid{pdfvolumenum}{\@pdfvolumenum}{%
    \@acmVolume
  }%
  \hyxmp@use@first@valid{pdfissuenum}{\@pdfissuenum}{%
    \@acmNumber
  }%
}
```

Nullify `\hyxmp@parse@acmart` if the author is not using the acmart class.

```
\@ifclassloaded{acmart}{}{\let\hyxmp@parse@acmart=\relax}
```

`\hyxmp@dc@lang` `\hyxmp@dc@lang` is a comma-separated list of all languages used in the document.

```
\let\hyxmp@dc@lang=\@empty
```

`\hyxmp@detect@langs` If pdflang was not specified, try to determine the document language(s) using either babel's or polyglossia's definitions.

```
\newcommand*{\hyxmp@detect@langs}{%
  \@ifundefined{mainbcp47id}{%
    \@ifundefined{LocaleForEach}{%
```

The document doesn't appear to have loaded either babel or polyglossia. In this case we have one small task to do. 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 explicitly set `\@pdflang` to `\@empty` to avoid problems with non-expandable symbols.

```
      \let\@pdflang=\@empty
    }{%
```

\hyxmp@dc@lang
\hyxmp@lang@tag
\hyxmp@lang@name
\@pdflang

Use babel's \LocaleForEach and \getlocaleproperty to set \@pdflang to the document's main language and \hyxmp@dc@lang to a comma-separated list of all languages used.

```
      \BabelEnsureInfo
      \LocaleForEach{%
        \getlocaleproperty\hyxmp@lang@tag{##1}{identification/tag.bcp47}%
        \ifx\hyxmp@dc@lang\@empty
          \xdef\hyxmp@dc@lang{\hyxmp@lang@tag}%
        \else
          \xdef\hyxmp@dc@lang{\hyxmp@dc@lang,\hyxmp@lang@tag}%
        \fi
        \def\hyxmp@lang@name{##1}%
        \ifx\hyxmp@lang@name\bbl@main@language
          \edef\@pdflang{\hyxmp@lang@tag}%
        \fi
      }%
    }%
  }{%
```

Use polyglossia's \mainbcp47id as the document's main language and its \xpg@bcp@loaded as a comma-separated list of all document languages.

```
    \xdef\@pdflang{\csname mainbcp47id\endcsname}%
    \edef\hyxmp@dc@lang{\xpg@bcp@loaded}%
  }%
}
```

3.4 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 3.4.1); parse dates in both PDF and XMP formats (Section 3.4.2; trim spaces off the ends of strings (Section 3.4.3); convert text to XML (e.g., from <scott+hyxmp@pakin.org> to <scott+hyxmp@pakin.org>) (Section 3.4.4); simplify the pretty-printing of a begin tag, XML text, and end tag (Section 3.4.5; and provide metadata in multiple languages (Section 3.4.6).

3.4.1 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 the inability of older Adobe Acrobat versions to display author lists correctly 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.

```
\newcommand{\XMPTruncateList}[1]{{%
    \PackageWarning{hyperxmp}{%
      \noexpand\XMPTruncateList has been deprecated since\MessageBreak
      hyperxmp 4.0 and may be removed in future\MessageBreak
      versions of the package.  \noexpand\XMPTruncateList\MessageBreak
      was found}%
  \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
}}
```

3.4.2 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:20201122003649-07'00') [4], while XMP timestamps are of the form "YYYY-MM-DDThh:mm:ss+TT:tt" (e.g., 2020-11-22T00:36:49-07:00) [5]. 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\ifnum\expandafter`\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@xmp@define` Use TeX primitives to define a given macro as today's date in YYYY-MM-DDThh:mmZ format.

```
\def\hyxmp@today@xmp@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
  \xdef#1{#1Z}%
}
```

`\hyxmp@try@today` If `\hyxmp@today@xmp` is still empty and `#1` is defined, evaluate `#2`. Otherwise, do nothing.

```
\def\hyxmp@try@today#1#2{%
  \@ifmtargexp{\hyxmp@today@xmp}{%
    \@ifundefined{#1}{}{#2}%
  }%
  {}%
}
```

`\hyxmp@today@xmp` Define `\hyxmp@today@xmp` as the current date and (if available) time and timezone in XMP Date format [5].

```
\def\hyxmp@today@xmp{}
```

Case 1: `\pdfcreationdate` is defined (pdfLATEX and pre-0.85 LuaLATEX).

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

Case 2: `\pdffeedback` is defined (LuaLaTeX 0.85+).

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

`\hyxmp@timestamp`

Case 3: `\filemoddate` is defined (XeLaTeX). In this case, we treat the timestamp of the job's `.log` file as the current date/time.

```
\hyxmp@try@today{filemoddate}{%
  \edef\hyxmp@today@xmp{\filemoddate{\hyxmp@jobname.log}}%
  \edef\next{%
    \edef\noexpand\hyxmp@today@xmp{\noexpand\hyxmp@as@xmp@date{\hyxmp@today@xmp}}%
  }%
  \next
}%
```

Case 4: None of the above. Do the best we can using the available TeX primitives (`\year`, `\month`, `\day`, and `\time`.

```
\hyxmp@try@today{year}{%
  \hyxmp@today@xmp@define\hyxmp@today@xmp
}
```

`\hyxmp@today@pdf`

Define `\hyxmp@today@pdf` as the current date and (if available) time and timezone in PDF date format [4]. To do so we simply convert `\hyxmp@today@xmp`, defined above, from XMP to PDF using `\hyxmp@xmp@to@pdf@date`.

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

3.4.3 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 [7]. 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
```

3.4.4 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
```

`\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" [3]. 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 (XETEX 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
```

3.4.5 Outputting structured XML

An XMP packet consists of structured XML data. We define some helper routines to handle the repetitive tasks of indenting a consistent number of spaces, inserting begin and end tags, and escaping arbitrary text as necessary for XML compatibility.

`\hyxmp@extra@indent` This macro is used internally to increase the amount of indentation when writing certain XML data. It is normally defined as empty but can temporarily be redefined to a sequence of `\space` characters.

```
\newcommand*{\hyxmp@extra@indent}{}
```

`\hyxmp@add@simple` 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]{%
  \@ifnotmtargexp{#2}{%
    \hyxmp@xmlify{#2}%
    \hyxmp@add@to@xml{\hyxmp@extra@indent______<}%
    \xdef\hyxmp@xml{\hyxmp@xml#1}%
    \hyxmp@add@to@xml{>\hyxmp@xmlified</}%
    \xdef\hyxmp@xml{\hyxmp@xml#1>^^J}%
  }%
}
```

`\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{%
      \hyxmp@extra@indent______<#1>\hyxmp@xmlified</#1>^^J%
    }%
  \fi
}
```

`\hyxmp@add@simple@lang` 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. However, if the string begins with a language code in square brackets, specify that as the (sole) language for the tag.

```
\newcommand*{\hyxmp@add@simple@lang}[2]{%
  \@ifnotmtarg{#2}{%
    \hyxmp@xmlify{#2}%
```

```
    \expandafter\hyxmp@add@simple@lang@i\hyxmp@xmlified\relax{#1}%
  }%
}
```

`\hyxmp@add@simple@lang@i` This is a helper macro for `\hyxmp@add@simple@lang`. It takes an optional language code (in brackets), text up to `\relax`, and a tag, and typesets the text within the XML tag.

```
\newcommand*{\hyxmp@add@simple@lang@i}{%
  \@ifnextchar[\hyxmp@add@simple@lang@ii{\hyxmp@add@simple@lang@ii[\@pdfmetalang]}%
}
```

`\hyxmp@add@simple@lang@ii` This is another helper macro for `\hyxmp@add@simple@lang`. It takes an mandatory language code (in brackets; can be empty), text up to `\relax`, and a tag, and typesets the text within the XML tag.

```
\def\hyxmp@add@simple@lang@ii[#1]#2\relax#3{%
  \@ifnotmtarg{#2}{%
    \hyxmp@xmlify{#2}%
    \@ifmtarg{#1}{%
      \hyxmp@add@to@xml{%
______<#3>\hyxmp@xmlified</#3>^^J%
      }%
    }{%
      \hyxmp@add@to@xml{%
______<#3 xml:lang="#1">\hyxmp@xmlified</#3>^^J%
      }%
    }%
  }%
}
```

