sphinxcontrib-bibtex Documentation

Release 0.2.8

Matthias C. M. Troffaes

August 07, 2013

CONTENTS

Release 0.2.8

Date August 07, 2013

CONTENTS

1.1 Getting Started

1.1.1 Overview

A Sphinx extension for BibTeX style citations.

This extension allows BibTeX citations to be inserted into documentation generated by Sphinx, via a `bibliography` directive, and a `cite` role, which work similarly to LaTeX's `thebibliography` environment and `\cite` command.

For formatting, the extension relies on pybtex written by Andrey Golovizin. The extension is inspired by Matthew Brett's bibstuff.sphinxext.bibref.

- Download: http://pypi.python.org/pypi/sphinxcontrib-bibtex/#downloads

- Documentation: http://sphinxcontrib-bibtex.readthedocs.org/

- Development: http://github.com/mcmtroffaes/sphinxcontrib-bibtex/

1.1.2 Installation

Install the module with `pip install sphinxcontrib-bibtex`, or from source using `python setup.py install`. Then add:

```
extensions = ['sphinxcontrib.bibtex']
```

to your project's Sphinx configuration file `conf.py`.

1.1.3 Minimal Example

In your project's documentation, you can then write for instance:

```
See :cite:`1987:nelson` for an introduction to non-standard analysis.

.. bibliography:: refs.bib
```

where refs.bib would contain an entry:

```
@Book{1987:nelson,
  author = {Edward Nelson},
  title = {Radically Elementary Probability Theory},
```

```
    publisher = {Princeton University Press},
    year = {1987}
}
```

1.2 Usage

1.2.1 Roles and Directives

:cite:
> Create a citation to a bibliographic entry. For example:
>
> ```
> See :cite:`1987:nelson` for an introduction to non-standard analysis.
> ```
>
> which would be equivalent to the following LaTeX code:
>
> ```
> See \cite{1987:nelson} for an introduction to non-standard analysis.
> ```

.. bibliography:: refs.bib [...]
> Create bibliography for all cited references. The `all` flag forces all references to be included (equivalent to \nocite{*} in LaTeX). The `notcited` flag causes all references that were not cited to be included. The `cited` flag is recognized as well but is entirely optional. For example:
>
> ```
> .. rubric:: References
>
> .. bibliography:: refs.bib
>    :cited:
> ```
>
> which would be roughly equivalent to the following LaTeX code:
>
> ```
> \begin{thebibliography}{1}
>   \bibitem{1987:nelson}
>   Edward~Nelson
>   \newblock {\em Radically Elementary Probability Theory}.
>   \newblock Princeton University Press, 1987.
> \end{thebibliography}
> ```
>
> Note that, unlike LaTeX, the `bibliography` directive does not generate a default section title.
>
> > **Warning:** Sphinx may not be able to create an entry for `cite` keys when your `bibliography` directive resides in a different document; see *Unresolved Citations Across Documents* for more information and workarounds.
>
> You can also pick a bibliography style, using the `style` option. This is not yet quite as useful, as only `plain` and `unsrt` are supported. The `plain` style is the default.
>
> ```
> .. bibliography:: refs.bib
>    :style: unsrt
> ```
>
> All citations have numbered labels, as in the `plain` LaTeX bibliography style, regardless of the style chosen. This limitation might be lifted in a future version.
>
> > **Warning:** Sphinx will attempt to resolve references to the bibliography across all documents, so you must take care that no citation key is included more than once.
>
> You can also set the encoding of the bibliography files, using the `encoding` option.

```
.. bibliography:: refs.bib
   :encoding: latex+latin
```

Note that, usually, you want to prepend your encoding with `latex+`, in order to convert LaTeX control characters to unicode characters (for instance, to convert `\'e` into é). The latex codec is invoked by default, for your convenience. Be sure to write `\%` when you intend to format a percent sign.

