PDF tagging of highly-structured information; case study: bibliographies, using BibLaTeX

Ross Moore

Abstract

Here we discuss how Tagged PDF can represent the semantics inherent in the citation entries in a bibliography, using a specific example document. After first illustrating what the tagging could look like, we discuss aspects of the coding used to achieve it. Most important in this is identifying places in biblatex macro code where the structures required for tagging can be inserted, without compromising the visual presentation these macros were designed to construct, nor affecting other places where the macros might also be used.

This involves a process of 'patching' the result from executing a macro, by 'rebinding' the name to an alternative block of code that references the original expansion after having done additional work. A kind of 'hierarchical patching' is needed to ensure that modified expansions are used only in exactly the right places.

1 Tagging of bibliographic data

Visually an entry in a bibliography appears as if a single paragraph, but it often does not read easily as one. Instead the information is presented more or less as a concatenation of the values from database fields, perhaps separated by some punctuation, which could be just a comma, colon or space character. To a seasoned researcher, this structure is understandable, though conventions can be different in diverse research areas and with different kinds of publication being cited. To a novice, though, or when encountered by a reader with disabilities requiring Assistive Technology (AT), reading such information could be potentially confusing.

The primary purpose of 'tagging' is to allow a way to add 'semantics' to what could otherwise seem like arbitrary text. That is, to include a way to express the purpose of a collection of words or symbols within a given context. Whether this meaning is presented to the reader is up to the configuration of the software being used, and/or choices made by the reader him/herself. An experienced reader can ignore such choices, whereas they are available for someone who is less familiar.

In Figure 1 we can see how the names of the database fields, which have provided the textual information, can be used to 'tag' the parts of a bibliography entry. Part of the database 'resource' file is shown, to indicate the original source for what appears on the PDF page, as well as the tagging, both structural and of the content. Various features are evident.

a. Structure tags have a 'luggage-label' icon in the Tags view at right, in the image. Content tags, on the other hand, use a 'document box' or 'container' icon followed by the textual, or 'marked', content being tagged.

b. The 'entry-type' (here `@article`) and database 'key' (here `Miller_etal_2023`) appear within the Title of a ⟨Lbl⟩ structure tag. Also the 'key' is repeated with the ⟨BibEntry⟩ standard inline structure (PDF 1.7, [3] Table 338, [4] Table 5).

c. Each database field used results in a structural tag, named using a prefix of '`Bib-`'. These so-called 'Custom' tags are each `RoleMap`'d to the standard ⟨Span⟩ structure. Their content exports into other formats as plain text, or tagged with `<span>`.

d. In the list of authors, given names have been reduced to initials only, due to a biblatex setting requesting this.

e. Punctuation between field values (shown in red rectangles in the 'Tags' view) occurs as tagged content children of the ⟨Lbl⟩ and ⟨BibEntry⟩ structures. Mostly these include a trailing space character.

f. The presence of the `longdoi` field causes a line-break to occur before setting the real `doi`. This is an author-defined device to avoid breaking a text-anchor across lines, which might otherwise result in invalid PDF tagging. The actual value for `longdoi` is immaterial, just that there is one.

With the structural tagging and content as described above, it should be possible to build a tool that can extract the database field values from the tagged PDF, reconstructing a resource (`.bib` file, say) that allows the same cited files to be used within other contexts. The reconstructed resources need not be identical to what is seen in Figure 1, due to the lack of full given names. Nevertheless, the result when processed by biblatex should be identical.

Now, the structure of a bibliography environment is as an unnumbered (sub-)section. It consists of a title (here 'References cited') followed by a list of the actual entries. In PDF tagging, a ⟨List⟩ structure is a collection of ⟨LI⟩ children, each with a ⟨Lbl⟩ (label) and ⟨LBody⟩ (list body) for the bulk of the entry. For the document shown in Figure 1, an author–year citation and bibliography style is being used. Thus it is appropriate to set the `author` and `year` data