`\hyxmp@add@simple@pfx` Given an XMP tag (`#1`), a—typically hard-wired—prefix string (`#2`), and a main string (`#2`), if the main string is nonempty, add a begin tag, both strings, 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@pfx}[3]{%
  \@ifnotmtargexp{#3}{%
    \hyxmp@add@to@xml{\hyxmp@extra@indent______<}%
    \xdef\hyxmp@xml{\hyxmp@xml#1}%
    \hyxmp@pdfstringdef\hyxmp@iprefix{#2}%
    \hyxmp@xmlify{\hyxmp@iprefix}%
    \hyxmp@add@to@xml{>\hyxmp@xmlified}%
    \hyxmp@xmlify{#3}%
    \hyxmp@add@to@xml{\hyxmp@xmlified</}%
    \xdef\hyxmp@xml{\hyxmp@xml#1>^^J}%
  }%
}
```

3.4.6 Providing metadata in multiple languages

Certain XMP tags—dc:title, dc:description, and dc:rights (and others? Let me know.)—can be expressed in multiple languages. The same text is used for both language pdfmetalang (default: pdflang) and language "x-default". To express the same metadata in multiple languages, we provide an `\XMPLangAlt` macro to construct a list of alternative forms for a piece of metadata.

`\hyxmp@alt@title` `\hyxmp@alt@description` `\hyxmp@alt@rights`

Each of these macros is a list in which each element is of the form "`\do` ⟨*language*⟩ ⟨*text*⟩" in which ⟨*language*⟩ is an ISO 639-1 two-letter country code with an optional ISO 3166-1 two-letter region code. For example, `\hyxmp@alt@title` may contain an element, "`\do {es-MX} {Este es mi documento}`".

```
\def\hyxmp@alt@title{}
\def\hyxmp@alt@description{}
\def\hyxmp@alt@rights{}
```

`\hyxmp@LA@accept`

This macro wraps `\define@key` to make the option "`#1`=⟨*value*⟩" append ⟨*value*⟩ to list `#2`.

```
\newcommand{\hyxmp@LA@accept}[2]{%
  \define@key{hyxmp@LA}{#1}{%
```

`\hyxmp@value`

As Niklas Beisert observed, if the option passed to the current key contains LaTeX code, this code will be included in the XMP packet, which is undesirable. Hence, we first clean up the string using `\hyxmp@pdfstringdef`.

```
    \hyxmp@pdfstringdef\hyxmp@value{##1}%
    \xdef#2{#2\noexpand\do {\hyxmp@cur@lang} {\hyxmp@value}}%
  }
}
```

Define ⟨*key*⟩=⟨*value*⟩ options for appending to each of the `\hyxmp@alt`⟨*tag*⟩ lists.

```
\hyxmp@LA@accept{pdftitle}{\hyxmp@alt@title}
\hyxmp@LA@accept{pdfsubject}{\hyxmp@alt@description}
\hyxmp@LA@accept{pdfcopyright}{\hyxmp@alt@rights}
```

`\XMPLangAlt`

Argument `#1` is a language expressed as a two-letter country code and optional two-letter region code. Argument `#2` is a list of ⟨*key*⟩=⟨*value*⟩ pairs. Keys correspond to `\hypersetup` options such as "`pdftitle`", "`pdfsubject`", and "`pdfcopyright`". Values are the alternative-language form of the text provided for the corresponding option.

```
\newcommand{\XMPLangAlt}[2]{%
  \let\do=\relax
```

`\hyxmp@cur@lang`

Store the provided language, which will be used during option processing.

```
  \edef\hyxmp@cur@lang{#1}%
  \setkeys{hyxmp@LA}{#2}%
}
```

3.5 UUID generation

We use a linear congruential generator to produce pseudorandom version 4 UUIDs [12]. 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` $+ c \pmod{\texttt{\hyxmp@big@prime}}$.

```
\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` $\pmod{\texttt{\hyxmp@big@prime}}$. 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 [12], define macro `#1` as a UUID of the form "`uuid:`*xxxxxxxx*`-`*xxxx*`-4`*xxx*`-`*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` `\hyxmp@seed@string`

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{\hyxmp@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` `\hyxmp@seed@string`

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. For the current timestamp, we use both the document-specified timestamp from pdfdate and the TEX time. The former can be more precise (to sub-seconds) but may be less random (as it depends on manual document modifications) while the latter is typically less precise (to minutes) but may be more random (as it is updated automatically).

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

```
  \hyxmp@create@uuid\hyxmp@InstanceID
}
```

3.6 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" [5]. ("PI" is an abbreviation for "processing instructions"). The serialized XMP includes blocks of XML for various XMP schemata: Adobe PDF (Section 3.6.2), Dublin Core (Section 3.6.3), XMP Rights Management (Section 3.6.4), XMP Media Management (Section 3.6.5), XMP Basic (Section 3.6.6), Photoshop (Section 3.6.7), PDF/* Identification (Section 3.6.8), IPTC Photo Metadata (Section 3.6.9), PRISM Basic Metadata (Section 3.6.10), Journal Article Versions (Section 3.6.11), and XMP Paged-Text (Section 3.6.12). The `\hyxmp@construct@packet` macro (Section 3.6.14) 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.

3.6.1 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 [5]. `\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}
```

3.6.2 The Adobe PDF schema

Older versions of hyperref defined a default producer; newer versions do not. Instead, they let the TEX engine define the producer itself. This poses a problem for PDF/A compliance because hyperxmp sees an empty producer and therefore omits writing a pdf:Producer to the XMP packet, causing a mismatch between the data in the XMP packet and the data in the PDF Info dictionary. To ensure consistency between XMP and Info, we explicitly define our own default `\@pdfproducer` here.

`\@pdfproducer` `\hyxmp@define@pdfproducer` Define `\@pdfproducer` using the banner string if available or the TEX engine's version number if not.

```
\newcommand*{\hyxmp@define@pdfproducer}{%
  \gdef\@pdfproducer{TeX}
  \ifLuaTeX
    \expandafter\hyxmp@banner@to@producer\expandafter{\luatexbanner}%
  \else
    \ifPDFTeX
      \expandafter\hyxmp@banner@to@producer\expandafter{\pdftexbanner}%
    \else
      \ifXeTeX
        \edef\@pdfproducer{XeTeX version \the\XeTeXversion\XeTeXrevision}%
      \fi
    \fi
  \fi
}
```

\@pdfproducer
\hyxmp@banner@to@producer

Define \@pdfproducer as the TEX engine's banner string (e.g., "This is LuaHBTeX, Version 1.12.0 (TeX Live 2020)"), removing the initial "This is" if possible (specifically, when ε-TEX's \scantokens primitive is available).

```
\def\hyxmp@banner@to@producer#1{%
  \ifx\scantokens\relax
    \gdef\@pdfproducer{#1}%
  \else
    {\scantokens{\makeatletter\hyxmp@remove@this#1\relax}}%
  \fi
}
```

\@pdfproducer
\hyxmp@remove@this

Define \@pdfproducer as a given banner string with the initial "This is" stripped off the beginning.

```
\def\hyxmp@remove@this This is #1\relax{\gdef\@pdfproducer{#1}}
```

If pdfproducer wasn't specified and hyperref didn't already define \@pdfproducer—old versions of hyperref did; newer ones don't—try to assign a meaningful producer string and use that.

