The hyperxmp package*

Scott Pakin
scott+hyxmp@pakin.org

April 17, 2011

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 recently been promoting XMP [3]—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 of PDF, JPEG, HTML, or any other type, it is trivial for a program (or human) to locate, extract, and—using any standard XML parser—process the embedded XMP metadata.

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

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

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

*This document corresponds to hyperxmp v1.2, dated 2011/04/17.

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

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

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

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

- authors (`dc:creator`)
- copyright (`dc:rights`)
- date (`dc:date`)
- document identifier (`xapMM:DocumentID`)
- document instance identifier (`xapMM:InstanceID`)
- format (`dc:format`)
- keywords (`pdf:Keyword` and `dc:subject`)
- license URL (`xapRights:WebStatement`)
- PDF-generating tool (`pdf:Producer`)
- summary (`dc:description`)
- title (`dc:title`)

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

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

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

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

2 Usage

hyperxmp provides no commands of its own. Rather, it processes some of the package options honored by hyperref. To use hyperxmp, merely put a `\usepackage{hyperxmp}` somewhere in your document's preamble. hyperxmp will construct its XMP data using the following hyperref options:

- `pdfauthor`,
- `pdfkeywords`,
- `pdfproducer`,
- `pdfsubject`, and
- `pdftitle`.

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

- `pdfcopyright` and
- `pdflicenseurl`.

`\pdfcopyright` defines the copyright text. `pdflicenseurl` defines a URL that points to the document's license agreement.

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

```
\documentclass{article}
\usepackage{hyperxmp}
\usepackage{hyperref}
\title{%
  On a heuristic viewpoint concerning the production and
  transformation of light}
\author{Albert Einstein}
\hypersetup{%
  pdftitle={%
    On a heuristic viewpoint concerning the production and
    transformation of light},
  pdfauthor={Albert Einstein},
```

```
  pdfcopyright={Copyright (C) 1905, Albert Einstein},
  pdfsubject={photoelectric effect},
  pdfkeywords={energy quanta, Hertz effect, quantum physics}
}
\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
- LaTeX + Dvipdfm
- LaTeX + Dvips + Ghostscript

The combination LaTeX + Dvips + Adobe Acrobat Distiller *almost* works but is hampered by a Distiller bug (at least in version 7.0.5) that incorrectly replaces the first author with the complete list of authors in the generated PDF file. That is, if a document's authors are Jack Napier, Edward Nigma, and Harvey Dent, Distiller replaces "Jack Napier" with a single author named "Jack Napier, Edward Nigma, Harvey Dent" and leaves "Edward Nigma" and "Harvey Dent" as the second and third authors, respectively. Until Adobe fixes this bug, Adobe Acrobat Distiller is not recommended for use with hyperxmp.

Besides the approaches listed above, other approaches may work as well but have not been tested. Note that in many TeX distributions `ps2pdf` is a convenience script that calls Ghostscript with the appropriate options for converting PostScript to PDF and `dvipdf` is a convenience script that calls `dvips` and `ps2pdf`; both `ps2pdf` and `dvipdf` should be compatible with hyperxmp.

The resulting PDF file will contain an XMP packet that looks something like this:

```
<?xpacket begin="???" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="3.1-702">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
            <pdf:Keywords>energy quanta, Hertz effect,
            quantum physics</pdf:Keywords>
            <pdf:Producer>pdfeTeX-1.10b</pdf:Producer>
      </rdf:Description>
      <rdf:Description rdf:about=""
             xmlns:dc="http://purl.org/dc/elements/1.1/">
```