doi.org/10.47397/tb/46-2/tb143moore-tagging-biblatex

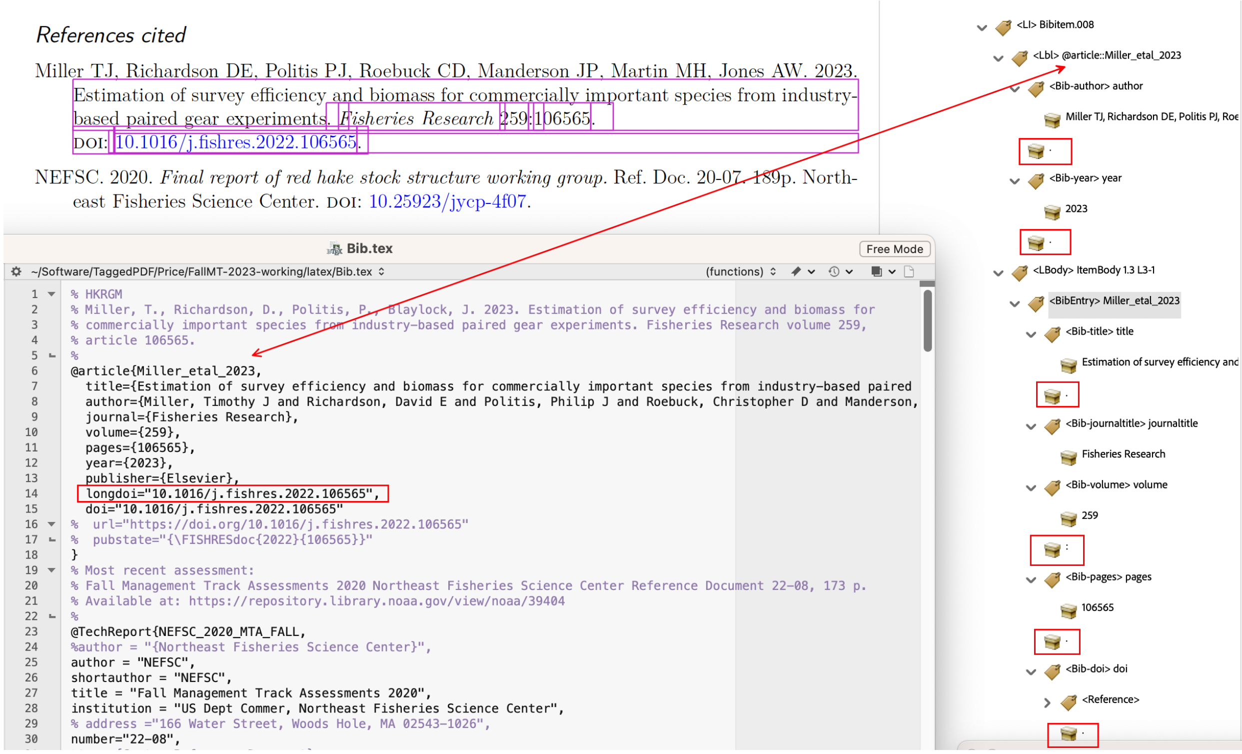

Figure 1: Tagging of bibliography data, based on database field names. PDF available on website [6].

as children of the ⟨Lbl⟩ tag. The remaining bibliographic fields come under the ⟨LBody⟩, as children of the ⟨BibEntry⟩ standard structure.

2 Bibliography entries in biblatex

The model adopted by biblatex for presentation of the field data of a bibliography entry is as 'Blocks' of 'Units', with punctuation; see [5, §4.7.1]. A database field constitutes a single Unit, with related Units grouped into a Block, having suitable punctuation between them, such as a colon, comma or simply a space. Typically a period ends a Block, with a space also when another Block follows. Finer details depend upon the particular bibliography style being used, and any customisations adopted by the author or mandated by an editorial team.

In the examples shown here an author-date style is used, imposed via the biblatex-chicago package [2], which implements specifications in the *Chicago Manual of Style* [1]. However, in the discussion which follows, describing how and where tagging constructs should occur within the LaTeX processing, similar considerations would apply when using other bibliography styles.

These 'Blocks of Units' are defined via 'bibliography driver' macros [5, §4.2.3], an example being shown in Figure 2. Field names occur within the arguments of `\usebibmacro`, `\bbx@print` and `\printfield` macros (underlined), indicating which Units are to be processed. Each `\newblock` command (boxed) denotes where punctuation is adjusted in advance of a new 'Block', [5, p. 276]. Whether any content is actually placed upon the page depends on whether the named fields exist for the specific bibliography entry. Typically 'bibliography driver' macros can be much longer and more involved than those shown in Figure 2.

Because these driver macros are so numerous (i.e., one for each entry-type), and are dependent upon the specific bibliography style being used, such macros are not suitable candidates for patching to implement PDF tagging. Instead we look at the lower-level commands for how a 'Unit' is processed, and the `\printfield` and similar macros which call them. Also we need to look at the `\blx@bibitem` macro which places each bibliography entry into a LaTeX list.

2.1 Structure tagging of a Unit

Figures 3 and 4 show internal biblatex commands which control if and how a unit is used to place the contents of a field onto the PDF page. This coding is sufficient to identify the appropriate places

```
\DeclareBibliographyDriver{book}{%
        \usebibmacro{bibindex}\usebibmacro{begentry}%
        \usebibmacro{author/editor/translator}%
        \printunit*{\bbx@ceperiod}\newblock
        \bbx@print{title}\usebibmacro{typespace}%
        \bbx@print{type}\printunit{\bbx@ceperiod}\newblock
        \usebibmacro{byauthor+byeditor+bytranslator}%
        \printunit{\bbx@ceperiod}\newblock
        \printfield{edition}\printunit{\bbx@ceperiod}\newblock
        \usebibmacro{location+publisher+date+note+pages}%
        \printunit{\bbx@ceperiod}\newblock
        \usebibmacro{eid+url+urldate}\printunit{\bbx@ceperiod}\newblock
        \usebibmacro{addendum+pageref}\usebibmacro{finentry}%
}
```

Figure 2: A rather short 'bibliography driver' macro for the `book` entry type, as defined by the biblatex-caspervector package.

```
\def\blx@begunit{%
  \toggletrue{blx@tempa}%
  \iftoggle{blx@insert}
    {\iftoggle{blx@unit}
       {\begingroup
          \let\blx@begunit\@empty
          \let\blx@endunit\@empty
          \let\blx@endnounit\@empty
          \blx@unitpunct\blx@postpunct
        \endgroup
        \global\togglefalse{blx@unit}%
        \togglefalse{blx@tempa}}
       {\blx@postpunct}%
     \iftoggle{blx@block}
       {\begingroup
          \let\blx@begunit\@empty
          \let\blx@endunit\@empty
          \let\blx@endnounit\@empty
          \newblockpunct
        \endgroup
        \global\togglefalse{blx@block}%
        \togglefalse{blx@tempa}}
       {}}
    {}%
  \blx@postpunct
  \blx@imc@resetpunctfont
  \iftoggle{blx@tempa}
    {}
    {\global\togglefalse{blx@insert}}%
  \blx@leavevmode
  \begingroup}
```

punctuation

maybe empty

mode switch

before content

Figure 3: The `\blx@begunit` macro, which sets up for placement of a database field's content as a 'Unit'. This includes placement of punctuation to separate a new unit from the previous one.

for tagging that content, as a child of a structure element named for that field.

