minim-xmp

version 2025/1.4

author Esger Renkema
contact minim@elrenkema.nl

This package enables simple XMP (eXtensible Metadata Platform) packet inclusion and will automatically generate pdf/a extension schemas. Use it by saying `\input minim-xmp.tex`. Thereafter, you can use `\setmetadata key {value}` and `\getmetadata key` for setting and retrieving document-level metadata values.

You do not need this package if you have your metadata ready-made in a separate file, for then you can simply say

```
\immediate\pdfextension obj uncompressed
    stream attr {/Type/Metadata /Subtype/XML}
    file {your-file.xmp}
\pdfextension catalog
    {/Metadata \pdffeedback lastobj 0 R}
```

Setting metadata

Metadata fields that contain (ordered or unordered) lists will be split on the `\metadataseparator` character; this is a semicolon by default. Alternatively, you can just make multiple assignments: these will be concatenated.

Where applicable, language alternatives can be given like `\setmetadata /de dc:title {...}` or `\setmetadata /{de_DE} dc:title {...}`. Braces are necessary in the second case because the catcode of the underscore is not 11 or 12. When no alternative is given, the value `x-default` will be used.

Instead of using `\setmetadata`, multiple fields can be set in one go with `\startmetadata`. This way is particularly useful when assigning structured data to a key (see later on). In this example, `key1` contains a normal value, `key2` language alternatives and `key3` structured data:

```
\startmetadata
    key1 {...}
    key2 {... (default) ...}
        /alt1 {...}
        /alt2 {...}
    key3
        /field1 {...}
        /field2 {...}
    stopmetadata
```

Since metadata values are read by lua as text, linebreaks and paragraphs are not preserved. You can work around this by saying `{\def\par{\Uchar"A\Uchar"A} \setmetadata abstract {...}}`.

Getting metadata

Metadata values can be retrieved again with `\getmetadata key`. This command is fully expandible.

If the data is a list, it will be returned according to the current value of `\metadataseparator`. If the data has language alternatives, the `x-default`

value will be returned: the alternatives are accessible by `\getmetadata /lang key`.

For structured types (discussed below), `\getmetadata /field key` will return the value of a single field and `\getmetadata key` will return all fields as `/{field1} {value1} /{field2} {value2} ...` (this can be used again as input to `\startmetadata`).

Supported metadata keys

Initially, the `\setmetadata` and `\getmetadata` recognise all pdf/a compatible fields in the `pdf`, `pdfaid`, `pdfuaid`, `dc`, `xmp`, `xmpMM` and `xmpRights` namespaces. Keys should be prefixed with their namespace, e.g. `dc:creator` or `xmp:CreatorTool`. Note that the `dc` namespace has lower-case fields; field names are case-sensitive.

Because the precise details of the above metadata namespaces can be confusing, you might want use one of the aliases `author` (dc:creator), `title` (dc:title), `date` (dc:date and xmp:CreateDate), `language` (dc:language), `keywords` (dc:subject and pdf:Keywords), `publisher` (dc:publisher), `abstract` (dc:description), `copyright` (dc:rights), `version` (xmpMM:VersionID) and `identifier` (dc:identifier). New aliases can be defined in the `aliases` table of the lua module.

Adding new keys and schemas

New metadata namespaces can be added in the following way:

```
require('minim-xmp').add_namespace(
  'Example namespace', 'colours',
  'http://example.com/minim/colours/', {
    -- metadata keys
    Favourite = { 'Colour', 'The author's favourite colour' },
  }, {
    -- value types
    Colour = { 'RGB Colour', {
      R = { 'Integer', 'Red component' },
      G = { 'Integer', 'Green component' },
      B = { 'Integer', 'Blue component' }
    }, prefix = 'c' },
  })
```

This will setup a namespace with prefix `colours` and one key: `Favourite`, of type `Colour`. That value type happens to be a structured type with three fields, which are also described. You can now use this namespace as

```
\startmetadata colours:Favourite /R 0 /G 0 /B 255 stopmetadata
```

or the equivalent but more verbose

```
\setmetadata/R colours:Favourite 0
\setmetadata/G colours:Favourite 0
\setmetadata/B colours:Favourite 255
```

after which the generated XMP code will be

```
  <rdf:Description rdf:about=""
      xmlns:colours="http://example.com/minim/colours/"
      xmlns:c="http://example.com/minim/colours/">
```

```
    <colours:Favourite rdf:parseType="Resource">
      <c:B>255</c:B>
      <c:G>0</c:G>
      <c:R>0</c:R>
    </colours:Favourite>
  </rdf:Description>
```