```
            <dc:format>application/pdf</dc:format>
            <dc:title>
               <rdf:Alt>
                  <rdf:li xml:lang="x-default">On a heuristic viewpoint
                  concerning the production and transformation of
                  light</rdf:li>
               </rdf:Alt>
            </dc:title>
            <dc:description>
               <rdf:Alt>
                  <rdf:li xml:lang="x-default">photoelectric effect</rdf:li>
               </rdf:Alt>
            </dc:description>
            <dc:rights>
               <rdf:Alt>
                  <rdf:li xml:lang="x-default">Copyright (C) 1905,
                  Albert Einstein</rdf:li>
               </rdf:Alt>
            </dc:rights>
            <dc:creator>
               <rdf:Seq>
                  <rdf:li>Albert Einstein</rdf:li>
               </rdf:Seq>
            </dc:creator>
            <dc:subject>
               <rdf:Bag>
                  <rdf:li>energy quanta</rdf:li>
                  <rdf:li>Hertz effect</rdf:li>
                  <rdf:li>quantum physics</rdf:li>
               </rdf:Bag>
            </dc:subject>
            <dc:date>
               <rdf:Seq>
                  <rdf:li>2006-04-19</rdf:li>
               </rdf:Seq>
            </dc:date>
         </rdf:Description>
         <rdf:Description rdf:about=""
               xmlns:xapMM="http://ns.adobe.com/xap/1.0/mm/">
            <xapMM:DocumentID>uuid:c4188820-aef2-0a82-626ce4182b62</xapMM:DocumentID>
            <xapMM:InstanceID>uuid:9b62b67f-d754-626c-4c959595fd75</xapMM:InstanceID>
         </rdf:Description>
      </rdf:RDF>
   </x:xmpmeta>
   <?xpacket end="w"?>
```

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 example 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}`

If you desperately need to include a comma within an author or keyword list you can define your own comma macro as follows:

```
\bgroup
\catcode`,=11
\gdef\mycomma{,}
\egroup
```

Thereafter, you can use `\mycomma` as a literal comma:

```
pdfauthor={Napier\mycomma\ Jack,
           Nigma\mycomma\ Edward,
           Dent\mycomma\ Harvey}
```

3 Implementation

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

3.1 Initial preparation

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

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

3.2 Integration with hyperref

An important design decision underlying hyperxmp is that the package should integrate seamlessly with hyperref. To that end, hyperxmp takes its XMP metadata from the hyperref `pdftitle`, `pdfauthor`, `pdfsubject`, and `pdfkeywords` options plus two new options, `pdfcopyright` and `pdflicenseurl`, introduced by hyperxmp.

```
\RequirePackage{keyval}
```

`\@pdfcopyright` Prepare to store the document's copyright statement. 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 the macro name with our package-specific `\hyxmp@` prefix.

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

`\@pdflicenseurl` Prepare to store the URL containing the document's license agreement. 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 the macro name with our package-specific `\hyxmp@` prefix.

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

`\@pdfauthortitle` Prepare to store the author's position/title (e.g., Staff Writer). 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 the macro name with our package-specific `\hyxmp@` prefix.

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

`\@pdfcaptionwriter` Prepare to store the name of the person who inserted the hyperxmp metadata. 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 the macro name with our package-specific `\hyxmp@` prefix.

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

`\hyxmp@find@metadata` Issue a warning message if the author failed to include any metadata at all.

```
\newcommand*{\hyxmp@find@metadata}{%
  \ifx\@pdfauthor\@empty
    \ifx\@pdfcopyright\@empty
      \ifx\@pdfkeywords\@empty
        \ifx\@pdflicenseurl\@empty
          \ifx\@pdfauthortitle\@empty
            \ifx\@pdfcaptionwriter\@empty
              \ifx\@pdfsubject\@empty
                \ifx\@pdftitle\@empty
                  \PackageWarningNoLine{hyperxmp}{%
\jobname.tex did not specify any metadata to\MessageBreak
include in the XMP packet.\space\space Please see the hyperxmp\MessageBreak
documentation for instructions on how to provide\MessageBreak
metadata values to hyperxmp%
                  }%
                \fi
              \fi
            \fi
          \fi
```

```
        \fi
      \fi
    \fi
  \fi
}
```

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

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

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

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

3.3 Manipulating author-supplied data

The author provides metadata information to hyperxmp via package options to hyperref or via the hyperref `\hypersetup` command. The functions in this section convert author-supplied lists (e.g., `pdfkeywords={foo, bar, baz}`) into LaTeX lists (e.g., `\@elt {foo} \@elt {bar} \@elt {baz}`) that can be more easily manipulated (Section 3.3.1); define macros for the XML entites `<`, `>`, and `&` (Section 3.3.2); trim spaces off the ends of strings (Section 3.3.3); and, in Section 3.3.4, convert text to XML (e.g., from `<scott+hyxmp@pakin.org>` to `&lt;scott+hyxmp@pakin.org&gt;`).

3.3.1 List manipulation

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

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

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

\hyxmp@commas@to@list@i Recursively construct macro #1 from comma-separated list #2. Stop if #2 is empty.
\next