Perhaps the most significant feature of Figure 3 is the occurrence of the `\blx@leavevmode` macro. This expands to use LaTeX's `\leavevmode`, which changes from vertical stacking of boxes into horizontal mode as with collecting letters for words and sentences, unless already in that mode. As such it indicates the end has occurred for the previous data and any appropriate punctuation. This is precisely what is needed for closing a previous structure tag and preparing for the next one. However, we need to know the name to be used for this new structure.

Looking now at Figure 4 we see where the macro `\blx@begunit` is called, which is indeed preceding the actual placement of the field's textual value, using a particular styling format via a macro named temporarily as `\blx@theformat`. However, `\blx@leavevmode` is also used elsewhere, particularly with citations; but `\blx@begunit` is the only place

```
% [<format>]{<field>}
\newrobustcmd*{\blx@imc@printfield}[2][]{%
  \blx@imc@iffieldundef{#2}
    {\blx@nounit}
    {\blx@getformat\blx@theformat{ffd}{#1}{#2}%
     \ifdefvoid\blx@theformat
       {\blx@nounit}
       {\blx@begunit
        \edef\currentfield{#2}%
        \expandafter\expandafter
        \expandafter\blx@theformat
        \expandafter\expandafter
        \expandafter{\csname abx@field@#2\endcsname}%
        \blx@endunit}}}
```

Test field-name
do nothing if not specified
do nothing if not needed
see previous figure
field content
expanded

Figure 4: This shows where `\blx@begunit` is called within the `\printfield` macro, which is an alias for `\blx@imc@printfield`. Coding is from `biblatex.sty`.

```
\def\TPDF@BLX@leavevmode{\futurelet\TPDFnext\TPDF@BLX@leavevmode@after}
\def\TPDF@BLX@leavevmode@after{\ifx\TPDFnext\begingroup
  \expandafter\TPDF@BLX@unithooks \else\expandafter\BLX@blx@leavevmode \fi}
\def\TPDF@BLX@unithooks\begingroup{\TPDF@BLX@structure@hook
  \BLX@blx@leavevmode \begingroup\TPDF@BLX@content@hook }

\def\TPDF@BLX@iffieldundef #1{\gdef\TPDF@BLX@thisfieldname{#1}%
  \TPDF@BLX@check@unitpunct
    ...  (more setup tasks) ...
  \BLX@iffieldundef{#1}}
```

Figure 5: Coding to replace `\blx@leavevmode` within bibliography entries, and capture the field-name within a container `\TPDF@BLX@thisfieldname`.

where it is immediately followed by `\begingroup`. Hence the TeX coding shown in Figure 5 can be used to detect whether the following token is that `\begingroup`. If it is not, then we just do what `\blx@leavevmode` would normally do, by executing `\BLX@blx@leavevmode`, which is a saved copy.

When the coding is indeed due to `\blx@begunit` then we execute a macro `\TPDF@BLX@unithooks`, which provides a couple of 'hook' macros. One is named `\TPDF@BLX@structure@hook`, used for tasks relating to structural tagging prior to doing the actual `\blx@leavevmode`. The other affects content tagging and so is named `\TPDF@BLX@content@hook`; it is executed inside the grouping.

In order for the coding in Figure 5 to be used, the following commands first need to have been executed:

```
\let\BLX@blx@leavevmode\blx@leavevmode
\let\blx@leavevmode\TPDF@BLX@leavevmode
```

Similarly to capture the name of the field, using the coding in Figure 4, we need to have executed:

```
\let\BLX@iffieldundef\blx@imc@iffieldundef
\let\blx@imc@iffieldundef
         \TPDF@BLX@iffieldundef
```

These 're-bindings' need to happen within the coding that sets up for tagging of a `\bibitem`, as to be discussed in Subsection 2.6. In fact there are several other commands that need patching and rebinding, in a totally analogous way. These are `\blx@imc@iflistundef`, `\blx@imc@ifnameundef`, as well as the shortened versions: `\iffieldundef` and `\iflistundef`. See [5, bottom of p. 252] for a summary of these commands.

2.2 A note on 'rebinding' macro names

Suppose `\a` has an expansion as a TeX or LaTeX macro name, also called a control sequence. What if you want to use the name `\a` to sometimes expand differently, into the expansion of another macro `\b`. To do this, first we use coding `\let\c\a` before 'rebinding' with `\let\a\b`. Now the expansion of `\a` has been changed to that of `\b`, at least within the TeX grouping (e.g., LaTeX environment) where the rebinding occurred. Furthermore, we can revert to the original expansion using `\let\a\c`, should this ever be needed within that grouping.

Another way to understand such a 'rebinding' is to realise that a TeX or LaTeX macro is essentially a (named) pointer to a block of expansion code located within the computer's memory. That is, the name is 'bound' to a memory address. Executing the macro, during TeX processing, causes this address to be looked up and the block of code found there is effectively added into the processing stream. When we want to change the coding that `\a` points to, the code `\let\a\b` rebinds the name `\a` to the address pointed-to by `\b`. By first having done `\let\c\a` we preserve a pointer to the original code, allowing us to return to it.