```
\AtBeginDocument{%
  \ifx\@pdfproducer\relax
    \hyxmp@define@pdfproducer
  \fi
}
```

\hyxmp@assign@major@minor

Assign \hyxmp@major@minor to be the PDF version targeted by the running TEX engine.

\hyxmp@major@minor

```
\newcommand*{\hyxmp@assign@major@minor}{%
  \@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).

```
      \xdef\hyxmp@major@minor{\the\pdfminorversion}%
      \@ifundefined{pdfmajorversion}{%
```

Case 2(a): \pdfmajorversion is not defined (older versions of pdfLATEX and LuaLATEX).

```
        \xdef\hyxmp@major@minor{1.\hyxmp@major@minor}%
      }{%
```

Case 2(b): \pdfmajorversion is defined (pdfLATEX 1.40.21+).

```
        \xdef\hyxmp@major@minor{\the\pdfmajorversion.\hyxmp@major@minor}%
      }%
    }%
  }{%
```

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

```
    \xdef\hyxmp@major@minor{\the\pdfvariable majorversion.\the\pdfvariable minorversion}%
  }%
}
```

`\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@simple@var{pdf:Producer}{@pdfproducer}%
  \hyxmp@add@simple@var{pdf:Keywords}{@pdfkeywords}%
  \hyxmp@add@simple{pdf:Trapped}{\@pdftrapped}%
  \hyxmp@assign@major@minor
  \hyxmp@add@simple@var{pdf:PDFVersion}{hyxmp@major@minor}%
}
```

3.6.3 The Dublin Core schema

`\ifhyxmp@multi@langs` `\hyxmp@multi@langstrue` `\hyxmp@multi@langsfalse` These macros are used locally to `\hyxmp@rdf@dc`. If the property being processed has values in different languages, `\ifhyxmp@multi@langs` evaluates to TRUE. If it has a value in only a single language, `\ifhyxmp@multi@langs` evaluates to FALSE.

```
\newif\ifhyxmp@multi@langs
```

`\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.

```
  \@ifmtargexp{#3}{\@tempswafalse}{\@tempswatrue}%
  #1
    \@tempswatrue
  \fi
```

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

```
  \if@tempswa
    \hyxmp@xmlify{#3}%
```

\hyxmp@value Store the XML-ified version of #3 in \hyxmp@value so we can reuse \hyxmp@xmlifiied if necessary.

```
    \let\hyxmp@value=\hyxmp@xmlified
    \hyxmp@add@to@xml{%
______<dc:#2>^^J%
________<rdf:Alt>^^J%
    }%
```

Record whether property #2 has definitions in multiple languages.

```
    \@if@def@and@nonempty{hyxmp@alt@#2}{%
      \hyxmp@multi@langstrue
    }{%
      \hyxmp@multi@langsfalse
    }%
```

There are now four cases to consider: the cross product of {pdfmetalang = "x-default", pdfmetalang ≠ "x-default"} and {\XMPLangAlt was specified, \XMPLangAlt was not specified}. We handle each of these in turn.

```
    \ifx\@pdfmetalang\hyxmp@x@default
      \ifhyxmp@multi@langs
```

Case 1: No pdfmetalang but \XMPLangAlt. We consider this an error because the x-default language will not have a matching non-default language, in violation of the XMP specification's guidance [5, p. 23]:

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

```
        \PackageError{hyperxmp}%
          {\string\XMPLangAlt\space was used without specifying
           pdfmetalang\MessageBreak
           or pdflang}%
          {Be sure to assign a language code to the pdfmetalang key and/or a
           document\MessageBreak
           language to the pdflang key (e.g., \string\hypersetup{pdfmetalang={en}}).}%
      \else
```

Case 2: No pdfmetalang and no \XMPLangAlt. Here we specify only x-default as the language, as per the guidance quoted above.

```
        \hyxmp@add@to@xml{%
__________<rdf:li xml:lang="\hyxmp@x@default">\hyxmp@value</rdf:li>^^J%
        }%
      \fi
    \else
      \ifhyxmp@multi@langs
```

Case 3: Both pdfmetalang and \XMPLangAlt. In this case, we include an x-default followed by the pdfmetalang language, followed by all of the language alternatives.

```
        \hyxmp@xmlify{\@pdfmetalang}%
        \hyxmp@add@to@xml{%
__________<rdf:li xml:lang="\hyxmp@x@default">\hyxmp@value</rdf:li>^^J%
__________<rdf:li xml:lang="\hyxmp@xmlified">\hyxmp@value</rdf:li>^^J%
        }%
        \def\do##1##2{
          \hyxmp@xmlify{##2}%
          \hyxmp@add@to@xml{%
    __________<rdf:li xml:lang="##1">\hyxmp@xmlified</rdf:li>^^J%
          }%
        }%
        \csname hyxmp@alt@#2\endcsname
      \else
```

Case 4: pdfmetalang but no `\XMPLangAlt`. To reduce redundancy we omit the x-default and include the single language in which the text appears.

```
        \hyxmp@xmlify{\@pdfmetalang}%
        \hyxmp@add@to@xml{%
__________<rdf:li xml:lang="\hyxmp@xmlified">\hyxmp@value</rdf:li>^^J%
        }%
      \fi
    \fi
```

Complete this XMP element.

```
    \hyxmp@add@to@xml{%
________</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.

```
  \@ifmtargexp{#4}{\@tempswafalse}{\@tempswatrue}%
  #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@singleton@dc` Given an optional list type (`Seq` or `Bag`), a Dublin Core property, and a string, append a block of XML representing a one-element list consisting of the given string.

```
\newcommand{\hyxmp@singleton@dc}[3][Bag]{%
  \@ifnotmtargexp{#3}{%
    \hyxmp@xmlify{#3}%
    \hyxmp@add@to@xml{%
______<dc:#2>^^J%
________<rdf:#1>^^J%
__________<rdf:li>\hyxmp@xmlified</rdf:li>^^J%
________</rdf:#1>^^J%
______</dc:#2>^^J%
    }%
  }
}
```

`\hyxmp@cond@dc@identifier` Conditionally add a dc:identifier tag. Given a prefix string (`#1`) and a main string (`#2`), wrap these in a dc:identifier if the main string is nonempty and `\hyxmp@xmlified` *is* empty (implying the dc:identifier has not yet been written).

```
\newcommand*{\hyxmp@cond@dc@identifier}[2]{%
  \ifx\hyxmp@xmlified\@empty
    \@ifnotmtargexp{#2}{%
      \hyxmp@add@simple@pfx{dc:identifier}{#1}{#2}%
    }%
  \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, the dc:language property if the author specified pdflang, the dc:type property if the author specified pdftype, and the dc:identifier if the author specified pdfidentifier or if we can derive it from other options. We also specify the dc:source property using the base name of the source file with `.tex` appended and the dc:date property using the date the document was run through LaTeX—unless the author specified pdfdate, in which case we use that.

```
\newcommand*{\hyxmp@dc@schema}{%
  \hyxmp@add@simple{dc:format}{application/pdf}%
  \hyxmp@rdf@dc[\ifHy@pdfa]{title}{\@pdftitle}%
  \hyxmp@rdf@dc[\ifHy@pdfa]{description}{\@pdfsubject}%
  \hyxmp@rdf@dc{rights}{\@pdfcopyright}%
  \hyxmp@singleton@dc{publisher}{\@pdfpublisher}%
  \@ifmtargexp{\@pdfdatetime}{%
    \hyxmp@singleton@dc[Seq]{date}{\hyxmp@today@xmp}%
  }{%
    \hyxmp@singleton@dc[Seq]{date}{\@pdfdatetime}%
  }%
  \hyxmp@singleton@dc{type}{\@pdftype}%
  \hyxmp@list@to@xml[\ifHy@pdfa]{creator}{Seq}{\hyxmp@pdfauthor}%
  \hyxmp@list@to@xml{subject}{Bag}{\hyxmp@pdfkeywords}%
  \ifx\@pdfsource\@empty
  \else
    \hyxmp@add@simple{dc:source}{\@pdfsource}%
  \fi
  \hyxmp@list@to@xml{language}{Bag}{\hyxmp@dc@lang}%
% If |\@pdfidentifier| is empty, try setting it to each of |\@pdfdoi|,
% |\@pdfeissn|, |\@pdfissn|, and |\@pdfisbn|, in turn, with proper
% syntactic adjustments.
%    \begin{macrocode}
  \@ifmtargexp{\@pdfidentifier}{%
    \let\hyxmp@xmlified=\@empty
    \hyxmp@cond@dc@identifier{info:doi/}{\@pdfdoi}%
    \hyxmp@cond@dc@identifier{urn:ISSN:}{\@pdfeissn}%
    \hyxmp@cond@dc@identifier{urn:ISSN:}{\@pdfissn}%
    \hyxmp@cond@dc@identifier{urn:ISBN:}{\@pdfisbn}%
  }{%
    \hyxmp@add@simple{dc:identifier}{\@pdfidentifier}%
  }%
}
```

3.6.4 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.