```
\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
}
```

3.3.2 Character-code and XML entity definitions

The hyperref package invokes \pdfstringdef on its metadata parameters, setting every character to TEX category code 11 ("other"). To match against these, we have to define a few category code 11 characters of our own. Furthermore, because XMP is an XML format, we have to replace the characters "&", "<", and ">" with equivalent XML entities.

\hyxmp@xml@amp Define category code 11 ("other") versions of the character "&" and map
\hyxmp@other@amp \hyxmp@other@amp to its XML entity, &.
\hyxmp@amp

```
\bgroup
\catcode`\&=11
\gdef\hyxmp@xml@amp{&amp;}
\global\let\hyxmp@other@amp=&
\gdef\hyxmp@amp{&}
```

\hyxmp@xml@lt Define a category code 11 ("other") version of the character "<" and map
\hyxmp@other@lt \hyxmp@other@lt to its XML entity, <.

```
\catcode`\<=11
\gdef\hyxmp@xml@lt{&lt;}
\global\let\hyxmp@other@lt=<
```

\hyxmp@xml@gt Define a category code 11 ("other") version of the character ">" and map
\hyxmp@other@gt \hyxmp@other@gt to its XML entity, >.

```
\catcode`\>=11
\gdef\hyxmp@xml@gt{&gt;}
\global\let\hyxmp@other@gt=>
```

\hyxmp@other@space Define a category code 11 ("other") version of the space character.

\next
```
\def\next#1{#1}
\next{\global\let\hyxmp@other@space= } %
```

\hyxmp@other@bs Define a category code 11 ("other") version of the character "\".

```
\catcode`\|=0
\catcode`\\=11
|global|let|hyxmp@other@bs=\
|egroup
```

3.3.3 Trimming leading and trailing spaces

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

\hyxmp@trimspaces Redefine a macro as its previous value but without leading or trailing spaces. This code—as well as that for its helper macros, `\hyxmp@trimb` and `\hyxmp@trimc`—was taken almost verbatim from a solution to an *Around the Bend* puzzle [4]. Inline comments are also taken from the solution text.

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

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

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

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

```
  \begingroup
```

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

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

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

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

Transfer the trimmed text back into `#1`.

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

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

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

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

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

3.3.4 Converting text to XML

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

`\hyxmp@xmlify` `\hyxmp@xmlified` 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 `&`.

If `\pdfmark` is defined then there's a chance the user will run `dvips` on the resulting DVI file and `dvips` may convert some of the spaces to newlines, which is problematic for the proper display of an XMP packet. We therefore conditionally invoke `\hyxmp@obscure@spaces` to replace all spaces with `&#32;`.

```
\newcommand*{\hyxmp@xmlify}[1]{%
  \gdef\hyxmp@xmlified{}%
  \expandafter\hyxmp@xmlify@i#1\@empty
  \@ifundefined{pdfmark}{}{%
    \expandafter\hyxmp@obscure@spaces\expandafter{\hyxmp@xmlified}%
  }%
}
```

`\hyxmp@xmlify@i` `\hyxmp@one@token` Bind the next token in the input stream to `\hyxmp@one@token` and invoke `\hyxmp@xmlify@ii`. `\hyxmp@xmlify@i` (and therefore `\hyxmp@xmlify@ii`) is invoked on each character in the text supplied to `\hyxmp@xmlify`.

```
\def\hyxmp@xmlify@i{\futurelet\hyxmp@one@token\hyxmp@xmlify@ii}
```

`\hyxmp@xmlify@ii` `\next` Given a token in `\hyxmp@one@token`, define `\next` to consume the token, append the corresponding text to `\hyxmp@xmlified`, and recursively invoke `\hyxmp@xmlify@i` to consume subsequent tokens.

```
\def\hyxmp@xmlify@ii{%
  \if\hyxmp@one@token\hyxmp@other@lt
```

Replace "<" with `<`.

