beanbag-docutils

Beanbag, Inc.

Oct 15, 2023

CONTENTS

This is a set of extensions to the Sphinx ReStructuredText-based documentation system used by our products to help generate better documentation

Amongst other enhancements, this package offers:

- A parser for the Beanbag docstring format (a variation on the Google docstring format), which we use for Python and JavaScript documentation
- Enhancements for Sphinx's intersphinx system to provide per-page intersphinx resolution options (useful for pages, such as release notes, that need to link to different versions of the same docs, such as Django or Python)
- Enhancements to ReStructuredText references to let a reference name span lines (useful for long Python/JavaScript module/class names)
- Linking code references to GitHub documentation
- High-DPI image embedding
- A role for HTTP status codes
- Access to document-defined metadata in a structured form when parsing documents

CHAPTER

ONE

INSTALLATION

It's very easy to get going. Just install `beanbag-docutils` like so:

```
$ pip install beanbag-docutils
```

We like to place ours in a `doc-requirements.txt` file in each of our repositories, and use:

```
$ pip install -r doc-requirements.txt
```

CHAPTER

TWO

SPHINX EXTENSIONS

The following extensions are provided:

autodoc_utils

- Enables parsing of the Beanbag docstring format for Python.
- Advanced options for defining excluded and deprecated classes/functions to skip, either in a Sphinx project's `conf.py` or directly in the Python modules.

collect_files

- Allows arbitrary supplemental files to be "collected" along with any of your documentation.

django_utils

- Roles and directives for describing Django-related concepts in your documentation.
- Automatic support for resolving "lazy" localized strings, so they appear correctly.

extlinks

- An improved version of Sphinx's own `sphinx.ext.extlinks` that supports anchors in the string passed to the role (e.g., `:myext:`name#anchor``).

github

- Provides links to the appropriate versions of the tree in GitHub for any source code references, instead of bundling copies of the source code with the documentation.

http_role

- Provides a *http* role for specifying HTTP status codes and linking them to useful documentation.

image_srcsets

- Drop-in support for generating `<img srcset="...">` images using the built-in `image` directive. Appropriate images can be automatically determined or manually specified.

intersphinx_utils

- Allows individual documentation pages to specify which intersphinx mappings should be used if multiple mappings contained the same reference. Useful for having, say, different release notes pages linking to different versions of Python or Django documentation.

json_writer

- An enhanced JSON writer that contains structured data for the docs-wide Table of Contents, and HTML for in-page anchors for docs.

New in version 2.2.

metadata

- Extracts metadata from `meta` directives into the document's metadata, allowing tools or custom doc rendering platforms to access it.

New in version 2.2.

ref_utils

- Allows Python and JavaScript references to span multiple lines, in case of very long class or module paths.

retina_images

- Collects all high-DPI versions of images (e.g., any with a `@2x` in the filename) for the resulting documentation.

CHAPTER

THREE

SEE ALSO

3.1 Sphinx Extensions

beanbag_docutils.sphinx.ext	
beanbag_docutils.sphinx.ext.autodoc_utils	Sphinx extension to add utilities for autodoc.
beanbag_docutils.sphinx.ext.collect_files	Sphinx extension to collect additional files in the build directory.
beanbag_docutils.sphinx.ext.django_utils	Sphinx extension to add useful Django-related functionality.
beanbag_docutils.sphinx.ext.extlinks	Sphinx extension to define external links that support anchors.
beanbag_docutils.sphinx.ext.github	Sphinx extension to link to source code on GitHub.
beanbag_docutils.sphinx.ext.http_role	Sphinx extension to add a *http* role for docs.
beanbag_docutils.sphinx.ext.image_srcsets	Sphinx extension for srcsets in images.
beanbag_docutils.sphinx.ext.intersphinx_utils	Sphinx extension to enhance intersphinx support.
beanbag_docutils.sphinx.ext.json_writer	Sphinx extension offering an enhanced JSON document builder.
beanbag_docutils.sphinx.ext.metadata	Sphinx extension for extracting metadata for the page.
beanbag_docutils.sphinx.ext.ref_utils	Sphinx extension to improve references.
beanbag_docutils.sphinx.ext.retina_images	Sphinx extension for Retina images.

3.1.1 beanbag_docutils.sphinx.ext

3.1.2 beanbag_docutils.sphinx.ext.autodoc_utils

Sphinx extension to add utilities for autodoc.

This enhances autodoc support for Beanbag's docstring format and to allow for excluding content from docs.

Beanbag's Docstrings

By setting `napoleon_beanbag_docstring = True` in `conf.py`, and turning off `napoleon_google_docstring`, Beanbag's docstring format can be used.