If use of the pdf/a format is detected (i.e. a `pdfaid` entry is present in the metadata), the following pdf/a extension schema will also be generated:

```
<rdf:Description rdf:about=""
    xmlns:pdfaExtension="http://www.aiim.org/pdfa/ns/extension/"
    xmlns:pdfaSchema="http://www.aiim.org/pdfa/ns/schema#"
    xmlns:pdfaProperty="http://www.aiim.org/pdfa/ns/property#"
    xmlns:pdfaType="http://www.aiim.org/pdfa/ns/type#"
    xmlns:pdfaField="http://www.aiim.org/pdfa/ns/field#" >
  <pdfaExtension:schemas>
    <rdf:Bag>
      <rdf:li rdf:parseType="Resource">
        <pdfaSchema:schema>Example namespace</pdfaSchema:schema>
        <pdfaSchema:namespaceURI>http://example.com/minim/colours/</pdfaSchema:namespaceURI>
        <pdfaSchema:prefix>colours</pdfaSchema:prefix>
        <pdfaSchema:property>
          <rdf:Seq>
            <rdf:li rdf:parseType="Resource">
              <pdfaProperty:name>Favourite</pdfaProperty:name>
              <pdfaProperty:valueType>Colour</pdfaProperty:valueType>
              <pdfaProperty:category>external</pdfaProperty:category>
              <pdfaProperty:description>The author's favourite colour</pdfaProperty:description>
            </rdf:li>
          </rdf:Seq>
        </pdfaSchema:property>
        <pdfaSchema:valueType>
          <rdf:Seq>
            <rdf:li rdf:parseType="Resource">
              <pdfaType:type>Colour</pdfaType:type>
              <pdfaType:namespaceURI>http://example.com/minim/colours/</pdfaType:namespaceURI>
              <pdfaType:prefix>c</pdfaType:prefix>
              <pdfaType:description>RGB Colour</pdfaType:description>
              <pdfaType:field>
                <rdf:Seq>
                  <rdf:li rdf:parseType="Resource">
                    <pdfaField:name>B</pdfaField:name>
                    <pdfaField:valueType>Integer</pdfaField:valueType>
                    <pdfaField:description>Blue component</pdfaField:description>
                  </rdf:li>
                  <rdf:li rdf:parseType="Resource">
                    <pdfaField:name>G</pdfaField:name>
                    <pdfaField:valueType>Integer</pdfaField:valueType>
                    <pdfaField:description>Green component</pdfaField:description>
                  </rdf:li>
                  <rdf:li rdf:parseType="Resource">
                    <pdfaField:name>R</pdfaField:name>
                    <pdfaField:valueType>Integer</pdfaField:valueType>
                    <pdfaField:description>Red component</pdfaField:description>
                  </rdf:li>
                </rdf:Seq>
              </pdfaType:field>
            </rdf:li>
          </rdf:Seq>
        </pdfaSchema:valueType>
      </rdf:li>
    </rdf:Bag>
  </pdfaExtension:schemas>
</rdf:Description>
```

You probably will not need defining your own value types, so in most cases you should omit the fifth argument to `add_namespace`. If you do define a new value type, you can specify its prefix if it differs from the namespace prefix (as is done above) and likewise its `uri` identifier if it differs from the namespace URI.

List types can be given as `'Bag TypeName'` or `'Seq TypeName'`; language alternatives as `'Lang Alt'`. All other types will be treated as `'Text'`, though for `'Boolean'`, `'Integer'` and `'Date'` some validation is performed when setting values.

Additional metadata value type handling can be defined in the `getters` and `setters` tables of the lua module. Appropriate entries to those tables will be generated automatically for new structured types (which is why you could set the colour like we did above). Value types without fields, however, will be stored and retrieved as if they were `Text` until you provide another way.

Generated XMP

All metadata will be written to the PDF file uncompressed.

The `\metadatamodification` setting controls whether XMP packets will be marked as read-only (value 0; default) or writeable (value 1). Writeable XMP packets will be padded with about 2 kB of whitespace. You can prohibit writing metadata altogether by specifying `\writedocumentmetadata = 0`.

If the document-level metadata contains values in the `pdfaid` namespace, metadata extension schemas will be appended to the document-level metadata packet automatically. These extension schema's will only include keys that have been set somewhere, though they need not have been set in the document-level metadata. No extension schemas are generated for the built-in namespaces, as they are already included in the pdf/a standards.

Licence