1.2.2 Advanced Features

Bullet Lists and Enumerated Lists

New in version 0.2.4. You can change the type of list used for rendering the bibliography. By default, a paragraph of standard citations is generated. However, instead, you can also generate a bullet list, or an enumerated list.

```
.. bibliography:: refs1.bib
   :list: bullet
   :all:
```

```
.. bibliography:: refs2.bib
   :list: enumerated
   :all:
```

Note that citations to these types of bibliography lists will not be resolved.

For enumerated lists, you can also specify the type (default is `arabic`), and the start of the sequence (default is `1`).

```
.. bibliography:: refs2.bib
   :list: enumerated
   :enumtype: upperroman
   :start: 3
   :all:
```

The enumtype can be any of `arabic` (1, 2, 3, ...), `loweralpha` (a, b, c, ...), `upperalpha` (A, B, C, ...), `lowerroman` (i, ii, iii, ...), or `upperroman` (I, II, III, ...).

The start can be any positive integer (1, 2, 3, ...) or `continue` if you wish the enumeration to continue from the last `bibliography` directive. This is helpful if you split up your bibliography but still want to enumerate the entries continuously.

Label Prefixing

New in version 0.2.5. If you have multiple bibliographies, and experience duplicate labels, use the `labelprefix` option.

```
.. rubric:: References
```

```
.. bibliography:: refs.bib
   :cited:
   :labelprefix: A
```

```
.. rubric:: Further reading
```

```
.. bibliography:: refs.bib
   :notcited:
   :labelprefix: B
```

Filtering

New in version 0.2.7. Whilst the `cited`, `all`, and `notcited` options will cover many use cases, sometimes more advanced selection of bibliographic entries is desired. For this purpose, you can use the `filter` option:

```
.. bibliography:: refs.bib
   :list: bullet
   :filter: author % "Einstein"
```

The string specified in the filter option must be a valid Python expression.

Note: The expression is parsed using `ast.parse()` and then evaluated using an `ast.NodeVisitor`, so it should be reasonably safe against malicious code.

The filter expression supports:

- The boolean operators `and` and `or`.

- The unary operator `not`.

- Binary comparison ==, <=, <, >=, and >.

- Regular expression matching using the `%` operator, where the left hand side is the string to be matched, and the right hand side is the regular expression. Matching is case insensitive. For example:

  ```
  .. bibliography:: refs.bib
     :list: bullet
     :filter: title % "relativity"
  ```

  would include all entries that have the word "relativity" in the title.

  Note: The implementation uses `re.search()`.

- Single and double quoted strings, such as `'hello'` or `"world"`.

- Various identifiers, such as:

  - `type` is the entry type, as a lower case string (i.e. `"inproceedings"`).

  - `key` is the entry key, as a lower case string (this is because keys are considered case insensitive).

  - `cited` evaluates to `True` if the entry was cited in the document, and to `False` otherwise.

  - `True` and `False`.

  - `author` is the entry string of authors in standard format (last, first), separated by "and".

  - `editor` is similar to `author` but for editors.

  - Any other (lower case) identifier evaluates to a string containing the value of the correspondingly named field, such as `title`, `publisher`, `year`, and so on. If the item is missing in the entry then it evaluates to the empty string. Here is an example of how one would typically write an expression to filter on an optional field:

    ```
    .. bibliography:: refs.bib
       :list: bullet
       :filter: cited and year and (year <= "2003")
    ```

    which would include all cited entries that have a year that is less or equal than 2003; any entries that do not specify a year would be omitted.

Custom Formatting, Sorting, and Labelling

`pybtex` provides a very powerful way to create and register new styles, using setuptools entry points, as documented here: http://pybtex.sourceforge.net/plugins.html

One way to leverage the pybtex plugin system from within Sphinx, is to create a package stub with the desired entry points (if you know of a simpler way, please let me know!). Start with laying out your documentation folder as follows:

```
conf.py
index.rst
plugins/plugins.py
plugins/setup.py
plugins/plugins.egg-info/dependency_links.txt
plugins/plugins.egg-info/entry_points.txt
plugins/plugins.egg-info/PKG-INFO
plugins/plugins.egg-info/SOURCES.txt
plugins/plugins.egg-info/top_level.txt
...
```