This works just like the Google docstring format, but with a few additions:

- A new `Context:` section to describe what happens within the context of a context manager (including the variable).
- New `Model Attributes:` and `Option Args:` sections for defining the attributes on a model or the options in a dictionary when using JavaScript.
- New `Deprecated:`, `Version Added:`, and `Version Changed:` sections for defining version-related information.
- Parsing improvements to allow for wrapping argument types across lines, which is useful when you have long module paths that won't fit on one line.

This requires the `sphinx.ext.napoleon` module to be loaded.

Excluding Content

A module can define top-level `__autodoc_excludes__` or `__deprecated__` lists. These are in the same format as `__all__`, in that they take a list of strings for top-level classes, functions, and variables. Anything listed here will be excluded from any autodoc code.

`__autodoc_excludes__` is particularly handy when documenting an `__init__.py` that imports contents from a submodule and re-exports it in `__all__`. In this case, autodoc would normally output documentation both in `__init__.py` and the submodule, but that can be avoided by setting:

```
__autodoc_excludes = __all__
```

Excludes can also be defined globally, filtered by the type of object the docstring would belong to. See the documentation for autodoc-skip-member for more information. You can configure this in `conf.py` by doing:

```
autodoc_excludes = {
    # Applies to modules, classes, and anything else.
    '*': [
        '__annotations__',
        '__dict__',
        '__doc__',
        '__module__',
        '__weakref__',
    ],
    'class': [
        # Useful for Django models.
        'DoesNotExist',
        'MultipleObjectsReturned',
        'objects',

        # Useful forms.
        'base_fields',
        'media',
    ],
}
```

That's just an example, but a useful one for Django users.

By default, `autodoc_excludes` is set to:

```
autodoc_excludes = {
    # Applies to modules, classes, and anything else.
    '*': [
        '__dict__',
        '__doc__',
        '__module__',
        '__weakref__',
    ],
}
```

If overriding, you can set `'__defaults__': True` to merge your changes in with these defaults.

Changed in version 2.0: Changed from a default of an empty dictionary, and added the `__defaults__` option.

Setup

Changed in version 2.0: Improved setup by enabling other default extensions and options when enabling this extension.

To use this, add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.autodoc_utils',
    ...
]
```

This will automatically enable the `sphinx.ext.autodoc`, `sphinx.ext.intersphinx`, and `sphinx.ext.napoleon` extensions, and enable the following autodoc options:

```
autodoc_member_order = 'bysource'
autoclass_content = 'class'
autodoc_default_options = {
    'members': True,
    'special-members': True,
    'undoc-members': True,
    'show-inheritance': True,
}
```

These default options can be turned off by setting `use_beanbag_autodoc_defaults=False`.

Configuration

autodoc_excludes
: Optional global exclusions to apply, as shown above.

napoleon_beanbag_docstring
: Set whether the Beanbag docstring format should be used.

  This is the default as of `beanbag-docutils` 2.0.

use_beanbag_autodoc_defaults
: Set whether autodoc defaults should be used.

New in version 2.0.

Functions

setup(app)	Set up the Sphinx extension.

Classes

BeanbagDocstring(*args, **kwargs)	Docstring parser for the Beanbag documentation.

class beanbag_docutils.sphinx.ext.autodoc_utils.**BeanbagDocstring**(**args*, ***kwargs*)

Bases: GoogleDocstring

Docstring parser for the Beanbag documentation.

This is based on the Google docstring format used by Napoleon, but with a few additions:

- Support for describing contexts in a context manager (using the `Context:` section, which works like `Returns` or `Yields`).
- `Model Attributes:` and `Option Args:` argument sections.
- Parsing improvements for arguments to allow for wrapping across lines, for long module paths.

partial_typed_arg_start_re = re.compile('\\s*(.+?)\\s*\\(\\s*(.*[^\\s]+)\\s*[^:)]*$')

partial_typed_arg_end_re = re.compile('\\s*(.+?)\\s*\\):$')

extra_returns_sections = [('context', 'Context', {}), ('type', 'Type', {'require_type': True})]

extra_fields_sections = [('keys', 'Keys'), ('model attributes', 'Model Attributes'), ('option args', 'Option Args'), ('tuple', 'Tuple')]

extra_version_info_sections = [('deprecated', 'deprecated'), ('version added', 'versionadded'), ('version changed', 'versionchanged')]

MAX_PARTIAL_TYPED_ARG_LINES = 3

COMMA_RE = re.compile(',\\s*')

__init__(**args*, ***kwargs*)

Initialize the parser.

Parameters

- ***args** (tuple) – Positional arguments for the parent.
- ****kwargs** (dict) – Keyword arguments for the parent.

register_returns_section(*keyword*, *label*, *options={}*)

Register a Returns-like section with the given keyword and label.

Parameters

- **keyword** (unicode) – The keyword used in the docs.

- **label** (unicode) – The label outputted in the section.
- **options** (dict, *optional*) – Options for the registration.

  This accepts:

  Keys
  require_type (bool, *optional*) – Whether the type is required, and assumed to be the first line.

  New in version 2.0.

register_fields_section(*keyword*, *label*)

Register a fields section with the given keyword and label.

Parameters

- **keyword** (unicode) – The keyword used in the docs.
- **label** (unicode) – The label outputted in the section.

register_version_info_section(*keyword*, *admonition*)

Register a version section with the given keyword and admonition.

Parameters

- **keyword** (unicode) – The keyword used in the docs.
- **admonition** (unicode) – The admonition to use for the section.

peek_lines(*num_lines=1*)

Return the specified number of lines without consuming them.

New in version 1.9.

Parameters
num_lines (int, *optional*) – The number of lines to return.

Returns
The resulting lines.

Return type
list of str

consume_lines(*num_lines*)

Consume the specified number of lines.

This will ensure that these lines are not processed any further.

New in version 1.9.

Parameters
num_lines (int, *optional*) – The number of lines to consume.

queue_line(*line*)

Queue a line for processing.

This will place the line at the beginning of the processing queue.

New in version 1.9.

Parameters
line (str) – The line to queue.

make_type_reference(*type_str*)

Create references to types in a type string.

This will parse the string, separating out a type from zero or more suffixes (like `, optional`).

The type will be further parsed into a space-separated tokens. Each of those will be set as a reference, allowing Sphinx to try to link it. The exceptions are the words "of" and "or", which we use to list optional types.

The suffixes are each formatted with emphasis.

New in version 2.0.

Parameters
type_str (`unicode`) – The string to parse and create references from.

Returns
The new string.

Return type
`unicode`

beanbag_docutils.sphinx.ext.autodoc_utils.**setup**(*app*)

Set up the Sphinx extension.

This sets up the configuration and event handlers needed for enhancing auto-doc support.

Parameters
app (`sphinx.application.Sphinx`) – The Sphinx application to register configuration and listen to events on.

3.1.3 beanbag_docutils.sphinx.ext.collect_files

Sphinx extension to collect additional files in the build directory.

This is used to copy files (indicated by glob patterns) from the source directory into the destination build directory. Each destination file will be in the same relative place in the tree.

This is useful when you have non-ReST/image files that you want part of your built set of files, perhaps containing metadata or packaging that you want to ship along with the documentation.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.collect_files',
    ...
]
```