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

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

```
  \let\hyxmp@rights=\@empty
  \ifx\@pdflicenseurl\@empty
  \else
    \def\hyxmp@rights{YES}%
  \fi
  \ifx\@pdfcopyright\@empty
  \else
    \def\hyxmp@rights{YES}%
  \fi
```

Include the license-statement URL and/or the copyright indication. The copyright statement itself is included by `\hyxmp@dc@schema` in Section 3.6.3.

```
  \ifx\hyxmp@rights\@empty
  \else
    \ifx\@pdfcopyright\@empty
    \else
      \hyxmp@add@simple{xmpRights:Marked}{True}%
    \fi
    \hyxmp@add@simple{xmpRights:WebStatement}{\@pdflicenseurl}%
  \fi
}
```

3.6.5 The XMP Media Management schema

`\hyxmp@aep@toks` Once we reach the end of the preamble and know that `\@pdftitle` and `\@pdfauthor` are no longer expected to change we use those macros (and others) to define one UUID for the document (`\hyxmp@DocumentID`) and one for the document instance (`\hyxmp@InstanceID`). As explained in Section 3.1, we defer the invocation of `\AtEndPreamble` to the end of the file.

```
\expandafter\hyxmp@aep@toks\expandafter=\expandafter{%
  \the\hyxmp@aep@toks
  \AtEndPreamble{%
    \@ifmtargexp{\hyxmp@DocumentID}{\hyxmp@def@DocumentID}{}%
    \@ifmtargexp{\hyxmp@InstanceID}{\hyxmp@def@InstanceID}{}%
  }%
}
```

`\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 [5]. As seen in Section 3.5, we do what we can to honor this intention from within a TEX-based workflow. We additionally support the xmpMM:VersionID property, whose value is supplied by the author using pdfversionid.

```
\gdef\hyxmp@mm@schema{%
```

```
  \hyxmp@add@simple{xmpMM:DocumentID}{\hyxmp@DocumentID}%
  \hyxmp@add@simple{xmpMM:InstanceID}{\hyxmp@InstanceID}%
  \hyxmp@add@simple{xmpMM:VersionID}{\@pdfversionid}%
  \hyxmp@add@simple{xmpMM:RenditionClass}{\@pdfrendition}%
}
```

3.6.6 The XMP Basic schema

`\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}{%
```

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

```
  \@ifmtargexp{\@pdfcreationdate}{%
    \hyxmp@add@simple{xmp:CreateDate}{\hyxmp@today@xmp}%
  }{%
    \hyxmp@add@simple{xmp:CreateDate}{%
      \expandafter\hyxmp@as@xmp@date\expandafter{\@pdfcreationdate}}%
  }%
```

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

```
  \@ifmtargexp{\@pdfmoddate}{%
    \hyxmp@add@simple{xmp:ModifyDate}{\hyxmp@today@xmp}%
  }{%
    \hyxmp@add@simple{xmp:ModifyDate}{%
      \expandafter\hyxmp@as@xmp@date\expandafter{\@pdfmoddate}}%
  }%
```

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

```
  \@ifmtargexp{\@pdfmetadatetime}{%
    \hyxmp@add@simple{xmp:MetadataDate}{\hyxmp@today@xmp}%
  }{%
    \hyxmp@add@simple{xmp:MetadataDate}{\@pdfmetadatetime}%
  }%
```

Define the creation tool and the base URL.

```
  \hyxmp@add@simple{xmp:CreatorTool}{\@pdfcreator}%
  \hyxmp@add@simple{xmp:BaseURL}{\@baseurl}%
}
```

3.6.7 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}%
  \hyxmp@add@simple{photoshop:AuthorsPosition}{\@pdfauthortitle}%
  \hyxmp@add@simple{photoshop:CaptionWriter}{\@pdfcaptionwriter}%
}
```

3.6.8 PDF/* Identification schemata

`\hyxmp@pdfa@id@schema` Add properties defined by the PDF/A Identification schema [13] 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@simple{pdfaid:part}{\@pdfapart}%
    \hyxmp@add@simple{pdfaid:conformance}{\@pdfaconformance}%
  \fi
}
```

`\hyxmp@pdfua@id@schema` If the document conforms to a PDF/UA standard, the author can indicate the standard version with pdfuapart.

```
\newcommand*{\hyxmp@pdfua@id@schema}{%
  \hyxmp@add@simple{pdfuaid:part}{\@pdfuapart}%
}
```

`\hyxmp@pdfx@id@schema` If the document conforms to a PDF/X standard, the author can indicate the standard version with pdfxstandard. We separately handle PDF/X-1, PDF/X-2 and PDF/X-3, and PDF/X-4 onwards.

```
\newcommand*{\hyxmp@pdfx@id@schema}{%
  \@tempcnta=0\hyxmp@pdfx@major\relax
  \ifnum\@tempcnta=0
  \else
    \ifnum\@tempcnta=1
      \hyxmp@add@simple{pdfx:GTS_PDFXVersion}{PDF/X-1:2001}%
      \hyxmp@add@simple{pdfx:GTS_PDFXConformance}{\@pdfxstandard}%
    \else
      \ifnum\@tempcnta<4
        \hyxmp@add@simple{pdfx:GTS_PDFXVersion}{\@pdfxstandard}%
      \else
        \hyxmp@add@simple{pdfxid:GTS_PDFXVersion}{\@pdfxstandard}%
      \fi
    \fi
  \fi
}
```

3.6.9 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 if the list is empty.

```
\newcommand*{\hyxmp@list@to@lines}[2]{%
  \@ifnotmtargexp{#2}{%
    \bgroup
      \hyxmp@add@to@xml{%
        \hyxmp@extra@indent_____<#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
  }%
}
```

`\hyxmp@iptc@schema` Add properties defined by the IPTC Photo Metadata schema [10] to the `\hyxmp@xml` macro. We currently support only the Iptc4xmpCore:CreatorContactInfo property, although this is a structure containing multiple fields.

```
\gdef\hyxmp@iptc@schema{%
```

Because we currently support only Iptc4xmpCore:CreatorContactInfo it suffices to check if we have any relevant data. If so, we instantiate a Iptc4xmpCore:ContactInfo structure with all available fields.

```
  \ifx\hyxmp@iptc@data\@empty
  \else
    \hyxmp@add@to@xml{%
______<Iptc4xmpCore:CreatorContactInfo rdf:parseType="Resource">^^J%
    }%
```

We locally redefine `\hyxmp@extra@indent` to increase the indentation of the assignments to Iptc4xmpCore:CreatorContactInfo's fields.