The egg-info files are generated by running `python setup.py egg_info` You do not actually need to install the plugins package. Our `conf.py` will load it, and its entry points, using the `pkg_resources` module, through the following code (along with the rest of your configuration):

```python
import pkg_resources
for dist in pkg_resources.find_distributions("plugins/"):
    pkg_resources.working_set.add(dist)
```

The `plugins/setup.py` file, which is only used to generate the egg-info files, should be:

```python
from setuptools import setup

setup(
    name='plugins',
    version='0.1.0',
    entry_points={
        'pybtex.style.formatting': [
            'mystyle = plugins:MyStyle',
            ]
        },
    py_modules=['plugins']
    )
```

The actual custom style(s) reside in `plugins/plugins.py`; for instance:

```python
from pybtex.style.formatting.unsrt import Style as UnsrtStyle
from pybtex.style.template import toplevel # ... and anything else needed


class MyStyle(UnsrtStyle):
    name = 'mystyle'
    default_name_style = 'lastfirst' # 'lastfirst' or 'plain'
    default_label_style = 'number' # 'number' or 'alpha'
    default_sorting_style = 'author_year_title' # 'none' or 'author_year_title'

    def format_XXX(self, e):
        template = toplevel [
            # etc.
            ]
        return template.format_data(e)
```

The formatting code uses a very intuitive template engine. The source code for `unsrt` provides many great examples: http://bazaar.launchpad.net/~pybtex-devs/pybtex/trunk/view/head:/pybtex/style/formatting/unsrt.py

The above example only demonstrates a custom formatting style plugin. It is also possible to register custom author/editor naming plugins (using the `pybtex.style.names` group) and labelling plugins (using the `pybtex.style.labels` group).

Note: There is no documented entry point for sorting plugins, but the `pybtex.style.sorting` group appears to work.

An minimal example is available here: https://github.com/mcmtroffaes/sphinxcontrib-bibtex/tree/develop/test/custom_style

1.2.3 Known Issues and Workarounds

Tinkerer

To use the bibtex extension with Tinkerer, be sure to specify the bibtex extension first in your `conf.py` file:

```
extensions = ['sphinxcontrib.bibtex', 'tinkerer.ext.blog', 'tinkerer.ext.disqus']
```

Encoding: Percent Signs

When using the LaTeX codec (which is by default), be sure to write `\%` for percent signs at all times (unless your file contains a genuine comment), otherwise the bibtex lexer will ignore the remainder of the line.

If you don't want any LaTeX symbols to be reinterpreted as unicode, use the option `:encoding: utf` (without the `latex+` prefix).

Unresolved Citations Across Documents

If you cite something that has its bibliography in another document, then, at the moment, the extension may, or may not, realise that it has to add this citation. There are a few ways to work around this problem:

- Use the option `:all:` in the `bibliography` directive (which will simply cause all entries to be included).
- Ensure that the `bibliography` directive is processed after all `cite`s. Sphinx appears to process files in an alphabetical manner. For instance, in case you have only one file containing a `bibliography` directive, simply name that file `zreferences.rst`.

Hopefully, this limitation can be lifted in a future release.

KeyError When Using `:style: plain`

When using the `plain` style, or any style that sorts entries, pybtex may raise `KeyError: 'author'` for entries that have no author. A patch has been submitted upstream:

https://code.launchpad.net/~matthias-troffaes/pybtex/sorting-bugfix

Duplicate Labels When Using `:style: plain`

With `:style: plain`, labels are numerical, restarting at `[1]` for each `bibliography` directive. Consequently, when inserting multiple `bibliography` directives with `:style: plain`, you are bound to get duplicate labels for entries. There are a few ways to work around this problem:

- Use a single bibliography directive for all your references.

- Use the `labelprefix` option, as documented above.

