Pergamon

BibLaTeX-inspired bibliography management for Typst
https://github.com/alexanderkoller/pergamon

v0.0.1, 27 August 2025
Alexander Koller

1 Introduction

Pergamon is a package for typesetting bibliographies in Typst. It is inspired by BibLaTeX, in that the way in which it typesets bibliographies can be easily customized through Typst code. Like Typst's regular bibliography management model, Pergamon can be configured to use different styles for typesetting references and citations; unlike it, these styles are all defined through Typst code, rather than CSL.

Pergamon has a number of advantages over the builtin Typst bibliographies:

- Pergamon styles are simply pieces of Typst code and can be easily configured or modified.
- The document can be easily split into different `refsections`, each of which can have its own bibliography (similar to Alexandria).
- Paper titles can be automatically made into hyperlinks - as in blinky, but much more flexibly and correctly.
- Bibliographies can be filtered, and bibliography entries programmatically highlighted, which is useful e.g. for CVs.
- References retain nonstandard Bibtex fields (unlike in Hayagriva), making it e.g. possible to split bibliographies based on keywords.

At the same time, Pergamon is very new and has a number of important limitations compared to the builtin system.

- Pergamon currently supports only bibliographies in Bibtex format, not the Hayagriva YAML format.
- Only a handful of styles are supported at this point, in contrast to the large number of available CSL styles. Pergamon comes with implementations of the BibLaTeX styles `numeric`, `alphabetic`, and `authoryear`.
- Pergamon still requires a lot of testing and tweaking.

Pergamon was an ancient Greek city state in Asia Minor. Its library was second only to the Library of Alexandria around 200 BC.

2 Example

The following piece of code typesets a bibliography using Pergamon.

```
1 #import "@preview/pergamon:0.1.0": *
2
```

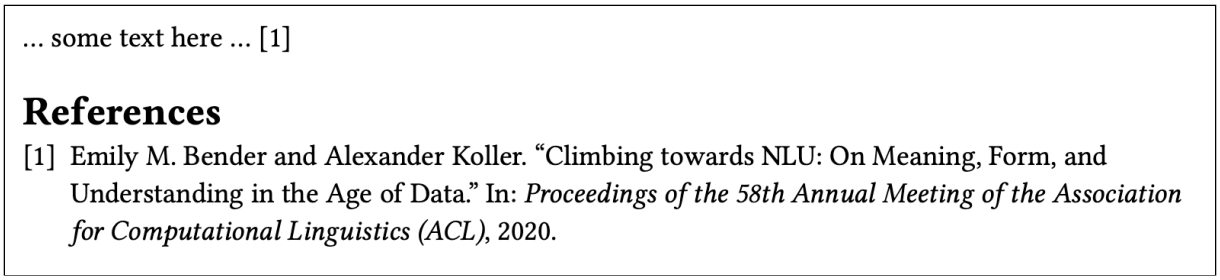

Figure 1: Example bibliography typeset with Pergamon.

```
 3 #let style = format-citation-numeric()
 4 #add-bib-resource(read("bibliography.bib"))
 5 #refsection(format-citation: style.format-citation)[
 6   ... some text here ...
 7   @bender20:_climb_nlu
 8
 9   #print-bibliography(
10        format-reference:
11              format-reference(reference-label: style.reference-label),
12        label-generator: style.label-generator,
13        sorting: "nyt")
14 ]
```

This will generate the output shown in Figure 1. Let's go through the different parts of this code one by one.

The first relevant function that is called here is `add-bib-resource`. It parses a Bibtex file and adds all the Bibtex entries to Pergamon's internal list of references. Notice that you have to `read` the Bibtex file yourself and pass its contents to `add-bib-resource` as a string. This is because Typst packages can't access files in your working directory for security reasons.

We then create a `refsection`. A `refsection` is a section of Typst content that shares a bibliography. Pergamon tracks the citations within each refsection separately and prints only those references that were cited within the current refsection when you call `print-bibliography`.