```
    \bgroup
      \edef\hyxmp@extra@indent{\hyxmp@extra@indent\space\space}%
      \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" [10]. 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.

```
      \def\xmplinesep{,}%
      \hyxmp@list@to@lines{Iptc4xmpCore:CiTelWork}{\@pdfcontactphone}%
      \hyxmp@list@to@lines{Iptc4xmpCore:CiEmailWork}{\@pdfcontactemail}%
      \hyxmp@list@to@lines{Iptc4xmpCore:CiUrlWork}{\@pdfcontacturl}%
    \egroup
    \hyxmp@add@to@xml{%
______</Iptc4xmpCore:CreatorContactInfo>^^J%
    }%
  \fi
}
```

3.6.10 The PRISM Basic Metadata schema

`\hyxmp@prism@schema` Add properties defined by the PRISM Basic Metadata schema [8].

```
\newcommand*{\hyxmp@prism@schema}{%
  \ifx\hyxmp@prism@data\@empty
  \else
    \hyxmp@add@simple{prism:complianceProfile}{three}%
  \fi
  \hyxmp@add@simple@lang{prism:subtitle}{\@pdfsubtitle}%
  \hyxmp@add@simple@lang{prism:publicationName}{\@pdfpublication}%
  \hyxmp@add@simple{prism:aggregationType}{\@pdfpubtype}%
```

```
  \hyxmp@add@simple@lang{prism:bookEdition}{\@pdfbookedition}%
  \hyxmp@add@simple{prism:volume}{\@pdfvolumenum}%
  \hyxmp@add@simple{prism:number}{\@pdfissuenum}%
  \hyxmp@add@simple{prism:pageRange}{\@pdfpagerange}%
  \hyxmp@add@simple{prism:isbn}{\@pdfisbn}%
  \hyxmp@add@simple{prism:issn}{\@pdfissn}%
  \hyxmp@add@simple{prism:eIssn}{\@pdfeissn}%
  \hyxmp@add@simple{prism:doi}{\@pdfdoi}%
  \hyxmp@add@simple{prism:url}{\@pdfurl}%
  \hyxmp@add@simple{prism:byteCount}{\@pdfbytes}%
  \hyxmp@add@simple{prism:pageCount}{\@pdfnumpages}%
}
```

3.6.11 The Journal Article Versions (JAV) schema

`\hyxmp@jav@schema` Add properties defined by the NISO/ALPSP Journal Article Versions schema [1].

```
\newcommand*{\hyxmp@jav@schema}{%
  \hyxmp@add@simple{jav:journal_article_version}{\@pdfpubstatus}%
}
```

3.6.12 The XMP Paged-Text schema

`\hyxmp@xmptpg@schema` The XMP Paged-Text schema [5] includes properties related to the construction of the PDF file itself. We acquire most of this information through LuaTeX mechanisms and therefore include much less information when run from other TeX engines.

```
\newcommand*{\hyxmp@xmptpg@schema}{%
  \ifLuaTeX
    \luadirect{write_xmp_font_list(\the\hyxmp@cct)}%
  \fi
  \hyxmp@add@simple{xmpTPg:NPages}{\@pdfnumpages}%
}
```

`\hyxmp@cct` We store the current category-code table to ensure that `write_xmp_font_list`'s output uses our redefined category codes.

```
\ifLuaTeX
  \newcatcodetable\hyxmp@cct
  \savecatcodetable\hyxmp@cct
\fi
```

`\hyxmp@prot@us` Define an underscore character that's protected from being converted into a space when passed to `\hyxmp@add@to@xml`. `\hyxmp@prot@us` is used within `write_xmp_font_list` (below) in particular to typeset filenames that may contain underscores.

```
\bgroup
  \catcode`\_=11
  \gdef\hyxmp@prot@us{_}%
\egroup
```

Here we define a Lua function, `write_xmp_font_list`, that writes font information to the XMP packet.

```
\ifLuaTeX
  \begin{luacode*}
function write_xmp_font_list (cct)
    local function show_field(name, ...)
    for i = 1, select("#", ...) do
      local val = select(i, ...)
      if val then
        local xml = string.gsub(val, "&", "&amp;")
        xml = string.gsub(xml, "<", "&lt;")
        xml = string.gsub(xml, ">", "&gt;")
        xml = string.gsub(xml, "_", "\\hyxmp@prot@us ")
        tex.print(cct, "___________<stFnt:" .. name .. ">" ..
                  xml .. "</stFnt:" .. name .. ">^^J%")
        return
      end
    end
  end
  tex.print(cct, "\\hyxmp@add@to@xml{%")
  tex.print(cct, "______<xmpTPg:Fonts>^^J%")
  tex.print(cct, "________<rdf:Bag>^^J%")
  for i, f in font.each() do
    tex.print(cct, "__________<rdf:li rdf:parseType=\"Resource\">^^J%")
    if f.filename then
      local fname = string.gsub(f.filename, "^harfloaded:(.*)", "%1")
      local info = fontloader.info(fname)
      if info then
        show_field("fontFace", info.fullname)
        show_field("fontFamily", info.familyname)
        show_field("fontName", info.fontname)
        show_field("versionString", info.version)
      end
      local baseName = string.gsub(f.filename, ".*[/\\](.*)", "%1")
      show_field("fontFileName", baseName)
    else
      show_field("fontName", f.psname, f.fullname, f.name)
    end
    if f.format and f.format ~= "unknown" then
      show_field("fontType", f.format)
    end
    tex.print(cct, "__________</rdf:li>^^J%")
  end
  tex.print(cct, "________</rdf:Bag>^^J%")
  tex.print(cct, "______</xmpTPg:Fonts>^^J%")
  tex.print(cct, "}")
end
  \end{luacode*}
\fi
```

3.6.13 XMP extension schemata

Not all of the schemata supported by hyperxmp are predefined by XMP. PDF/A conversion would normally fail for documents that employ "custom" schemata. However, this problem can be circumvented by declaring non-standard schemata in the XMP packet itself, following a technique described in a PDF Association technical note [14]. In this section, we declare only those schemata we actually use.

`\hyxmp@check@iptc@data` `\hyxmp@iptc@data` Define `\hyxmp@iptc@data` as the concatenation of all IPTC photo metadata supplied by the document.

```
\newcommand*{\hyxmp@check@iptc@data}{%
  \edef\hyxmp@iptc@data{%
    \@pdfcontactaddress
    \@pdfcontactcity
    \@pdfcontactregion
    \@pdfcontactpostcode
    \@pdfcontactcountry
    \@pdfcontactphone
    \@pdfcontactemail
    \@pdfcontacturl
  }%
}%
```

`\hyxmp@check@prism@data` `\hyxmp@prism@data` Define `\hyxmp@prism@data` as the concatenation of all PRISM metadata supplied by the document.

```
\newcommand*{\hyxmp@check@prism@data}{%
  \edef\hyxmp@prism@data{%
    \@pdfbookedition
    \@pdfbytes
    \@pdfdoi
    \@pdfeissn
    \@pdfisbn
    \@pdfissn
    \@pdfissuenum
    \@pdfnumpages
    \@pdfpagerange
    \@pdfpublication
    \@pdfpubtype
    \@pdfsubtitle
    \@pdfurl
    \@pdfvolumenum
  }%
}%
```

`\hyxmp@check@jav@data` `\hyxmp@jav@data` Define `\hyxmp@jav@data` as the concatenation of all JAV metadata supplied by the document.

```
\newcommand*{\hyxmp@check@jav@data}{%
  \edef\hyxmp@jav@data{%
    \@pdfpubstatus
```

```
  }%
}
```

`\hyxmp@begin@extension@decls` Begin a block of XML tags that indicates we're declaring one or more extension schemata.

```
\newcommand*{\hyxmp@begin@extension@decls}{%
  \hyxmp@add@to@xml{%
______<pdfaExtension:schemas>^^J%
________<rdf:Bag>^^J%
  }%
}
```

`\hyxmp@end@extension@decls` End the block of XML tags begun by `\hyxmp@begin@extension@decls`.

```
\newcommand*{\hyxmp@end@extension@decls}{%
  \hyxmp@add@to@xml{%
________</rdf:Bag>^^J%
______</pdfaExtension:schemas>^^J%
  }%
}
```

`\hyxmp@begin@ext@decl` Begin the declaration of a single extension schema. `\hyxmp@begin@ext@decl` accepts the schema's name, prefix, and namespace URI.

```
\newcommand*{\hyxmp@begin@ext@decl}[3]{%
  \hyxmp@add@to@xml{%
__________<rdf:li rdf:parseType="Resource">^^J%
____________<pdfaSchema:schema>#1</pdfaSchema:schema>^^J%
____________<pdfaSchema:prefix>#2</pdfaSchema:prefix>^^J%
____________<pdfaSchema:namespaceURI>#3</pdfaSchema:namespaceURI>^^J%
____________<pdfaSchema:property>^^J%
______________<rdf:Seq>^^J%
  }%
}%
```