Compare use of `\let` with the alternative 'deferred' definition `\def\a{\b}`, which also results in the `\a` executing the expansion of `\b`. (Or using LaTeX's `\renewcommand` instead of TeX's `\def`.) This way requires two memory accesses, firstly to find the coding of `\a`, giving the name `\b`, then finding the address bound to `\b` to get its expansion. Thus using `\let` is more efficient, by accessing directly the address of the expansion of `\b`. Furthermore it is unaffected if the expansion of `\b` is changed, since `\a` was bound to the *address* of the expansion, not to its name.

When implementing tagging, we need to rebind macros already defined by biblatex, allowing extra coding to generate the tagging. Having a prefix to the name helps keep track of related expansions. E.g., `\BLX@blx@leavevmode` is bound to the address of the original expansion of `\blx@leavevmode`, whereas `\TPDF@BLX@leavevmode` is bound to the expansion that is used when tagging.

Indeed, biblatex uses a similar prefix scheme, by defining an expansion for `\blx@imc@iffieldundef`. The `@imc` part can be viewed as indicating 'internal macro code'. Then `\iffieldundef` is bound to the address of that expansion. The shorter, more user-friendly, name is used when defining driver macros for types of bibliography entry, especially within extension packages like biblatex-chicago.

Sometimes it is useful to have macros 'patch' and 'revert' commands which do rebindings, as follows:

```
\def\TPDF@BLX@⟨macro⟩@patch{%
    \let\⟨macro⟩\TPDF@BLX@⟨macro⟩}
\def\TPDF@BLX@⟨macro⟩@revert{%
    \let\⟨macro⟩\BLX@⟨macro⟩}
```

where ⟨*macro*⟩ is the name of a command defined by biblatex or in LaTeX itself, or in some other package.

2.3 Content tagging of a Unit

As seen in Figure 1 the content of a Unit is commonly the only direct child of a field's structure tag. On the other hand, it is desirable to have punctuation as children of the ⟨Lbl⟩ and ⟨BibEntry⟩ structures. Thus content tagging can usually, but not always, start as part of `\TPDF@BLX@structure@hook`.

The nature of punctuation can vary according to which fields are present within a block of units. The `\TPDF@BLX@content@hook` can have sub-hooks which are tailored to allow for customisations for a specific field-name and entry-type. That is, it calls sub-hook macros of the following kinds:

```
\TPDF@BLX@⟨field⟩@hook
\TPDF@BLX@⟨entrytype⟩@⟨field⟩@hook
```

where `\TPDF@BLX@thisfieldname` holds the value for ⟨*field*⟩, and `\abx@field@entrytype` provides the ⟨*entrytype*⟩ value. Such macro names are referenced using TeX's `\csname...\endcsname` construction.

Typically these hooks are used to locally patch or revert macros, such as the ones mentioned in the next few subsections. When a rebinding is needed in a wider scope, one can use:

```
\aftergroup\TPDF@BLX@⟨macro⟩@patch
```

within a unit preceding when the rebinding is desired; but then it may be necessary to do the corresponding 'revert' command with later units.

A biblatex driver macro can insert strings such as 'edited by', 'translated by', and abbreviations 'eds.', 'ser.', to list just a few. Such strings are not part of any field's value, so should be tagged as children of the ⟨BibEntry⟩ structure. These are specified using macros `\bibstring` and `\printtext` which expand into internal macros having parameters: *viz.*

```
\blx@bibstring{#1}{#2}{#3}
\blx@wrapbibstring{#1}{#2}{#3}{#4}
\blx@imc@printtext{#1}{#2}{#3}
```

These macros also need to be patched via rebindings, but they may be different within differing contexts. For example `\TPDF@BLX@leavevmode@revert` may be needed before `\BLX@blx@bibstring`, with `\TPDF@BLX@leavevmode@rebind` coming afterward.

To cope with instances where it is not correct to start tagging content immediately *after* the style tag, the expansion of `\TPDF@BLX@structure@hook` has a sub-hook:

```
\TPDF@BLX@⟨entrytype⟩@⟨field⟩@prehook
```

using field values as in a previous paragraph. An instance of this is when quotes are applied to the `title` (and `subtitle`) field of some entry-types; e.g., `periodical`, `thesis`, `inbook`, `incollection` and `inproceedings` using the Chicago style. The formatting is: ``⟨*title*⟩: ⟨*subtitle*⟩'' , in which the quotes and ': ' delimiter should be children of ⟨BibEntry⟩, not of the ⟨Bib-⟨*field*⟩⟩ tags; see Figure 6. This is achieved by using the `..@prehook` to start content tagging early, before the parent structure changes to become `Bib-title`. Also we rebind `\textquotedblright` from LaTeX font encodings to close tagging appropriately, thus being different to when quotes are used elsewhere, such as within a title.

Note also how the closing period and quote character are combined within a single marked-content container. This is not strictly necessary according to standards, but can be preferable to having each character being tagged on its own.