Notice that `refsection` has a parameter `format-citation` to which we passed `style.format-citation` in the example. This tells the `refsection` how to typeset citations – in the example, that `@bender20:_climb_nlu` should be rendered as "[1]".

The `format-citation` function is typically defined in a *Pergamon style*, along with a companion function `format-reference` that specifies how the individual references in the bibliography are rendered. In the example, the `style` is obtained through a call to `format-citation-numeric()` in line 4. Observe that it has an opening and closing bracket after the function name. This is because citation and reference formatters can be configured by passing arguments to this function. In the example, we just use the default configuration for the numeric style.

Finally, the example calls `print-bibliography` to typeset the bibliography itself. This is where you pass the `format-reference` function that renders the individual references. You can furthermore specify how the references should be ordered in the bibliography through the

`sorting` parameter. In the example, the references are ordered by ascending author name; then ascending publication year; then ascending title. Note also that `style.label-generator` is passed as an argument to `print-bibliography`. This function generates internal style-specific information that is used to typeset both references and citations.

All of these functions can take additional parameters that you can use to customize the appearance of the bibliography. See Section 5 for details.

3 Pergamon styles

Pergamon is highly configurable with respect to the way that references and citations are rendered. Its defining feature is that the *styles* that control the rendering process are defined as ordinary Typst functions, rather than in CSL.

There are two different types of styles in Pergamon:

- *Reference styles* define how the individual references are typeset in the bibliography.
- *Citation styles* define how citations are typeset in the text.

Obviously, the reference and citation style that is used in a refsection should fit together to avoid confusing the reader.

Pergamon comes with one predefined reference style and three predefined citation styles. We will explain these below, and then we will sketch how to define your own custom styles.

3.1 Builtin reference style

The builtin reference style is defined by the `format-reference` function. This reference style replicates the builtin reference style of BibLaTeX, with some limitations that are described in Section 6.

The builtin reference style can currently render the following BibLaTeX entry types:

- `@article`
- `@book`
- `@incollection`
- `@inproceedings`
- `@misc`
- `@thesis`

These are explained in more detail in Section 2.1.1 of the BibLaTeX documentation. Bibtex entries of a different type are typeset as a references in a dummy style which displays the entry type and Bibtex key. The aim is to eventually support all BibLaTeX styles; see issue #1 to track the progress, and feel free to submit pull requests implementing them.

The builtin `format-reference` function can be customized by passing arguments to it. These arguments are explained in detail in Section 5.3. As one example, the following arguments will change the output of the example above to look like in Figure 2:

```
1 #print-bibliography(
```

... some text here ... [1]

References

[1] Emily M. Bender and Alexander Koller (2020). Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).*

Figure 2: Bibliography with the configuration of Section 3.1.

```
2   format-reference: format-reference(
3     reference-label: style.reference-label,
4     print-date-after-authors: true,
5     format-quotes: it => it
6   ),
7   label-generator: style.label-generator,
8   sorting: "nyt")
```

We passed `true` for the argument `print-date-after-authors`. This moved the year from the end of the reference to just after the authors and put it in brackets.

We also passed the function `it => it` as the `format-quotes` argument. The builtin reference style surrounds all papers that are contained in bigger collections (in this case, a volume of conference proceedings) in quotes by applying the `format-quotes` function. By replacing the default `format-quotes` function with the identity function, we can make the quotes disappear in the output.

Pergamon exploits Typst's ability to pass functions as arguments quite heavily. This makes it cleaner in some ways than BibLaTeX, which is built on top of LaTeX, whose macros are much less flexible.

3.2 Builtin citation styles

Pergamon comes with three builtin citation styles: *alphabetic*, *numeric*, and *authoryear*. These replicate the BibLaTeX styles of the same names (see e.g. the examples on Overleaf).

Like in Typst's regular bibliography mechanism, you can write `@key` to insert a citation to the bib entry with the entry-key `key` into your document. The exact string that is inserted depends on the citation style you use. Note that unlike in regular Typst, `@key` does not resolve to the cite function in Pergamon, but to the ref function. This is because `cite` can only be used with a standard Typst bibliography. This should not make a big difference to you in practice, but it matters when you want to style citations; see Section 4.3 for details.