`\hyxmp@end@ext@decl` End the declaration of a single extension schema.

```
\newcommand*{\hyxmp@end@ext@decl}{%
  \hyxmp@add@to@xml{%
______________</rdf:Seq>^^J%
____________</pdfaSchema:property>^^J%
__________</rdf:li>^^J%
  }%
}%
```

`\hyxmp@declare@property` Declare a single extension-schema property. `\hyxmp@declare@property` takes as input an optional type (defaults to Text) and a mandatory name, category, and description.

```
\newcommand{\hyxmp@declare@property}[4][Text]{%
  \hyxmp@add@to@xml{%
________________<rdf:li rdf:parseType="Resource">^^J%
__________________<pdfaProperty:name>}%
```

```
  \xdef\hyxmp@xml{\hyxmp@xml#2}%
  \hyxmp@add@to@xml{</pdfaProperty:name>^^J%
__________________<pdfaProperty:valueType>#1</pdfaProperty:valueType>^^J%
__________________<pdfaProperty:category>#3</pdfaProperty:category>^^J%
__________________<pdfaProperty:description>#4</pdfaProperty:description>^^J%
_________________</rdf:li>^^J%
  }%
}%
```

`\hyxmp@declare@field` Declare a single field in a custom datatype required by an extension schema. `\hyxmp@declare@field` takes as input an optional type (defaults to Text) and a mandatory name and description.

```
\newcommand{\hyxmp@declare@field}[3][Text]{%
  \hyxmp@add@to@xml{%
_______________________<rdf:li rdf:parseType="Resource">^^J%
_________________________<pdfaField:name>#2</pdfaField:name>^^J%
_________________________<pdfaField:valueType>#1</pdfaField:valueType>^^J%
_________________________<pdfaField:description>#3</pdfaField:description>^^J%
_______________________</rdf:li>^^J%
  }%
}
```

`\hyxmp@pdf@extensions` Declare the Adobe PDF schema.

```
\newcommand*{\hyxmp@pdf@extensions}{%
  \hyxmp@begin@ext@decl
      {Adobe PDF Schema}%
      {pdf}%
      {http://ns.adobe.com/pdf/1.3/}%
  \hyxmp@declare@property
      {Trapped}%
      {internal}%
      {Indication if the document has been modified to include trapping information}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@mm@extensions` Declare the XMP Media Management schema.

```
\newcommand*{\hyxmp@mm@extensions}{%
  \hyxmp@begin@ext@decl
      {XMP Media Management Schema}%
      {xmpMM}%
      {http://ns.adobe.com/xap/1.0/mm/}%
  \hyxmp@declare@property
      [URI]
      {DocumentID}%
      {internal}%
      {UUID based identifier for all versions and renditions of a document}%
  \hyxmp@declare@property
      [URI]
      {InstanceID}%
```

```
      {internal}%
      {UUID based identifier for specific incarnation of a document}%
  \hyxmp@declare@property
      {VersionID}%
      {internal}%
      {Document version identifier}%
  \hyxmp@declare@property
      [RenditionClass]%
      {RenditionClass}%
      {internal}%
      {The manner in which a document is rendered}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@pdfa@id@extensions` Declare the PDF/A Identification schema [13].

```
\newcommand*{\hyxmp@pdfa@id@extensions}{%
  \hyxmp@begin@ext@decl
      {PDF/A Identification Schema}%
      {pdfaid}%
      {http://www.aiim.org/pdfa/ns/id/}%
  \hyxmp@declare@property
      [Integer]%
      {part}%
      {internal}%
      {Part of PDF/A standard}%
  \hyxmp@declare@property
      {conformance}%
      {internal}%
      {Conformance level of PDF/A standard}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@pdfua@id@extensions` Declare the PDF/UA Universal Accessibility schema.

```
\newcommand*{\hyxmp@pdfua@id@extensions}{%
  \hyxmp@begin@ext@decl
      {PDF/UA Universal Accessibility Schema}%
      {pdfuaid}%
      {http://www.aiim.org/pdfua/ns/id/}%
  \hyxmp@declare@property
      [Integer]%
      {part}%
      {internal}%
      {Part of ISO 14289 standard}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@pdfx@id@extensions` Declare the schema used pre-PDF/X-4. Because Adobe Acrobat DC (at least) defines this even for PDF/X-4 and later, we follow suit.

```
\newcommand*{\hyxmp@pdfx@id@extensions}{%
```

```
  \ifx\hyxmp@pdfx@major\empty
  \else
    \hyxmp@begin@ext@decl
        {Adobe Document Info PDF/X eXtension Schema}%
        {pdfx}%
        {http://ns.adobe.com/pdfx/1.3/}%
    \hyxmp@declare@property
        {GTS_PDFXVersion}%
        {internal}%
        {ID of PDF/X standard}%
    \hyxmp@declare@property
        {GTS_PDFXConformance}%
        {internal}%
        {Conformance level of PDF/X standard}%
    \hyxmp@end@ext@decl
  \fi
```

Declare the schema used in PDF/X-4 and later versions.

```
  \@tempcnta=0\hyxmp@pdfx@major\relax
  \ifnum\@tempcnta>3
    \hyxmp@begin@ext@decl
        {PDF/X ID Schema}%
        {pdfxid}%
        {http://www.npes.org/pdfx/ns/id/}%
    \hyxmp@declare@property
        {GTS_PDFXVersion}%
        {internal}%
        {ID of PDF/X standard}%
    \hyxmp@end@ext@decl
  \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 IPTC metadata. Declaring the IPTC metadata we support enables the document to be converted to PDF/A format.

```
\newcommand*{\hyxmp@iptc@extensions}{%
  \hyxmp@begin@ext@decl
      {IPTC Core Schema}%
      {Iptc4xmpCore}%
      {http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/}%
  \hyxmp@declare@property
      [ContactInfo]
      {CreatorContactInfo}
      {external}
      {Document creator's contact information}
```

We can't call `\hyxmp@end@ext@decl` because we need first need to define the Iptc4xmpCore:ContactInfo structure.

```
  \hyxmp@add@to@xml{%
```

```
_____________</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/</pdfaType:
___________________<pdfaType:prefix>Iptc4xmpCore</pdfaType:prefix>^^J%
___________________<pdfaType:description>%
                     Basic set of information to get in contact with a person%
                   </pdfaType:description>^^J%
___________________<pdfaType:field>^^J%
_____________________<rdf:Seq>^^J%
  }%
  \hyxmp@declare@field
      {CiAdrCity}%
      {Contact information city}%
  \hyxmp@declare@field
      {CiAdrCtry}%
      {Contact information country}%
  \hyxmp@declare@field
      {CiAdrExtadr}%
      {Contact information address}%
  \hyxmp@declare@field
      {CiAdrPcode}%
      {Contact information local postal code}%
  \hyxmp@declare@field
      {CiAdrRegion}%
      {Contact information regional information such as state or province}%
  \hyxmp@declare@field
      {CiEmailWork}%
      {Contact information email address(es)}%
  \hyxmp@declare@field
      {CiTelWork}%
      {Contact information telephone number(s)}%
  \hyxmp@declare@field
      {CiUrlWork}%
      {Contact information Web URL(s)}%
  \hyxmp@add@to@xml{%
_____________________</rdf:Seq>^^J%
___________________</pdfaType:field>^^J%
_________________</rdf:li>^^J%
_____________</rdf:Seq>^^J%
___________</pdfaSchema:valueType>^^J%
__________</rdf:li>^^J%
  }%
}
```

`\hyxmp@prism@extensions` Because PRISM metadata are not recognized by the PDF/A standard, PDF/A conversion would normally fail for documents that utilize PRISM metadata. Declaring

the PRISM metadata we support enables the document to be converted to PDF/A format.