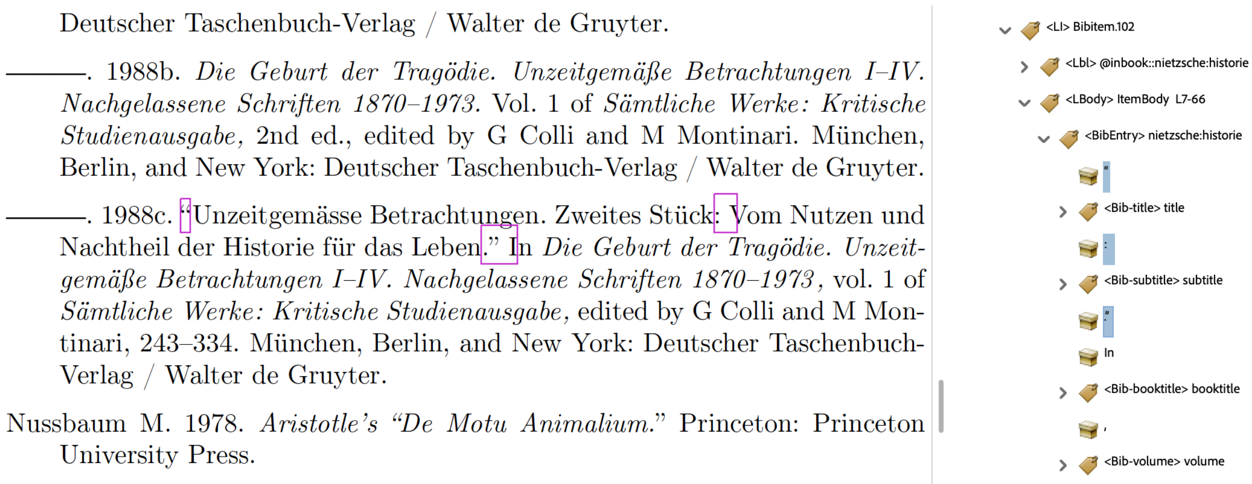

Figure 6: Tagging of quotes around the `title` and `subtitle`. PDF available on website [6].

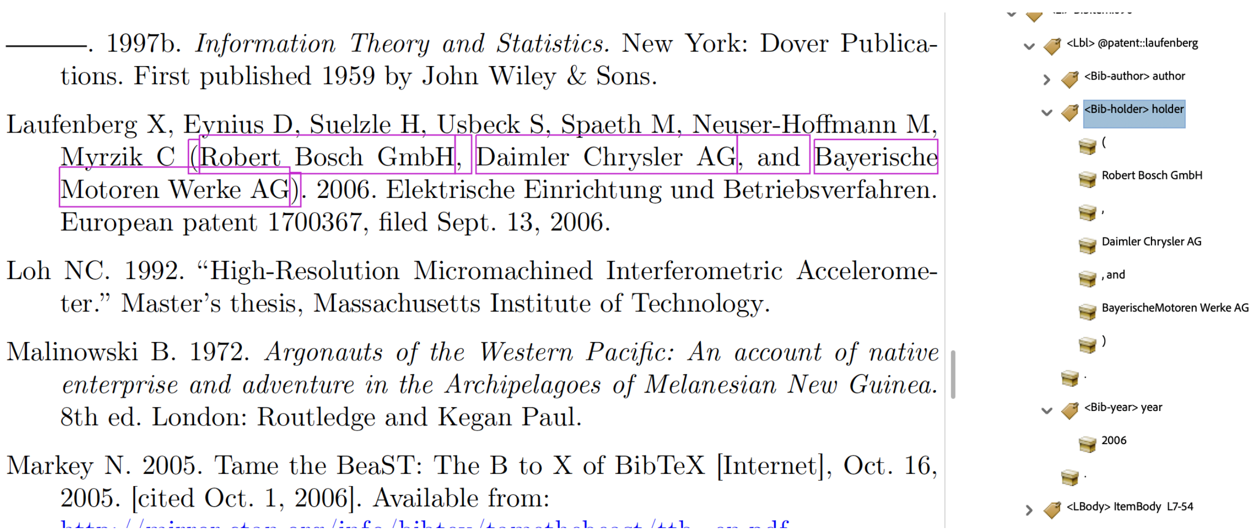

Figure 7: Tagging and punctuation in a list of patent 'holder' companies. PDF available on website [6].

2.4 Content tagging of punctuation

For a fuller description of how punctuation works in biblatex, see [5, §4.11.8]. As for tagging, the macro `\blx@unitpunct` holds the punctuation commands to finish off the previous unit, if any. From Figures 3 and 4 we see that it is used before `\blx@leavevmode`, but after `\..iffieldundef` (or `\..iflistundef` or `\..ifnameundef`, etc.). Typically this will need a rebinding to be tagged correctly, unless it consists of macros that already have a rebinding. Testing for the most commonly-used punctuation values (e.g., `\addcomma\addspace`, `\addcolon\addspace`, `\addcomma\space`, `\addcolon\space`) is done using `\TPDF@BLX@check@unitpunct` as seen in Figure 5. More customisations can be done within the `..(more setup tasks)..`, however care has to be taken with scoping.

Other macros are defined for punctuation associated with specific fields; *viz.*, `\postvolpunct`, `\journalpagespunct`, `\jtitleaddonpunct`. These, and others, apply with Chicago style. Typically they need rebindings for tagging, if different to the common values. Expansions for the rebindings can be defined globally, but applied locally through the 'hook' macros in Subsection 2.3 for appropriate ⟨*field*⟩ and ⟨*entrytype*⟩ values.

2.5 Content tagging of special characters and list delimiters

Some characters are used to prefix or suffix field values, to separate related fields, and other special purposes. Thus rebindings are needed for macros such as: `\bibleftparen`, `\bibrightparen`, `\midsentence`, `\bibnamedash`, `\finentrypunct`, `\newunitpunct`.

Even just placing a space character with `\addspace`, or non-breaking `\addnbspace`, requires special treatment for tagging.

Some database fields are of list-type, with an entry having multiple separate items. These are handled in a driver macro using `\printlist`, which uses macros `\multilistdelim` and `\finallistdelim` between and after items; see [5, p. 304]. There can be brackets, braces or parentheses around the list as a whole. Figure 7 shows an example of a `patent` entry having a list in the `holder` field. Note how the entries, delimiters and parentheses are separated into content containers, each as an individual child of the ⟨Bib-holder⟩ structure.