Whenever you write `@key`, Pergamon replaces this command with a citation to the reference generated by `format-reference`. Note that unlike in the regular Typst bibliography mechanism, `@key` is not a `cite` command, but a `ref` command.

The difference between the three builtin citation styles is illustrated in Figure 1 (numeric), Figure 3 (alphabetic), and Figure 4 (authoryear). *Numeric* and *alphabetic* both create a label for each bibliography entry; in the case of *numeric*, the label is the position in the bibliography,

... some text here ... [BK20]

References

[BK20] Emily M. Bender and Alexander Koller. "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020.

Figure 3: Bibliography with the `alphabetic` citation style.

... some text here ... (Bender and Koller 2020)

References

Emily M. Bender and Alexander Koller. "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020.

Figure 4: Bibliography with the `authoryear` citation style.

and in the case of *alphabetic*, it is a unique string consisting of the first characters of the author names and the year. In both cases, these labels are displayed next to the references and also used as the string to which `@key` expands. By contrast, *authoryear* does not display any labels next to the references; it expands `@key` to a string consisting of the last names of the authors and the year.

Some of the citation styles have options that will let you control the appearance of the citations in detail. These are documented in Section 5.3. For instance, to enclose the year in the *authoryear* style in square brackets rather than round ones, you can replace line 4 in the above example with

```
1 #let style = format-citation-authoryear(format-parens: nn(it => [(#it)]))
```

Citation forms

Each citation style offers you different *citation forms* for presenting your citation. These are documented in Table 1. Citation forms are selected using the `supplement` argument to the `ref` function:

```
1 #let citet(lbl) = ref(lbl, supplement: it => "t")
2 #citet(<bender20:_climb_nlu>)
```

Observe that you have to cannot simply pass `"t"` as the supplement; it has to be a one-parameter function that returns `"t"`. This is a regrettable consequence of the fact that `ref` supplements can be `auto`, functions, and content, but not strings. If you do not specify a citation form (`ref(<key>)` or `@key`), the `auto` citation forms will be used.

Note that the `n` citation form has special value for combining multiple citations. Unlike the builtin Typst `cite` function, Pergamon is currently not able to aggregate multiple citations into a single string (e.g. `@key1 @key2` into "(Author, 2020; Person, 2021)"; see issue #11. You

	authoryear	alphabetic	numeric
`auto`	(Bender and Koller 2020)	[BK20]	[1]
p	(Bender and Koller 2020)	–	–
t	Bender and Koller (2020)	–	–
g	Bender and Koller's (2020)	–	–
n	Bender and Koller 2020	BK20	1

Table 1: Citation forms.

can approximate this by using citations of form `n` and surrounding them with brackets and semicolons manually.

3.3 Implementing custom styles

Instead of using the builtin styles, you can also define your own Pergamon style – either a reference style or a citation style or both.

Implementing a reference style amounts to defining a Typst function that can be passed as the `format-reference` argument to `print-bibliography`. Such a function receives arguments `(index, reference, eval-mode)` containing the zero-based position of the reference in the bibliography; a reference dictionary; and the mode in which `eval` should evaluate the paper titles. A call to your function should return some `content`, which will be displayed in the bibliography.

Reference dictionaries are a central data structure of Pergamon. They represent the information contained in a single bibliography entry and are explained in detail in Section 5.1.