```
\newcommand*{\hyxmp@prism@extensions}{%
  \hyxmp@begin@ext@decl
      {PRISM Basic Metadata}%
      {prism}%
      {http://prismstandard.org/namespaces/basic/3.0/}%
  \hyxmp@declare@property
      {complianceProfile}%
      {internal}%
      {PRISM specification compliance profile to which this document adheres}%
  \hyxmp@declare@property
      {publicationName}%
      {external}%
      {Publication name}%
  \hyxmp@declare@property
      {aggregationType}%
      {external}%
      {Publication type}%
  \hyxmp@declare@property
      {bookEdition}%
      {external}%
      {Edition of the book in which the document was published}%
  \hyxmp@declare@property
      {volume}%
      {external}%
      {Publication volume number}%
  \hyxmp@declare@property
      {number}%
      {external}%
      {Publication issue number within a volume}%
  \hyxmp@declare@property
      {pageRange}%
      {external}%
      {Page range for the document within the print version of its publication}%
  \hyxmp@declare@property
      {issn}%
      {external}%
      {ISSN for the printed publication in which the document was published}%
  \hyxmp@declare@property
      {eIssn}%
      {external}%
      {ISSN for the electronic publication in which the document was published}%
  \hyxmp@declare@property
      {isbn}%
      {external}%
      {ISBN for the publication in which the document was published}%
  \hyxmp@declare@property
      {doi}%
```

```
      {external}%
      {Digital Object Identifier for the document}%
  \hyxmp@declare@property
      [URL]
      {url}%
      {external}%
      {URL at which the document can be found}%
  \hyxmp@declare@property
      [Integer]
      {byteCount}%
      {internal}%
      {Approximate file size in octets}%
  \hyxmp@declare@property
      [Integer]
      {pageCount}%
      {internal}%
      {Number of pages in the print version of the document}%
  \hyxmp@declare@property
      {subtitle}%
      {external}%
      {Document's subtitle}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@jav@extensions` Because JAV metadata are not recognized by the PDF/A standard, PDF/A conversion would normally fail for documents that utilize JAV metadata. Declaring the JAV metadata we support enables the document to be converted to PDF/A format.

```
\newcommand*{\hyxmp@jav@extensions}{%
  \hyxmp@begin@ext@decl
      {NISO/ALPSP Journal Article Versions}%
      {jav}%
      {http://www.niso.org/schemas/jav/1.0/}%
  \hyxmp@declare@property
      [Closed Choice of Text]%
      {journal_article_version}%
      {external}%
      {Article status, one of "AO", "SMUR", "AM", "P", "VoR", "CVoR", or "EVoR"}%
  \hyxmp@end@ext@decl
}%
```

`\hyxmp@declare@extensions` Declare all XMP extension schemata. We'll always have at least one, the XMP Media Management extensions, because we automatically generate xmpMM:DocumentID and xmpMM:InstanceID values.

```
\newcommand*{\hyxmp@declare@extensions}{%
  \hyxmp@begin@extension@decls
```

Declare the Adobe PDF schema (always present).

```
  \hyxmp@pdf@extensions
```

Declare the XMP Media Management extensions (always present).

```
\hyxmp@mm@extensions
```

Declare the PDF/A Identification extensions, but only when generating a PDF/A document.

```
\ifHy@pdfa
  \hyxmp@pdfa@id@extensions
\fi
```

Conditionally declare the PDF/UA Universal Accessibility extensions.

```
\ifx\@pdfuapart\@empty
\else
  \hyxmp@pdfua@id@extensions
\fi
```

\next Conditionally declare the PDF/X extensions.

```
\ifx\@pdfxversion\@empty
\else
  \hyxmp@pdfx@id@extensions
\fi
```

Conditionally declare IPTC photo metadata extensions.

```
\ifx\hyxmp@iptc@data\@empty
\else
  \hyxmp@iptc@extensions
\fi
```

Conditionally declare PRISM basic metadata extensions.

```
\ifx\hyxmp@prism@data\@empty
\else
  \hyxmp@prism@extensions
\fi
```

Conditionally declare JAV metadata.

```
\ifx\hyxmp@jav@data\@empty
\else
  \hyxmp@jav@extensions
\fi
\hyxmp@end@extension@decls
}
```

3.6.14 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/">^^J%
__<rdf:RDF %
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\hyxmp@hash">^^J%
____<rdf:Description rdf:about=""^^J%
```

Specify every namespace we can potentially use, even the ones we end up not actually using.

```
_____________________xmlns:pdf="http://ns.adobe.com/pdf/1.3/"^^J%
_____________________xmlns:xmpRights="http://ns.adobe.com/xap/1.0/rights/"^^J%
_____________________xmlns:dc="http://purl.org/dc/elements/1.1/"^^J%
_____________________xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/"^^J%
_____________________xmlns:xmp="http://ns.adobe.com/xap/1.0/"^^J%
_____________________xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"^^J%
_____________________xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent\hyxmp@hash"^^
_____________________xmlns:pdfaid="http://www.aiim.org/pdfa/ns/id/"^^J%
_____________________xmlns:pdfuaid="http://www.aiim.org/pdfua/ns/id/"^^J%
_____________________xmlns:pdfx="http://ns.adobe.com/pdfx/1.3/"^^J%
_____________________xmlns:pdfxid="http://www.npes.org/pdfx/ns/id/"^^J%
_____________________xmlns:prism="http://prismstandard.org/namespaces/basic/3.0/"^^J%
_____________________xmlns:jav="http://www.niso.org/schemas/jav/1.0/"^^J%
_____________________xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"^^J%
_____________________xmlns:stFnt="http://ns.adobe.com/xap/1.0/sType/Font\hyxmp@hash"^^J%
_____________________xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/"^^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%
  }%
```

Declare non-standard schemata.

```
  \hyxmp@check@iptc@data
  \hyxmp@check@prism@data
  \hyxmp@check@jav@data
  \hyxmp@declare@extensions
```

Insert all the metadata we know how to insert.

```
  \hyxmp@pdf@schema
```

```
  \hyxmp@xmpRights@schema
  \hyxmp@dc@schema
  \hyxmp@photoshop@schema
  \hyxmp@xmp@basic@schema
  \hyxmp@pdfa@id@schema
  \hyxmp@pdfua@id@schema
  \hyxmp@pdfx@id@schema
  \hyxmp@mm@schema
  \hyxmp@iptc@schema
  \hyxmp@prism@schema
  \hyxmp@jav@schema
  \hyxmp@xmptpg@schema
  \hyxmp@add@to@xml{%
____</rdf:Description>^^J%
__</rdf:RDF>^^J%
</x:xmpmeta>^^J%
\hyxmp@padding
<?xpacket end="w"?>^^J%
  }%
}
```

3.7 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" [4] 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
              \hyxmp@jobname.tex's XMP metadata will *not* be\MessageBreak
              embedded in the resulting file}%
          }{%
            \hyxmp@embed@packet@pdfmark
          }%
        \fi
      \fi
    \fi
  \fi
}
```

3.7.1 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
}
```

3.7.2 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}%
}
```

3.7.3 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
  }%
}
```

3.7.4 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
}
```

3.7.5 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
    >>
  }%
}
```

3.8 Final clean-up

As explained in Section 3.1, all invocations of `\AtEndPreamble` have been stored in `\hyxmp@aep@toks` rather than executed. Now that hyperxmp has been initialized completely, it is finally safe to execute the accumulated `\AtEndPreamble` stanzas.