While it is not necessary according to tagging standards, the use of separate containers helps capture the structure of such lists. Thus these delimiter macros need rebinding to enable this. However the specific delimiter and bracketing characters may depend upon the field-type and entry-type. So any rebinding is delayed until `\TPDF@BLX@content@hook`, and is scoped to just the particular field entry by the `\begingroup` preceding this.

Similarly there can be detailed content tagging of a 'name list', using `\printnames` and delimiters `\multinamedelim` and `\finalnamedelim`, and others; see [5, p. 304]. Some of the fields using this are `author`, `editor`, `translator`. As with general lists, the specific delimiters used can depend upon the field itself, so rebinding should be delayed until needed. For example the `holder` field for Figure 7 is a name-list, but with the `author` field in Figure 1 there is no separation.

2.6 Structure tagging of a bibliography item

All the rebinding of macros described in the preceding sections are intended to apply *only* when processing bibliography items. To achieve such local rebindings, it is necessary to locate appropriate places in the processing done by the internal command `\blx@bibitem`; see Figure 8. The numbers 1–9 indicate places where it would be possible to include extra coding associated with tagging; i.e., to produce the ⟨LI⟩, ⟨Lbl⟩, ⟨LBody⟩ and ⟨BibItem⟩ structures, and their respective children. One could rebind `\blx@bibitem`, adding up to 9 'hooks' at the numbered places. However, not all of these numbered places are actually needed with the `author-date` bibliography style, as there is no need for a numbered or alphabetical label.

It turns out that a single macro rebinding suffices to provide hooks where tagging work can be done, and further patching performed. This is shown in Figure 9 which uses a pattern that matches at several places, allowing 'hooks' for extra coding at places 3, 4, 7, 8 and 9. The parameters `#1`, `#3` pick up the tokens between places 4–5 and 7–8 respectively, while `#2` picks up `\bibsentence`.

As `\blx@theenv` expands into '`bibliography`', the following rebinding:

```
\let\BLX@item@bibliography
      \blx@item@bibliography
\let\blx@item@bibliography
      \TPDF@BLX@item@bibliography
```

enables those hooks to be used at the place indicated by `\item`, which is the original value of `\blx@item@bibliography`.

Summarising, `\TPDF@BLX@preparefor@LI` generates the ⟨LI⟩ tagging and rebinds some of the punctuation and other macros; e.g., `\bibstring`, `\finentrypunct` and `\bibnamedash`. Inside the TeX grouping imposed by `\blx@beglangbib`, the hook `\TPDF@BLX@preparefor@LBody` is where most of the rebindings are done; including `\blx@leavevmode`, `\iffieldundef`, etc. and `\midsentence`. It is followed by `\abx@field@title` to allow a rescan of the tokens in the title, in case of any `\catcode` changes that may be needed for tagging. Either side of `\blx@endlangbib`, which closes a grouping, the hook `\TPDF@BLX@finish@LBody` closes the structure commenced within the grouping, with that outside being closed by `\TPDF@BLX@finish@LI`.

An indirection is used with the final definition of `\TPDF@BLX@item@bibliography` to allow for easier replacement, or addition of tracing for debugging purposes, should this be required or desired. For example, with a numerical or alphabetic bibliography style a rebinding of `\blx@thelabelnumber` would be used to generate the ⟨Lbl⟩ tagging. Now the ⟨LBody⟩ and ⟨BibItem⟩ structures could be generated a bit differently.

2.7 Iterating the list of bib-items

In biblatex a list of items is processed using the `\blx@listloop` mechanism, as follows.

```
\let\blx@do\blx@bibitem
  ...
\blx@listloop@i .. | .. | .. (items) .. ||
```

From the definition of `\blx@listloop@i`:

```
\long\def\blx@listloop@i#1|{%
  \ifblank{#1}
    {\blx@break}
    {\blx@do{#1}\blx@listloop@i}}
```

we see that `\blx@do`, bound to `\blx@bibitem`, acts on successive items. For tagging, firstly we rebind