Implementing a citation style is a little more involved, because a citation style consists of three different functions:

- The *label generator* is passed as an argument to `print-bibliography`. It is a function that receives a reference dictionary and the bibliography position as arguments and is expected to return an array of length two. Its first element is the bib entry's *label*; it is stored under the `label` field of the reference dictionary and can contain any information that your style finds useful. The second element is a string summarizing the contents of the label. It is used to recognize when two bib entries have the same label and therefore need an "extradate" to make it unique, e.g. the letter "a" in a citation string like "Smith et al. (2025a)". It is up to your style to ensure that entries with the same label also have the same string summary.
- The *reference labeler* is passed as an argument to `format-reference`. It receives a reference dictionary and the bibliography position as arguments and returns content. If this content is not `none`, it will be typeset as the first column in the bibliography, next to the reference itself. Of the builtin citation styles, the reference labeler of `authoryear` always returns `none` (indicating a one-column bibliography layout), and the other two return their respective labels

in square brackets. Pergamon assumes that the reference labeler of a citation style either always returns `none` or never returns `none`, making for a consistent number of columns.

- The *citation formatter* is passed as an argument to `refsection`. It receives a reference dictionary and a citation form (see above) as arguments and returns content. This function generates the actual citation that is typeset into the document.

Note that the label information that the label generator produces will be stored in the `label` field of the reference dictionary. When the reference labeler and the citation formatter are called, the `label` information will still be available, allowing you to precompute any information you find useful.

4 Advanced usage

4.1 Styling the bibliography

A Pergamon bibliography is displayed as a grid, with one row per bibliography entry. The grid has one or two columns, depending on whether a first column is needed to display a label or not.

You can style this grid by passing a dictionary in the `grid-style` argument of `print-bibliography`. The values in this dictionary will be used to overwrite the default values.

4.2 Sorting the bibliography

You can furthermore control the order in which references are presented in the bibliography. To this end, you can pass a sorting string in the `sorting` argument of `print-bibliography`. See the documentation of this argument in Section 5 for details.

4.3 Styling the citations

Citations in Pergamon are `ref` elements, and can be styled using show rules. However, it is not entirely trivial to distinguish a `ref` element that represents a Pergamon citation from any other `ref` element (referring e.g. to a section or a figure).

You can use the `if-citation` function to make this distinction. The following piece of code typesets all Pergamon citations in blue:

```
#show ref: it => if-citation(it, value => {
  show link: set text(fill: blue)
  it
})
```

We have to style `link` because Pergamon rewrites citation refs into link elements.

The `value` argument contains metadata about the citation; `value.reference` is the reference dictionary. You can use this information to style citations conditionally. For instance, in order to typeset all citations to my own papers in green and all other citations in blue, I could write:

```
#show ref: it => if-citation(it, value => {
  if "Koller" in family-names(value.reference.fields.parsed-author) {
    show link: set text(fill: green)
    it
  } else {
    show link: set text(fill: blue)
    it
}})
```

4.4 Showing the entire bibliography

It is sometimes convenient to display the entire bibliography, and not just those references that were actually cited in the current refsection. You can instruct `print-bibliography` to do this using the `show-all` argument:

```
#print-bibliography(
  format-reference: format-reference(),
  show-all: true
)
```

To obtain finer control over the bibliography entries that are shown, you can use the `filtering` argument. This function receives a refrence dictionary as its argument and returns `true` if this reference should be included in the bibliography and `false` otherwise. For instance, the following call shows all journal articles and nothing else:

```
#print-bibliography(
  format-reference: format-reference(),
  show-all: true,
  filtering: reference => reference.entry_type == "article"
)
```

4.5 Highlighting references

The builtin `format-reference` function accepts a parameter `highlight`, which you can use to highlight individual references in the bibliography. The `highlight` function takes a `rendered-reference` as argument; this is a piece of content representing the entire rendered bibliography entry, just before it is printed. It also receives the corresponding reference dictionary and the position in the bibliography.

Let's say that you use the `keywords` field in your Bibtex entries to contain the keyword `highlight` if you want to highlight the paper. You can then selectively highlight references like this:

```
#let f-r = format-reference(
  highlight: (rendered, reference, index) => {
   if "highlight" in reference.fields.at("keywords", default: ()) {
      [#text(size: 8pt)[#emoji.star.box] #rendered]
```

Emily M. Bender and Alexander Koller. "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2020.