```
    \def\next##1{%
      \xdef\hyxmp@xmlified{\hyxmp@xmlified\hyxmp@xml@lt}%
      \hyxmp@xmlify@i
    }%
  \else
    \if\hyxmp@one@token\hyxmp@other@gt
```

Replace ">" with `>`.

```
      \def\next##1{%
        \xdef\hyxmp@xmlified{\hyxmp@xmlified\hyxmp@xml@gt}%
        \hyxmp@xmlify@i
      }%
    \else
      \if\hyxmp@one@token\hyxmp@other@amp
```

Replace "&" with `&`.

```
        \def\next##1{%
```

```
          \xdef\hyxmp@xmlified{\hyxmp@xmlified\hyxmp@xml@amp}%
          \hyxmp@xmlify@i
        }%
      \else
        \ifx\hyxmp@one@token\hyxmp@other@space
```

Store spaces. We need a special case for this to avoid inadvertently discarding spaces.

```
          \def\next##1{%
            \g@addto@macro\hyxmp@xmlified{ }%
            \hyxmp@xmlify@i##1%
          }%
        \else
          \if\hyxmp@one@token\hyxmp@other@bs
```

Replace \⟨*ooo*⟩ with &#⟨*ddd*⟩;. For example, \100, the octal code for "@", is represented in XML as @.

```
            \def\next##1{\futurelet\hyxmp@one@token\hyxmp@xmlify@iii}
          \else
            \ifx\hyxmp@one@token\@empty
```

End the recursion upon encountering \@empty.

```
              \def\next##1{}%
            \else
```

In most cases we merely append the next character in the input to \hyxmp@xmlified without any special processing.

```
              \def\next##1{%
                \g@addto@macro\hyxmp@xmlified{##1}%
                \hyxmp@xmlify@i
              }%
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
```

Recursively process the next character in the input stream.

```
  \next
}
```

\hyxmp@xmlify@iii
\next

hyperref's \pdfstringdef macro converts certain special characters to a backslash followed by a three-digit octal number. However, it also replaces "(" and ")" with "\(" and "\)". The \hyxmp@xmlify@iii macro is called after encountering (and removing) a backslash. If the next character in the input stream (\hyxmp@one@token) is a parenthesis, \hyxmp@xmlify@iii leaves it alone. Otherwise, \hyxmp@xmlify@iii assumes it's an octal number and replaces it with its XML equivalent.

```
\def\hyxmp@xmlify@iii{%
  \def\next##1##2##3{%
```

```
    \@tempcnta='##1##2##3
    \xdef\hyxmp@xmlified{\hyxmp@xmlified
      \hyxmp@amp\hyxmp@hash\the\@tempcnta;%
    }%
    \hyxmp@xmlify@i
  }%
  \if\hyxmp@one@token(
    \let\next=\hyxmp@xmlify@i
  \else
    \if\hyxmp@one@token)
      \let\next=\hyxmp@xmlify@i
    \fi
  \fi
  \next
}
```

`\hyxmp@obscure@spaces` The `dvips` backend rather obnoxiously word-wraps text. Doing so can cause XMP metadata to be displayed incorrectly. For example, Adobe Acrobat displays the document's `dc:rights` (copyright notice) within a single-line field. By introducing an extra line break in the middle of the copyright notice, `dvips` implicitly causes it to be truncated when displayed.

To thwart `dvips`'s word-wrapping, we define `\hyxmp@obscure@spaces` to replace each space in a given piece of text with an XML `&#32;` (space) entity.

```
\newcommand*{\hyxmp@obscure@spaces}[1]{%
  \gdef\hyxmp@xmlified{}%
  \expandafter\hyxmp@obscure@spaces@i#1 {} %
}
```

`\hyxmp@obscure@spaces@i` `\hyxmp@one@token` `\next` Do all of the work for `\hyxmp@obscure@spaces`.