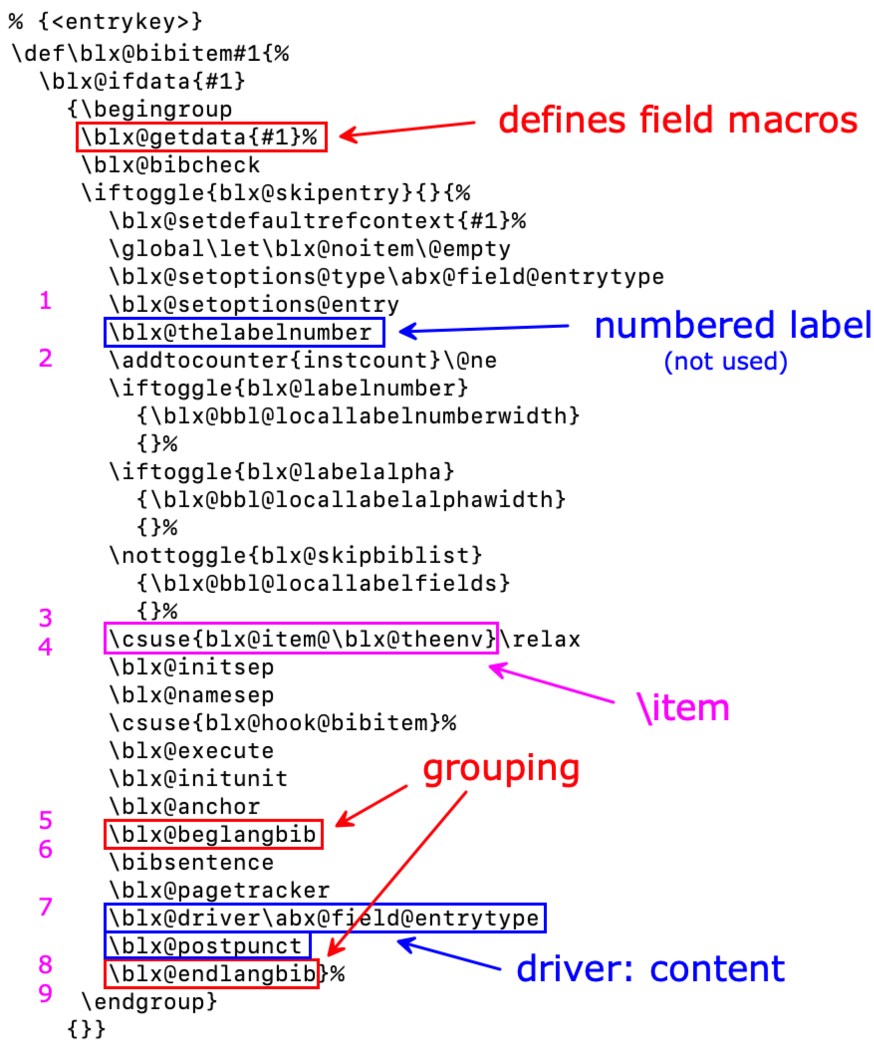

Figure 8: This shows places within `\blx@ibitem` where important steps of the processing occur. Coding is from `biblatex.sty`.

```
\def\TPDF@BLX@bibitem@pattern #1\blx@beglangbib #2\blx@pagetracker #3\blx@endlangbib {%
 \TPDF@BLX@preparefor@LI     % generates <LI> tagging and more
  \BLX@blx@item@bibliography %  --> \item
  #1\blx@beglangbib
  #2\blx@pagetracker
  \TPDF@BLX@preparefor@LBody\abx@field@title  %  setup for  <LBody> , <BibItem> etc.
   #3%    processing of the driver macro occurs here
  \TPDF@BLX@finish@LBody   % closes <BibItem> , <LBody>
  \blx@endlangbib
 \TPDF@BLX@finish@LI   % close <LI>
}
\def\TPDF@BLX@item@bibliography{\TPDF@BLX@bibitem@pattern}
```

Figure 9: The macro `\TPDF@BLX@bibitem@pattern` gives a patched expansion for `\blx@item@bibliography`, to implement tagging a list of bibliography items.

`\blx@bibitem` to `\TPDF@BLX@firstitem`, which has coding to start the ⟨L⟩ structure. This also rebinds

```
\let\blx@do\TPDF@BLX@lateritem
```

to act on subsequent items, finishing at the `||` where the item is empty. Both the first and later item macros execute the coding of Figure 8, via a saved pointer `\BLX@blx@bibitem`.

Note that it is not possible to get an ⟨L⟩ structure having no children, as `\blx@break` will occur immediately, having never called `\blx@do`.

3 Summary: hierarchical patching

From the previous (sub)sections we see that a lot of patching, through rebindings, is needed to implement tagging to the level of detail shown in the various figures. Since some of the biblatex macros being rebound are also used within other contexts, much of this rebinding needs to be done locally, scoped to be effective within a TeX grouping. Here we summarise the main features of when rebindings occur, and the names or purpose of the hooks which this patching introduces. This is similar to a 'bootstrap' procedure, whereby a piece of coding provides access to more coding at appropriate places, which then hooks further and so on, until everything needed has been loaded.

As described in Subsection 2.2, macros named like `\TPDF@BLX@`⟨*name*⟩ define expansions for the target of rebindings. These are defined globally during package-loading in the LaTeX preamble. On the other hand a macro named `\BLX@`⟨*name*⟩ is essentially a pointer to the expansion of a biblatex macro named `\`⟨*name*⟩, or sometimes `\blx@`⟨*name*⟩. These may be defined globally within the package, though some must be delayed, as the target macro is not yet defined, or not yet with the required expansion until later in the processing of a bibliography.

There are three macros that have global rebindings, since `\printbibliography` can appear more or less anywhere within a document.

- `\\blx@head@bibliography`: sets up the LaTeX `\subsubsection` (or similar) structure for the bibliography, with tagging as a ⟨Biblio⟩ custom structure, RoleMap'd to a standard ⟨Sect⟩ element. The heading is tagged at the ⟨H*n*⟩ corresponding to the sectioning level.
- `\blx@env@bibliography`: sets up the LaTeX list environment. It patches further macros:
  - `\blx@bibitem` →`\TPDF@BLX@firstitem`: starts list tagging as an ⟨L⟩ structure, rebinds `\blx@do` →`\TPDF@BLX@lateritem`.
- `\blx@item@bibliography` → `\TPDF@BLX@item@bibliography@pattern` which has hooks:
  - `\TPDF@BLX@preparefor@LI`: tags the list item ⟨LI⟩ structure; rebinds `\finentrypunct`, `\bibnamedash`.
  - `\TPDF@BLX@preparefor@LBody` most patching is done here: `\blx@leavevmode`, `\iffieldundef` and similar for handling the ⟨Lbl⟩ and ⟨Bib-⟨*field*⟩⟩ structures, having hooks:
    - `\TPDF@BLX@structure@hook`: `\TPDF@BLX@`⟨*entrytype*⟩`@`⟨*field*⟩`@prehook`
    - `\TPDF@BLX@content@hook`: `\TPDF@BLX@`⟨*field*⟩`@hook` `\TPDF@BLX@`⟨*entrytype*⟩`@`⟨*field*⟩`@hook`