Figure 5: Highlighting a reference.

```
(
  entry_type: "inproceedings",
  entry_key: "bender20:_climb_nlu",
  fields: (
    author: "Emily M. Bender and Alexander Koller",
    award: "Best theme paper",
    booktitle: "Proceedings of the 58th Annual Meeting of the Association
                for Computational Linguistics (ACL)",
    doi: "10.18653/v1/2020.acl-main.463",
    keywords: "highlight",
    title: "Climbing towards NLU: On Meaning, Form, and Understanding
            in the Age of Data",
    year: "2020",
    parsed-author: (
      (given: "Emily M.", family: "Bender"),
      (given: "Alexander", family: "Koller"),
    ),
    sortstr-author: "Bender,Emily M. Koller,Alexander",
    parsed-editor: none,
    parsed-translator: none,
  ),
  label: ("Bender and Koller", "2020"),
  label-repr: "Bender and Koller 2020",
)
```

Figure 6: Example of a reference dictionary.

```
    } else {
      rendered
}})
```

This will place a marker before each reference with the "highlight" keyword, and will leave all other references unchanged.

5 Detailed documentation

5.1 Reference dictionaries

(explain them here)

5.2 Main functions

add-bib-resource

Parses Bibtex references and makes them available to Bibtypst. Due to architectural limitations in Typst, Bibtypst cannot read Bibtex from a file. You will therefore typically call `read` yourself, like this:

```
#add-bib-resource(read("bibliography.bib"))
```

You can call `add-bib-resource` multiple times, and this will add the contents of multiple bib files.

Parameters

```
add-bib-resource(bibtex_string: str) -> none
```

bibtex_string `str`

A Bibtex string to be parsed.

if-citation

Helper function that conditionally renders a reference to a bibliography entry. The first argument is assumed to be a Typst ref element, obtained e.g. as the argument of a show rule. If this `ref` is a citation pointing to a bibliography entry managed by Pergamon, the function passes the metadata of this bib entry to the `citation-content` function and returns the content this function generated. Otherwise, the `ref` is passed to `other-content` for further processing.

The primary purpose of `if-citation` is to facilitate the definition of show rules. A typical example is the following show rule, which colors references to my own publications green and all others blue.

```
#show ref: it => if-citation(it, value => {
   if "Koller" in family-names(value.reference.fields.parsed-author) {
     show link: set text(fill: green)
     it
   } else {
     show link: set text(fill: blue)
     it
 }})
```

Parameters

```
if-citation(
  it: ref,
  citation-content: function,
```

```
  other-content: function
) -> content
```

it `ref`

A Typst `ref` element.

citation-content `function`

A function that maps the metadata associated with a Pergamon reference to a piece of content. The metadata is a dictionary with keys `reference`, `index`, and `key`. `reference` is a reference dictionary (see Section 5.1), `key` is the key of the bib entry, and `index` is the position in the bibliography.

other-content `function`

A function that maps the `ref` to a piece of content. The default argument simply leaves the `ref` untouched, permitting other show rules to trigger and render it appropriately.

Default: `x => x`

print-bibliography

Prints the bibliography for the current `refsection()`.

Parameters

```
print-bibliography(
  format-reference: function,
  label-generator: function,
  sorting: function str none,
  show-all: bool,
  filtering: function,
  grid-style: dict,
  eval-mode: str none,
  title: str none,
  name-fields
) -> none
```

format-reference `function`

A function that renders the reference for inclusion in the printed bibliography. This function will typically be defined in a Bibtypst style, to be compatible with the `format-citation` function that is passed to `refsection()`.

`format-reference` is passed the position of the reference in the bibliography as a zero-based `int` in the first argument. It is passed the current reference in the second argument.

It returns an array of contents. The elements of this array will be laid out as the columns of a grid, in the same row, permitting e.g. bibliography layouts with one column for the reference label and one with the reference itself. For this reason, all calls to `format-reference` should return arrays of the same length.