```
\def\hyxmp@obscure@spaces@i #1 #2 {%
  \def\hyxmp@one@token{#2}%
  \ifx\hyxmp@one@token\@empty
    \xdef\hyxmp@xmlified{\hyxmp@xmlified#1}%
    \let\next=\relax
  \else
    \xdef\hyxmp@xmlified{\hyxmp@xmlified#1\hyxmp@amp\hyxmp@hash32;}%
    \def\next{\expandafter\hyxmp@obscure@spaces@i\expandafter#2 }%
  \fi
  \next
}
```

3.4 UUID generation

We use a linear congruential generator to produce pseudorandom UUIDs. True, this method has its flaws but it's simple to implement in TEX and is good enough for producing the XMP DocumentID and 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@iv` Invoke `\hyxmp@append@hex` four times.

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

`\hyxmp@create@uuid` Define macro `#1` as a UUID of the form "`uuid:`*xxxxxxxx*`-`*xxxx*`-`*xxxx*`-`*xxxxxxxxxxxx*" in which each "*x*" is a lowercase hexadecimal digit. 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{-}%
  \hyxmp@append@hex@iv#1%
  \g@addto@macro#1{-}%
  \hyxmp@append@hex@iv#1%
  \hyxmp@append@hex@iv#1%
  \hyxmp@append@hex@iv#1%
}
```

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

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

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

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

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

3.5 Constructing the XMP packet

An XMP packet comprises a header, "serialized XMP", padding, and a trailer [3]. The serialized XMP includes blocks of XML for various XMP schemata: Adobe PDF (Section 3.5.2), Dublin Core (Section 3.5.3), XMP Rights Management (Section 3.5.4), and XMP Media Management (Section 3.5.5). The `\hyxmp@construct@packet` macro constructs the XMP packet into `\hyxmp@xml`. It first writes the appropriate XML header, then calls the various schema-writing macros, then injects `\hyxmp@padding` as padding, and finally writes the appropriate XML trailer.

3.5.1 XMP utility functions

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

```
\newcommand*{\hyxmp@add@to@xml}[1]{%
  \bgroup
    \@tempcnta=0
    \loop
      \lccode\@tempcnta=\@tempcnta
      \advance\@tempcnta by 1
      \ifnum\@tempcnta<256
    \repeat
    \lccode`\_=`\ \relax
    \lowercase{\xdef\hyxmp@xml{\hyxmp@xml#1}}%
  \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 [3] recommends leaving a few kilobytes of whitespace at the end of each XMP packet to facilitate editing the packet in place. `\hyxmp@padding` is defined to contain 32 lines of 50 spaces and a newline apiece for a total of 1632 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@today` Define today's date in *YYYY-MM-DD* format.

```
\xdef\hyxmp@today{\the\year}%
\ifnum\month<10
  \xdef\hyxmp@today{\hyxmp@today-0\the\month}%
\else
  \xdef\hyxmp@today{\hyxmp@today-\the\month}%
\fi
\ifnum\day<10
  \xdef\hyxmp@today{\hyxmp@today-0\the\day}%
\else
  \xdef\hyxmp@today{\hyxmp@today-\the\day}%
\fi
```

3.5.2 The Adobe PDF schema

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

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

`\hyxmp@have@any` Include an Adobe PDF schema block if at least one of `\@pdfkeywords` and `\@pdfproducer` is defined.

```
  \let\hyxmp@have@any=!%
  \ifx\@pdfkeywords\@empty
    \ifx\@pdfproducer\@empty
      \let\hyxmp@have@any=\@empty
    \fi
```

```
  \fi
  \ifx\hyxmp@have@any\@empty
  \else
```

Add a block of XML to `\hyxmp@xml` that lists the document's keywords (the Keywords property) and the tools used to produce the PDF file (the Producer property).

```
    \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
__________xmlns:pdf="http://ns.adobe.com/pdf/1.3/">^^J%
    }%
    \ifx\@pdfkeywords\@empty
    \else
      \hyxmp@xmlify{\@pdfkeywords}%
      \hyxmp@add@to@xml{%
__________<pdf:Keywords>\hyxmp@xmlified</pdf:Keywords>^^J%
      }%
    \fi
    \ifx\@pdfproducer\@empty
    \else
      \hyxmp@xmlify{\@pdfproducer}%
      \hyxmp@add@to@xml{%
__________<pdf:Producer>\hyxmp@xmlified</pdf:Producer>^^J%
      }%
    \fi
    \hyxmp@add@to@xml{%
______</rdf:Description>^^J%
    }%
  \fi
}
```

3.5.3 The Dublin Core schema

`\hyxmp@rdf@dc` Given a Dublin Core property (`#1`) and a macro containing some `\pdfstringdef`-defined text (`#2`), append the appropriate block of XML to the `\hyxmp@xml` macro but only if `#2` is non-empty.