- Use a style that has non-numerical labelling. Unfortunately, pybtex does not yet support such styles. A patch for non-numerical styles, such as `:style: alpha`, has been submitted upstream:

  https://code.launchpad.net/~matthias-troffaes/pybtex/label-alpha

  When this becomes part of pybtex, the plan is to change the default citation style to `:style: alpha`, as this style is also more in line with how citations are usually labelled in Sphinx.

1.3 Extension API

1.3.1 Sphinx Interface

`sphinxcontrib.bibtex.`**`setup`**(*app*)

Set up the bibtex extension:

  •register directives

  •register nodes

  •register roles

  •register transforms

  •connect events to functions

> **Parameters app** (`sphinx.application.Sphinx`) – The sphinx application.

`sphinxcontrib.bibtex.`**`init_bibtex_cache`**(*app*)

Create `app.env.bibtex_cache` if it does not exist yet. Reset citation label dictionary.

> **Parameters app** (`sphinx.application.Sphinx`) – The sphinx application.

`sphinxcontrib.bibtex.`**`purge_bibtex_cache`**(*app*, *env*, *docname*)

Remove all information related to *docname* from the cache.

> **Parameters**
>
> - **app** (`sphinx.application.Sphinx`) – The sphinx application.
>
> - **env** (`sphinx.environment.BuildEnvironment`) – The sphinx build environment.

`sphinxcontrib.bibtex.`**`process_citations`**(*app*, *doctree*, *docname*)

Replace labels of citation nodes by actual labels.

> **Parameters**
>
> - **app** (`sphinx.application.Sphinx`) – The sphinx application.
>
> - **doctree** (`docutils.nodes.document`) – The document tree.
>
> - **docname** (`str`) – The document name.

sphinxcontrib.bibtex.**process_citation_references**(*app*, *doctree*, *docname*)
> Replace text of citation reference nodes by actual labels.

> > **Parameters**

> > > - **app** (`sphinx.application.Sphinx`) – The sphinx application.

> > > - **doctree** (`docutils.nodes.document`) – The document tree.

> > > - **docname** (`str`) – The document name.

sphinxcontrib.bibtex.**check_duplicate_labels**(*app*, *env*)
> Check and warn about duplicate citation labels.

> > **Parameters**

> > > - **app** (`sphinx.application.Sphinx`) – The sphinx application.

> > > - **env** (`sphinx.environment.BuildEnvironment`) – The sphinx build environment.

1.3.2 New Doctree Roles

class sphinxcontrib.bibtex.roles.**CiteRole**(*fix_parens=False*, *lowercase=False*, *nodeclass=None*, *innernodeclass=None*, *warn_dangling=False*)

> Bases: `sphinx.roles.XRefRole`

> Class for processing the `cite` role.

> **result_nodes**(*document*, *env*, *node*, *is_ref*)
> > Transform reference node into a citation reference, and note that the reference was cited.

1.3.3 New Doctree Nodes

class sphinxcontrib.bibtex.nodes.**bibliography**(*rawsource=''*, **children*, ***attributes*)
> Node for representing a bibliography. Replaced by a list of citations by `BibliographyTransform`.

1.3.4 New Doctree Directives

class sphinxcontrib.bibtex.directives.**BibliographyDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

> Class for processing the `bibliography` directive.

> Parses the bibliography files, and produces a `bibliography` node.

> **See Also:**

> Further processing of the resulting `bibliography` node is done by `BibliographyTransform`.

> **run**()
> > Process .bib files, set file dependencies, and create a node that is to be transformed to the entries of the bibliography.

> **process_bibfile**(*bibfile*, *encoding*)
> > Check if env.bibtex_cache.bibfiles[bibfile] is still up to date. If not, parse the *bibfile* (see `update_bibfile_cache()`), and store parsed data in the bibtex cache.