Default: `(index, reference, eval-mode) => ([REFERENCE],)`

label-generator `function`

Generates label information for the given reference. The function takes the reference and its index in the sorted bibliography as input and returns values `(label, label-repr)`, where `label` can be anything the style finds useful for generating the citations and `label-repr` is a string representation of the label. These string representations are used to detect label collisions, which cause the generation of extradates.

The default implementation simply returns a number that is guaranteed to be unique to each reference. Styles that want to work with `extradate` will almost certainly want to pass a different function here.

The function passed as `label-generator` does not control whether labels are printed in the bibliography; it only computes information for internal use. A style can decide whether it wants to print labels through its `format-reference` function.

Note that `label-repr` *must* be a `str`.

Default: `(index, reference) => (index + 1, str(index + 1))`

sorting `function` or `str` or `none`

A function that defines the order in which references are shown in the bibliography. `sorting` takes a reference as input and returns a value that can be sorted, e.g. a number, a string, or an array of sortable values.

Alternatively, you can specify a Biblatex-style sorting string. The following strings are supported:

- `n`: author name (lastname firstname)
- `t`: paper title
- `y` or `d`: the year in which the paper was published; write `yd` or `dd` for descending order
- `v`: volume, if defined
- `a`: the contents of the `label` field (if defined); for the `alphabetic` style, this amounts to the alphabetic paper key

For instance, `"nydt"` sorts the references first by author name, then by descending year, then by title. Note that Bibtypst currently makes no distinction between the year and the full date.

If `none` or the string `"none"` is passed as the `sorting` argument, the references are sorted in an arbitrary order. There is currently no reliable support for sorting the references in the order in which they were cited in the document.

Default: `none`

show-all `bool`

Determines whether the printed bibliography should contains all references from the loaded bibliographies (`true`) or only those that were cited in the current refsection (`false`).

Default: `false`

filtering `function`

Filters which references should be included in the printed bibliography. This makes sense only if `show-all` is `true`, otherwise not all your citations will be resolved to bibliography entries. The parameter should be a function that takes a reference as argument and returns a boolean value. The printed bibliography will contain exactly those references for which the function returned `true`.

Default: `reference => true`

grid-style `dict`

A dictionary for styling the grid in which the bibliography is laid out. By default, the grid is laid out with `row-gutter: 1.2em` and `column-gutter: 0.5em`. You can overwrite these values and specify new ones with this argument; the revised style specification will be passed to the `grid` function.

Default: `(:)`

eval-mode `str` or `none`

The output of `format-reference` can be passed through the Typst eval function for final rendering. This is useful e.g. to typeset math in a paper title correctly. Pass the `eval` mode in this argument, or pass `none` if you don't want to call `eval`.

Default: `"markup"`

title `str` or `none`

The title that will be typeset above the bibliography in the document. The string given here will be rendered as a first-level heading without numbering. Pass `none` to suppress the bibliography title.

Default: `"References"`

name-fields

Bibtex fields that contain names and should be parsed as such. For each X in this list, Bibtypst will enrich the reference with a field "parsed-X" that contains a list of dictionaries of name parts, such as ("family": "Smith", "given": "John").

Default: `("author", "editor", "translator")`

refsection

Defines a section of the document that shares a bibliography. You need to load a bibliography with the "add-bibliography" function in a place that is earlier than the refsection in rendering order.

Parameters

```
refsection(
  format-citation: function,
  doc: content
) -> none
```

format-citation `function`

A function that generates the citation string for a given reference. This function will typically be defined in a Pergamon style, to be compatible with the `format-reference` function that is passed to `print-bibliography()`. Note that `format-citation` can return any content it wants, but it does not need to generate a hyperlink to the bibliography; the citation string is automatically wrapped in a `link` by Pergamon.

Default: `(reference, form) => [CITATION]`

doc `content`

The section of the document that is to be wrapped in this `refsection`.

5.3 The builtin styles

Below, we explain the arguments to the builtin reference style in detail (see Section 3.1 for the big picture).

format-citation-authoryear

Formats citations in the *authoryear* style (see Figure 4 for an example).