    rebinds `\midsentence`, `\blx@bibstring`, `\blx@endunit`, `\blx@wrapbibstring`; rebinds delimiters, some punctuation.
  - `\TPDF@BLX@finish@LBody`: closes the ⟨BibItem⟩ and ⟨LBody⟩ structures.
  - `\TPDF@BLX@finish@LI`: closes ⟨LI⟩.
- `\blx@endenv@bibliography` closes tagging of the ⟨L⟩ and ⟨Sect⟩ structures, after having completed the bibliography itself.

4 Alternative methods of tagging.

The work described here has been implemented using the author's own system for 'Tagged PDF' from LaTeX, via a package called tpdf.sty. This is why the `TPDF@` prefix is used in the name for patched macro expansions. A link to the PDF used for Figures 1, 6 and 7 can be found on the author's webpage cited as [6].

The full tpdf package has many modules corresponding to coding needed to support other packages. For BibLaTeX the module file is named tpdfbltx.tex. Currently this also has coding to support citations, particularly in the author–date style, compatible with the bibliography style employed here. Macros defined in this module use the `TPDF@BLX@` prefix.

Other methods of tagging, such as LaTeX's in-development package named `tagpdf` will do some things differently. However, identification of *where* to apply patches to existing biblatex coding should be similar, whatever method is used for implementing the actual coding. Thus the 'hierarchical patching' as described in Section 3 should be applicable with any tagging method.

5 Further work

Other aspects of BibLaTeX which are not yet fully supported for tagging, are listed here. Future work may be done to address these aspects.

1. Tagging of 'entry sets': structurally these could be represented as a group of separate ⟨BibItem⟩ blocks, appearing under the same ⟨Lbl⟩ and ⟨LBody⟩. Alternatively, if there is a (numerical) label for each item in that list, then an ⟨L⟩ structure is needed, with an extra level of ⟨Lbl⟩, ⟨LBody⟩ and ⟨BibItem⟩ for each.
2. Support other bibliography styles: alphabetic, numeric, author–title. Some of these will require patching `\blx@thelabelnumber` (see Figure 8) to tag the ⟨Lbl⟩, and including all the fields under ⟨LBody⟩ and ⟨BibItem⟩.
3. Identify more special cases needing support via field-specific hooks: `\TPDF@BLX@`⟨*field*⟩`@hook`, `\TPDF@BLX@`⟨*entrytype*⟩`@`⟨*field*⟩`@hook` and `\TPDF@BLX@`⟨*entrytype*⟩`@`⟨*field*⟩`@prehook`.
4. Provide expanded text for common abbreviations; e.g., 'Vol.' can be spoken fully as Volume, 'eds.' as editors, 'll.' as lines, etc. Content tagging provides a key `/E` to hold the full version of such abbreviations, which are available from `\DeclareBibliographyStrings`. Some work needs to be done to capture these whenever biblatex includes the short form.
5. Support different language versions, whenever appropriate. This would include abbreviations, as in the previous item.

Concluding remarks

This work would not have been possible without the excellent work done by Philip Kime, Moritz Wemheuer and Philipp Lehman, in producing the BibLaTeX package and its documentation and examples [5]. The extensive use of indirection, modularisation and rebinding is what allows other coding to be patched-in at the places required to insert the extra processing to support PDF tagging. Here we have attempted to show how such places can be identified, and appropriate 'hooks' added.

References

[1] Chicago Manual of Style, Online. `chicagomanualofstyle.org/home.html`.

[2] Fussner, David; biblatex-chicago — Chicago style files for BibLaTeX; version 2.3b, 2024-04-15. LaTeX package, available at: `ctan.org/pkg/biblatex-chicago`. Documentation available using '`texdoc biblatex-chicago`'.

[3] ISO 32000-1:2008; Document management — Portable document format (PDF 1.7); Technical Committee ISO/TC 171/SC 2 (July 2008). `www.iso.org/standard/51502.html`.

[4] ISO 32005:2024; Document management — Portable Document Format PDF 1.7 and 2.0 structure namespace inclusion in ISO 32000-2; Technical Committee ISO/TC 171/SC 2/WG 9 (Nov. 2024).

[5] Philip Kime, Moritz Wemheuer, Philipp Lehman; BibLaTeX — Sophisticated bibliographies in LaTeX; version 3.21, 2025-07-10. LaTeX package, available at: `ctan.org/pkg/biblatex`. Documentation is available using '`texdoc biblatex`'.

[6] Moore, Ross; "FallMT 2023 PDF/UA and HTML — Accessible". Website with links to PDF and HTML versions, and related documents. `science.mq.edu.au/~ross/TaggedPDF/FallMT2023/index.html`.

[7] PDF 2.0; no-cost access to the PDF standard: ISO 32000-2 (PDF 2.0). PDF Association [8]. `pdfa.org/sponsored-standards`.

[8] PDF Association: home of the worldwide PDF technical community. `pdfa.org`.

◇ Ross Moore
School of Mathematics and Physics
Macquarie University
Sydney, Australia
`ross.moore (at) mq dot edu dot au`
`https://researchers.mq.edu.au/en/persons/ross-moore`
ORCID 0000-0001-7096-0785