```
\newcommand*{\hyxmp@rdf@dc}[2]{%
  \ifx#2\@empty
  \else
    \hyxmp@xmlify{#2}%
    \hyxmp@add@to@xml{%
_________<dc:#1>^^J%
____________<rdf:Alt>^^J%
________________<rdf:li xml:lang="x-default">\hyxmp@xmlified</rdf:li>^^J%
____________</rdf:Alt>^^J%
_________</dc:#1>^^J%
    }%
  \fi%
}%
```

`\hyxmp@list@to@xml` Given a Dublin Core property (`#1`), an RDF array (`#2`), and a macro containing a comma-separated list (`#3`), append the appropriate block of XML to the `\hyxmp@xml` macro but only if `#3` is non-empty.

```
\newcommand*{\hyxmp@list@to@xml}[3]{%
  \ifx#3\@empty
  \else
    \hyxmp@add@to@xml{%
_________<dc:#1>^^J%
____________<rdf:#2>^^J%
    }%
    \bgroup
      \hyxmp@commas@to@list\hyxmp@list{#3}%
      \def\@elt##1{%
        \hyxmp@xmlify{##1}%
        \hyxmp@add@to@xml{%
_______________<rdf:li>\hyxmp@xmlified</rdf:li>^^J%
        }%
      }%
      \hyxmp@list
    \egroup
    \hyxmp@add@to@xml{%
____________</rdf:#2>^^J%
_________</dc:#1>^^J%
    }%
  \fi
}
```

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

```
\newcommand*{\hyxmp@dc@schema}{%
  \hyxmp@add@to@xml{%
_____<rdf:Description rdf:about=""^^J%
____________xmlns:dc="http://purl.org/dc/elements/1.1/">^^J%
_________<dc:format>application/pdf</dc:format>^^J%
  }%
  \hyxmp@rdf@dc{title}{\@pdftitle}%
  \hyxmp@rdf@dc{description}{\@pdfsubject}%
  \hyxmp@rdf@dc{rights}{\@pdfcopyright}%
  \hyxmp@list@to@xml{creator}{Seq}{\@pdfauthor}%
  \hyxmp@list@to@xml{subject}{Bag}{\@pdfkeywords}%
  \hyxmp@list@to@xml{date}{Seq}{\hyxmp@today}%
  \hyxmp@add@to@xml{%
_____</rdf:Description>^^J%
  }%
}
```

3.5.4 The XMP Rights Management schema

\hyxmp@xapRights@schema

Add properties defined by the XMP Rights Management schema to the \hyxmp@xml macro. Currently, these are only the Marked property and the WebStatement property and only if the author defined a pdflicenseurl.

```
\newcommand*{\hyxmp@xapRights@schema}{%
  \ifx\@pdflicenseurl\@empty
  \else
    \hyxmp@xmlify{\@pdflicenseurl}%
    \hyxmp@add@to@xml{%
______<rdf:Description rdf:about=""^^J%
__________xmlns:xapRights="http://ns.adobe.com/xap/1.0/rights/">^^J%
_________<xapRights:Marked>True</xapRights:Marked>^^J%
_________<xapRights:WebStatement>\hyxmp@xmlified</xapRights:WebStatement>^^J%
______</rdf:Description>^^J%
    }%
  \fi
}
```

3.5.5 The XMP Media Management schema

\hyxmp@mm@schema

Add properties defined by the XMP Media Management schema to the \hyxmp@xml macro. Although the DocumentID property is defined in the XMP specification [3], the InstanceID property is not. However, an InstanceID field is produced by Adobe Acrobat 7.0 (the latest version at the time of this writing) so it's probably worth including here.

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

3.5.6 The Photoshop schema

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

Add properties defined by the Photoshop schema to the \hyxmp@xml macro. We support only the AuthorsPosition and CaptionWriter properties, as that's all that Adobe Acrobat currently displays.