Parameters

```
format-citation-authoryear(format-parens: function)
```

format-parens `function`

Wraps text in round brackets. The argument needs to be a function that takes one argument (`str` or `content`) and returns `content`.

It is essential that if the argument is `none`, the function must also return `none`. This can be achieved conveniently with the `nn` function wrapper, defined in `bibtypst-styles.typ`.

Default: `nn(it => [(#it)])`

format-reference

Generates a reference formatter using the specified options. References are formatted essentially as in the standard BibLaTeX.

Parameters

```
format-reference(
  reference-label: function,
  highlight: function,
  link-titles: bool,
  print-url: bool,
  print-doi: bool,
  print-eprint: bool,
  use-author: bool,
  use-translator: bool,
  use-editor: bool,
  print-date-after-authors: bool,
  eval-mode: str,
  list-middle-delim: str,
  list-end-delim-two: str,
  list-end-delim-many: str,
  author-type-delim: str,
  subtitlepunct: str,
  format-journaltitle: function,
  format-issuetitle: function,
  format-parens: function,
  format-brackets: function,
  format-quotes: function,
  name-format: str,
  volume-number-separator: str,
  bibeidpunct: str,
  bibpagespunct: str,
  print-isbn: bool,
  bibstring: dict,
```

```
  additional-fields: function none,
  suppress-fields: array none
)
```

reference-label `function`

Generates a label to be printed in the first column of the bibliography. This is useful e.g. for use with the alphabetic and numeric citation style. By default, the function returns constant `none`, indicating that there should be no label.

This function typically comes from a predefined style (e.g. authoryear, numeric, alphabetic), or you can define your own.

Default: `(index, reference) => none`

highlight `function`

Selectively highlights certain bibliography entries. The parameter is a function that is applied at the final stage of the rendering process, where the whole rest of the entry has already been rendered. This is an opportunity to e.g. mark certain entries in the bibliography by boldfacing them or prepending them with a marker symbol.

The highlighting function accepts arguments `rendered-reference` (`str` or `content` representing the reference as it is printed), `index` (position of the reference in the bibliography), and `reference` (the Bibtex reference dictionary). It returns `content`. The default implementation simply returns the `rendered-reference` unmodified.

Default: `(rendered-reference, index, reference) => rendered-reference`

link-titles `bool`

If `true`, titles are rendered as hyperlinks pointing to the reference's DOI or URL. When both are defined, the DOI takes precedence.

Default: `true`

print-url `bool`

If `true`, prints the reference's URL at the end of the bibliography entry.

Default: `false`

print-doi `bool`

If `true`, prints the reference's DOI at the end of the bibliograph entry.

Default: `false`

print-eprint `bool`

If `true`, prints the reference's eprint information at the end of the bibliography entry. This could be a reference to arXiv or JSTOR.

Default: `true`

use-author `bool`

If `true`, prints the reference's author if it is defined.

Default: `true`

use-translator `bool`

If `true`, prints the reference's translator if it is defined. Note that support for "authors" that are not the author is currently weak. See issue 28 to track progress on this.

Default: `true`

use-editor `bool`

If `true`, prints the reference's editor if it is defined. Note that support for "authors" that are not the author is currently weak. See issue 28 to track progress on this.

Default: `true`

print-date-after-authors `bool`

If `true`, Bibtypst will print the date right after the authors, e.g. 'Smith (2020). "A cool paper".' If `false`, Bibtypst will follow the normal behavior of BibLaTeX and place the date towards the end of the reference.

Default: `false`

eval-mode `str`

When Bibtypst renders a reference, the title is processed by Typst's eval function. The `eval-mode` argument you specify here is passed as the `mode` argument to `eval`.

The default value of `"markup"` renders the title as if it were ordinary Typst content, typesetting e.g. mathematical expressions correctly.

Default: `"markup"`

list-middle-delim `str`

When typesetting lists (e.g. author names), Bibtypst will use this delimiter to combine list items before the last one.

Default: `", "`

list-end-delim-two `str`

When typesetting lists (e.g. author names), Bibtypst will use this delimiter to combine the items of lists of length two.