```
\the\hyxmp@aep@toks
```

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 Help Wanted

Comma handling 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 10–11. 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/">
  <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/"
             xmlns:xmpRights="http://ns.adobe.com/xap/1.0/rights/"
             xmlns:dc="http://purl.org/dc/elements/1.1/"
             xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/"
             xmlns:xmp="http://ns.adobe.com/xap/1.0/"
             xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
             xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
             xmlns:pdfaid="http://www.aiim.org/pdfa/ns/id/"
             xmlns:pdfuaid="http://www.aiim.org/pdfua/ns/id/"
             xmlns:pdfx="http://ns.adobe.com/pdfx/1.3/"
             xmlns:pdfxid="http://www.npes.org/pdfx/ns/id/"
             xmlns:prism="http://prismstandard.org/namespaces/basic/3.0/"
             xmlns:jav="http://www.niso.org/schemas/jav/1.0/"
             xmlns:xmpTPg="http://ns.adobe.com/xap/1.0/t/pg/"
             xmlns:stFnt="http://ns.adobe.com/xap/1.0/sType/Font#"
             xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/"
             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>
```

⋮

[over 200 lines of boilerplate definitions not shown]

⋮

```
        </rdf:Bag>
      </pdfaExtension:schemas>
      <pdf:Keywords>
        energy quanta, Hertz effect, quantum physics
      </pdf:Keywords>
      <pdf:Producer>
        LuaHBTeX, Version 1.12.0 (TeX Live 2020)
      </pdf:Producer>
      <pdf:PDFVersion>1.5</pdf:PDFVersion>
```

```
<xmpRights:Marked>True</xmpRights:Marked>
<xmpRights:WebStatement>
  http://creativecommons.org/licenses/by-nc-nd/3.0/
</xmpRights:WebStatement>
<dc:format>application/pdf</dc:format>
<dc:title>
  <rdf:Alt>
    <rdf:li xml:lang="x-default">
      On a heuristic viewpoint concerning the production
      and transformation of light
    </rdf:li>
    <rdf:li xml:lang="en">
      On a heuristic viewpoint concerning the production
      and transformation of light
    </rdf:li>
    <rdf:li xml:lang="de">
      Über einen die Erzeugung und Verwandlung des Lichtes
      betreffenden heuristischen Gesichtspunkt
    </rdf:li>
  </rdf:Alt>
</dc:title>
<dc:description>
  <rdf:Alt>
    <rdf:li xml:lang="en">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:Alt>
</dc:rights>
<dc:publisher>
  <rdf:Bag>
    <rdf:li>Wiley-VCH</rdf:li>
  </rdf:Bag>
</dc:publisher>
<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>
<dc:identifier>info:lccn/50013519</dc:identifier>
<photoshop:AuthorsPosition>
  Technical Assistant, Level III
</photoshop:AuthorsPosition>
<photoshop:CaptionWriter>Scott Pakin</photoshop:CaptionWriter>
<xmp:CreateDate>2020-07-25T21:37:02-06:00</xmp:CreateDate>
<xmp:ModifyDate>2020-07-25T21:37:02-06:00</xmp:ModifyDate>
<xmp:MetadataDate>2020-07-25T21:37:02-06:00</xmp:MetadataDate>
<xmp:CreatorTool>LaTeX with hyperref package</xmp:CreatorTool>
<xmpMM:DocumentID>
  uuid:6d1ac9ec-4ff2-515a-954b-648eeb4853b0
</xmpMM:DocumentID>
<xmpMM:InstanceID>
  uuid:3e4c4182-b182-46c9-995f-754c41d13390
</xmpMM:InstanceID>
<xmpMM:VersionID>2.998e8</xmpMM:VersionID>
<xmpMM:RenditionClass>default</xmpMM:RenditionClass>
<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>
<prism:complianceProfile>three</prism:complianceProfile>
<prism:subtitle xml:lang="en-US">
  Putting that bum Maxwell in his place
</prism:subtitle>
<prism:publicationName xml:lang="de">
  Annalen der Physik
</prism:publicationName>
```

```
<prism:aggregationType>journal</prism:aggregationType>
<prism:volume>322</prism:volume>
<prism:number>6</prism:number>
<prism:pageRange>132-148</prism:pageRange>
<prism:issn>0003-3804</prism:issn>
<prism:eIssn>1521-3889</prism:eIssn>
<prism:doi>10.1002/andp.19053220607</prism:doi>
<prism:url>
  http://www.physik.uni-augsburg.de/annalen/history/einstein-papers/190517132-148.pdf
</prism:url>
<prism:byteCount>41197</prism:byteCount>
<prism:pageCount>1</prism:pageCount>
<jav:journal_article_version>VoR</jav:journal_article_version>
<xmpTPg:Fonts>
  <rdf:Bag>
    <rdf:li rdf:parseType="Resource">
      <stFnt:fontFace>LMRoman10-Regular</stFnt:fontFace>
      <stFnt:fontFamily>LM Roman 10</stFnt:fontFamily>
      <stFnt:fontName>LMRoman10-Regular</stFnt:fontName>
      <stFnt:versionString>
        2.004;PS 2.004;hotconv 1.0.49;makeotf.lib2.0.14853
      </stFnt:versionString>
      <stFnt:fontFileName>lmroman10-regular.otf</stFnt:fontFileName>
      <stFnt:fontType>opentype</stFnt:fontType>
    </rdf:li>
    <rdf:li rdf:parseType="Resource">
      <stFnt:fontFace>LMRoman17-Regular</stFnt:fontFace>
      <stFnt:fontFamily>LM Roman 17</stFnt:fontFamily>
      <stFnt:fontName>LMRoman17-Regular</stFnt:fontName>
      <stFnt:versionString>
        2.004;PS 2.004;hotconv 1.0.49;makeotf.lib2.0.14853
      </stFnt:versionString>
      <stFnt:fontFileName>lmroman17-regular.otf</stFnt:fontFileName>
      <stFnt:fontType>opentype</stFnt:fontType>
    </rdf:li>
    <rdf:li rdf:parseType="Resource">
      <stFnt:fontFace>LMRoman12-Regular</stFnt:fontFace>
      <stFnt:fontFamily>LM Roman 12</stFnt:fontFamily>
      <stFnt:fontName>LMRoman12-Regular</stFnt:fontName>
      <stFnt:versionString>
        2.004;PS 2.004;hotconv 1.0.49;makeotf.lib2.0.14853
      </stFnt:versionString>
      <stFnt:fontFileName>lmroman12-regular.otf</stFnt:fontFileName>
      <stFnt:fontType>opentype</stFnt:fontType>
    </rdf:li>
    <rdf:li rdf:parseType="Resource">
      <stFnt:fontName>cmr12</stFnt:fontName>
    </rdf:li>
    <rdf:li rdf:parseType="Resource">
      <stFnt:fontName>cmr8</stFnt:fontName>
```

```
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmr6</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmmi12</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmmi8</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmmi6</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmsy10</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmsy8</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmsy6</stFnt:fontName>
          </rdf:li>
          <rdf:li rdf:parseType="Resource">
            <stFnt:fontName>cmex10</stFnt:fontName>
          </rdf:li>
        </rdf:Bag>
      </xmpTPg:Fonts>
      <xmpTPg:NPages>1</xmpTPg:NPages>
    </rdf:Description>
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

References

[1] Beverley Acreman, Claire Bird, Catherine Jones, Peter McCracken, Cliff Morgan, John Ober, Evan Owens, T. Scott Plutchak, Bernie Rous, and Andrew Wray. Journal Article Versions (JAV): Recommendations of the NISO/ALPSP JAV Technical Working Group. Recommended practice, National Information Standards Organization, Baltimore, Maryland, USA, April 2008. ISBN 978-1-880124-79-6. Available from `https://www.niso.org/sites/default/files/2017-08/RP-8-2008.pdf`.

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

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

[4] 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`.

[5] 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`.

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

[7] 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`.

[8] International Digital Enterprise Alliance, Inc. *Publishing Requirements for Industry Standard Metadata, Version 3.0: PRISM Basic Metadata Specification*, October 12, 2012. Available from `http://www.prismstandard.org/specifications/3.0/PRISM_Basic_Metadata_3.0.htm`.

[9] International Digital Enterprise Alliance, Inc. *Publishing Requirements for Industry Standard Metadata, Version 3.0: PRISM Controlled Vocabularies Specification*, October 4, 2012. Available from `http://www.prismstandard.org/specifications/3.0/PRISM_CV_Spec_3.0.pdf`.

[10] 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`.

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

[12] 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`.

[13] 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`.

[14] 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`.

[15] 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

that the user can change the output directory 38

v5.9

General: At Karl Berry's request, rename `add_byteCount` to the less generic-sounding `hyperxmp-add-bytecount` . . . 19

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

L

M

N

O

P

Q

R

S