```
\gdef\hyxmp@photoshop@schema{%
  \def\hyxmp@photoshop@data{\@pdfauthortitle\@pdfcaptionwriter}%
  \ifx\@hyxmp@photoshop@data\@empty
  \else
    \hyxmp@add@to@xml{%
```

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

3.5.7 Constructing the XMP packet

`\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. The XMP specification [3] states that the argument to the begin attribute must be "the Unicode 'zero-width non-breaking space character' (U+FEFF)". However, Adobe Acrobat 7.0 (the latest version at the time of this writing) inserts the sequence $\langle EF\rangle\langle BB\rangle\langle BF\rangle$ so that's what we use here.

We explicitly mark characters $\langle EF\rangle$, $\langle BB\rangle$, $\langle BF\rangle$ as character code 12 ("letter") because the inputenc package re-encodes them as character code 13 ("active"), which causes LaTeX to abort with an "`Undefined control sequence`" error upon invoking `\hyxmp@construct@packet`.

```
\bgroup
\catcode`\^^ef=12
\catcode`\^^bb=12
\catcode`\^^bf=12
\gdef\hyxmp@construct@packet{%
  \gdef\hyxmp@xml{}%
  \hyxmp@add@to@xml{%
<?xpacket begin="^^ef^^bb^^bf" id="W5M0MpCehiHzreSzNTczkc9d"?>^^J%
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="3.1-702">^^J%
```

```
___<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\hyxmp@hash">^^J%
  }%
  \hyxmp@pdf@schema
  \hyxmp@xapRights@schema
  \hyxmp@dc@schema
  \hyxmp@photoshop@schema
  \hyxmp@mm@schema
  \hyxmp@add@to@xml{%
___</rdf:RDF>^^J%
</x:xmpmeta>^^J%
\hyxmp@padding
<?xpacket end="w"?>^^J%
  }%
}
\egroup
```

3.6 Embedding the XMP packet

The PDF specification [1] says that "a metadata stream can be attached to a document through the Metadata entry in the document catalog" so that's what we do here. hyperxmp does not currently support the embedding of XMP in any format other than PDF.

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

```
  \fi
}
```

3.6.1 Embedding using pdfTeX

\hyxmp@embed@packet@pdftex Embed the XMP packet using pdfTeX primitives.

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

3.6.2 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=/OBJ,
    Raw={/_objdef \string{hyxmp@Metadata\string} /type /stream}%
  }%
  \pdfmark{%
    pdfmark=/PUT,
    Raw={\string{hyxmp@Metadata\string}%
      <<
        /Type /Metadata
        /Subtype /XML
      >>
    }%
  }%
  \pdfmark{%
    pdfmark=/PUT,
    Raw={\string{hyxmp@Metadata\string} (\hyxmp@xml)}%
  }%
  \pdfmark{%
    pdfmark=/CLOSE,
    Raw={\string{hyxmp@Metadata\string}}%
  }%
```

Adobe's pdfmark reference [2] indicates that a metadata stream can be added to the document catalog by specifying the Metadata pdfmark instead of the PUT pdfmark. I see no advantage to this alternative mechanism and, furthermore, it

works only with Adobe Acrobat Distiller and only with versions 6.0 onwards. Consequently, hyperxmp uses the traditional PUT mechanism to point the document catalog to our metadata stream.

```
\pdfmark{%
  pdfmark=/PUT,
  Raw={\string{Catalog\string}%
    <<
      /Metadata \string{hyxmp@Metadata\string}%
    >>
  }%
}%
}
```

3.6.3 Embedding using dvipdfm

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

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

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

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

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

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

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

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

3.6.4 Embedding using XeTeX

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

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

3.7 Final clean-up

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

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

References

[1] Adobe Systems, Inc., San Jose, California. *PDF Reference, Fifth Edition: Adobe Portable Document Format Version 1.6*, November 2004. Available from `http://partners.adobe.com/public/developer/en/pdf/PDFReference16.pdf`.

[2] Adobe Systems, Inc., San Jose, California. *Adobe Acrobat 7.0.5 pdfmark Reference Manual*, October 2, 2005. Available from `http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/pdf_creation_apis_and_specs/pdfmarkReference.pdf`.

[3] Adobe Systems, Inc., San Jose, California. *XMP Specification*, June 2005. Available from `http://partners.adobe.com/public/developer/en/xmp/sdk/xmpspecification.pdf`.

[4] 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`.

Change History

Index

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