Default: `" and "`

list-end-delim-many `str`

When typesetting lists (e.g. author names), Bibtypst will use this delimiter in lists of length three of more to combine the final item in the list with the rest.

Default: `", and "`

author-type-delim `str`

String that is used to combine the name of an author with the author type, e.g. “Smith, editor”.

Default: `","`

subtitlepunct `str`

String that is used to combine a title with a subtitle.

Default: `"."`

format-journaltitle `function`

Renders the title of a journal as content. The default argument typesets it in italics.

Default: `it => emph(it)`

format-issuetitle `function`

Renders the title of a special issue as content. The default argument typesets it in italics.

Default: `it => emph(it)`

format-parens `function`

Wraps text in round brackets. The argument needs to be a function that takes one argument (`str` or `content`) and returns `content`.

It is essential that if the argument is `none`, the function must also return `none`. This can be achieved conveniently with the `nn` function wrapper, defined in `bibtypst-styles.typ`.

Default: `nn(it => [(#it)])`

format-brackets `function`

Wraps text in square brackets. The argument needs to be a function that takes one argument (`str` or `content`) and returns `content`.

It is essential that if the argument is `none`, the function must also return `none`. This can be achieved conveniently with the `nn` function wrapper, defined in `bibtypst-styles.typ`.

Default: `nn(it => [[#it]])`

format-quotes `function`

Wraps text in double quotes. The argument needs to be a function that takes one argument (`str` or `content`) and returns `content`.

It is essential that if the argument is `none`, the function must also return `none`. This can be achieved conveniently with the `nn` function wrapper, defined in `bibtypst-styles.typ`.

Default: `nn(it => ["#it"])`

name-format `str`

The format in which names (of authors, editors, etc.) are printed. This is an arbitrary string which may contain the placeholders `{given}` and `{family}`; these will be replaced by the person's actual name parts. You can use `{g}` and `{f}` for the first letters of the given and family name, respectively.

Default: `"{given} {family}"`

volume-number-separator `str`

Separator symbol for "volume" and "number" fields, e.g. in `@articles`.

Default: `"."`

bibeidpunct `str`

Separator symbol that connects the EID (Scopus Electronic Identifier) from other journal information.

Default: `","`

bibpagespunct `str`

Separator symbol that connects the "pages" field with related information.

Default: `","`

print-isbn `bool`

If `true`, prints the ISBN or ISSN of the reference if it is defined.

Default: `false`

bibstring `dict`

The bibstring table. This is a dictionary that maps language-independent IDs of bibliographic constants (such as "In: " or "edited by") to their language-dependent surface forms. Replace some or all of the values with your own surface forms to control the way the bibliography is rendered.

Default: `default-bibstring`

additional-fields `function` or `none`

An array of additional fields which will be printed at the end of each bibliography entry. Fields can be specified either as a string, in which case the field with that name is printed using `printfield`; or they can be specified as a function `(reference, options) => content`, in which case the returned content will be printed directly. Instead of an array, you can also pass `none` to indicate that no additional fields need to be printed.

For example, both of these will work:

```
additional-fields: ("award",)
additional-fields: ((reference, options) => ifdef(reference, "award", (:),
award => [*#award*]),)
```

Default: `none`

suppress-fields `array` or `none`

An array of field names that should not be printed. References are treated as if they do not contain values for these fields, even if the Bibtex file defines them. Instead of an array, you can also pass `none` to indicate that no fields should be suppressed.

Default: `none`

5.4 Utility functions

The following functions may be helpful in the advanced usage and customization of Pergamon.

nn

Wraps a function in `none`-handling code. `nn(func)` is a function that behaves like `func` on arguments that are not `none`, and it returns `none` if the argument is `none`. Only works for functions `func` that have a single argument.

Parameters

```
nn(func) -> function
```

family-names

Extracts the list of family names from the list of name-part dictionaries.

Parameters

```
family-names(parsed-names)
```

default-bibstring

TODO explain this

6 Limitations