> > **Parameters bibfile** (`str`) – The bib file name.
>
> > **Returns** The parsed bibliography data.
>
> > **Return type** `pybtex.database.BibliographyData`

> **update_bibfile_cache**(*bibfile*, *mtime*, *encoding*)
>
> > Parse *bibfile* (see `parse_bibfile()`), and store the parsed data, along with modification time *mtime*, in the bibtex cache.
>
> > > **Parameters**
> > >
> > > > • **bibfile** (`str`) – The bib file name.
> > > >
> > > > • **mtime** (`float`) – The bib file's modification time.
> > >
> > > **Returns** The parsed bibliography data.
> > >
> > > **Return type** `pybtex.database.BibliographyData`

> **parse_bibfile**(*bibfile*, *encoding*)
>
> > Parse *bibfile*, and return parsed data.
>
> > > **Parameters bibfile** (`str`) – The bib file name.
> > >
> > > **Returns** The parsed bibliography data.
> > >
> > > **Return type** `pybtex.database.BibliographyData`

sphinxcontrib.bibtex.directives.**process_start_option**(*value*)

> Process and validate the start option value of a `bibliography` directive. If *value* is `continue` then this function returns -1, otherwise *value* is converted into a positive integer.

1.3.5 New Doctree Transforms

class sphinxcontrib.bibtex.transforms.**BibliographyTransform**(*document*, *startnode=None*)

> Bases: `docutils.transforms.Transform`
>
> **default_priority = 10**
>
> > Priority of the transform. See http://docutils.sourceforge.net/docs/ref/transforms.html
>
> **apply**()
>
> > Transform each `bibliography` node into a list of citations.

class sphinxcontrib.bibtex.transforms.**FilterVisitor**(*entry*, *is_cited*)

> Bases: `ast.NodeVisitor`
>
> Visit the abstract syntax tree of a parsed filter expression.
>
> **entry = None**
>
> > The bibliographic entry to which the filter must be applied.
>
> **is_cited = False**
>
> > Whether the entry is cited.

sphinxcontrib.bibtex.transforms.**node_text_transform**(*node*, *transform*)

> Apply transformation to all Text nodes within node.

sphinxcontrib.bibtex.transforms.**transform_curly_bracket_strip**(*textnode*)

> Strip curly brackets from text.

sphinxcontrib.bibtex.transforms.**transform_url_command**(*textnode*)

> Convert '\url{...}' into a proper docutils hyperlink.

1.3.6 Cached Information

Classes and methods to maintain any information that is stored outside the doctree.

class `sphinxcontrib.bibtex.cache.`**`Cache`**

> Global bibtex extension information cache. Stored in `app.env.bibtex_cache`, so must be picklable.
>
> **`bibfiles`**
>> A `dict` mapping .bib file names (relative to the top source folder) to `BibfileCache` instances.
>
> **`bibliographies`**
>> Each bibliography directive is assigned an id of the form bibtex-bibliography-xxx. This `dict` maps each such id to information about the bibliography directive, `BibliographyCache`. We need to store this extra information separately because it cannot be stored in the `bibliography` nodes themselves.
>
> **`_cited`**
>> A `dict` mapping each docname to a `set` of citation keys.
>
> **`_enum_count`**
>> A `dict` mapping each docname to an `int` representing the current bibliography enumeration counter.
>
> **`add_cited`**(*key*, *docname*)
>> Add the given *key* to the set of cited keys for *docname*.
>>
>> **Parameters**
>>> • **key** (`str`) – The citation key.
>>>
>>> • **docname** (`str`) – The document name.
>
> **`get_all_cited_keys`**()
>> Yield all citation keys, sorted first by document (alphabetical), then by citation order in the document.
>
> **`get_enum_count`**(*docname*)
>> Get enumeration list counter for document *docname*.
>
> **`get_label_from_key`**(*key*)
>> Return label for the given key.
>
> **`inc_enum_count`**(*docname*)
>> Increment enumeration list counter for document *docname*.
>
> **`is_cited`**(*key*)
>> Return whether the given key is cited in any document.
>>
>> **Parameters key** (`str`) – The citation key.
>
> **`purge`**(*docname*)
>> Remove all information related to *docname*.
>>
>> **Parameters docname** (`str`) – The document name.
>
> **`set_enum_count`**(*docname*, *value*)
>> Set enumeration list counter for document *docname* to *value*.

class `sphinxcontrib.bibtex.cache.`**`BibfileCache`**(*mtime=None*, *data=None*)

> Contains information about a parsed .bib file.
>
> **`mtime`**
>> A `float` representing the modification time of the .bib file when it was last parsed.
>
> **`data`**
>> A `pybtex.database.BibliographyData` containing the parsed .bib file.

class sphinxcontrib.bibtex.cache.**BibliographyCache**(*docname=None,   bibfiles=None,
                                                          style=None,   list_='citation',
                                                          enumtype='arabic',   start=1,   la-
                                                          bels=None,   encoding=None,
                                                          curly_bracket_strip=True,   label-
                                                          prefix='', filter_=None*)

> Contains information about a bibliography directive.

> **docname**
> > A str containing the name of the document in which the directive occurs. We need this information during the Sphinx event *env-purge-doc*.

> **bibfiles**
> > A list of strs containing the .bib file names (relative to the top source folder) that contain the references.

> **style**
> > The bibtex style.

> **list_**
> > The list type.

> **enumtype**
> > The sequence type (only used for enumerated lists).

> **start**
> > The first ordinal of the sequence (only used for enumerated lists).

> **labels**
> > Maps citation keys to their final labels.

> **labelprefix**
> > This bibliography's string prefix for pybtex generated labels.

> **filter_**
> > An ast.AST node, containing the parsed filter expression.

1.4 Changes

1.4.1 0.2.8 (7 August 2013)

- Use pybtex-docutils to remove dependency on pybtex.backends.doctree.

1.4.2 0.2.7 (4 August 2013)

- Integrate with coveralls.io, first release with 100% test coverage.

- Minor bug fixes and code improvements.

- Remove ordereddict dependency for Python 2.7 and higher (contributed by Paul Romano, see pull requests #27 and #28).

- New :filter: option for advanced filtering (contributed by d9pouces, see pull requests #30 and #31).

- Refactor documentation of advanced features.

- Document how to create custom pybtex styles (see issues #25, #29, and #34).

- Code is now mostly pep8 compliant.

1.4.3 0.2.6 (2 March 2013)

- For unsorted styles, citation entries are now sorted in the order they are cited, instead of following the order in the bib file, to reflect more closely the way LaTeX handles unsorted styles (addresses issue #15).

- Skip citation label warnings on Sphinx [source] links (issue #17, contributed by Simon Clift).

1.4.4 0.2.5 (18 October 2012)

- Duplicate label detection (issue #14).

- New `:labelprefix:` option to avoid duplicate labels when having multiple bibliographies with a numerical label style (addresses issue #14).

1.4.5 0.2.4 (24 August 2012)

- New options for the bibliography directive for rendering the bibliography as bullet lists or enumerated lists: `:list:`, `:enumtype:`, and `:start:`.

- Minor latex codec fixes.

- Turn exception into warning when a citation cannot be relabeled (fixes issue #2).

- Document LaTeX encoding, and how to turn it off (issue #4).

- Use pybtex labels (fixes issue #6 and issue #7).

- Cache tracked citation keys and labels, and bibliography enumeration counts (fixes issues with citations in repeated Sphinx runs).

- Bibliography ids are now unique across documents (fixes issue that could cause the wrong bibliography to be inserted).

- The plain style is now the default (addresses issue #9).

1.4.6 0.2.3 (30 July 2012)

- Document workaround for Tinkerer (issue #1).

- Use tox for testing.

- Full 2to3 compatibility.

- Document supported versions of Python (2.6, 2.7, 3.1, and 3.2).

1.4.7 0.2.2 (6 July 2012)

- Documentation and manifest fixes.

1.4.8 0.2.1 (19 June 2012)

- First public release.

1.5 License

sphinxcontrib-bibtex is a Sphinx extension for BibTeX style citations

Copyright (c) 2011-2013 by Matthias C. M. Troffaes

All rights reserved.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S