And then configure `collect_file_patterns` to be a list of filenames/glob patterns.

Configuration

collect_file_patterns
List of filenames or glob patterns to include from the source directory into the build directory.

Functions

collect_files(app, env)	Collect configured files and put them into the build directory.
setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.collect_files.**collect_files**(*app*, *env*)

Collect configured files and put them into the build directory.

Parameters

- **app** (sphinx.application.Sphinx) – The Sphinx application to register roles and configuration on.
- **env** (sphinx.environment.BuildEnvironment, *unused*) – The build environment for the generated docs.

beanbag_docutils.sphinx.ext.collect_files.**setup**(*app*)

Set up the Sphinx extension.

This listens for the events needed to collect files.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application to listen to events on.

Returns
Information about the extension.

Return type
dict

3.1.4 beanbag_docutils.sphinx.ext.django_utils

Sphinx extension to add useful Django-related functionality.

This adds some improvements when working with Django-based classes in autodocs, and when referencing Django documentation.

Localized strings using ugettext_lazy() will be turned into actual strings for the documentation, which is useful for forms and models.

Roles are also added for common cross-references.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.django_utils',
    ...
]
```

Roles

:setting:

Creates and links to references for Django settings (those that live in `django.conf.settings`).

For example:

```
.. setting:: MY_SETTING

Settings go here.

And then to reference it: :setting:`MY_SETTING`.
```

Functions

setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.django_utils.**setup**(*app*)

Set up the Sphinx extension.

This registers cross-references and fixes up the lazily-localized strings for display.

Parameters

app (sphinx.application.Sphinx) – The Sphinx application to listen to events on.

3.1.5 beanbag_docutils.sphinx.ext.extlinks

Sphinx extension to define external links that support anchors.

Sphinx comes bundled with a `sphinx.ext.extlinks` extension, which allows a `conf.py` to define roles for external links. These don't support anchors, however, making it impossible to link to properly link in some cases.

This is a wrapper around that module that looks for anchors and appends them to the resulting URL.

Setup

This extension works identically to `sphinx.ext.extlinks` and contains the same configuration. To use it, configure external links the way you would for that extension, but add ours instead to `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.extlinks',
    ...
]
```

Functions

setup(app)	Set up the Sphinx extension.
setup_link_roles(app)	Register roles for each external link that's been defined.

Classes

ExternalLink(base_url)	Wraps a URL and formats references to allow for using anchors.

class beanbag_docutils.sphinx.ext.extlinks.**ExternalLink**(*base_url*)

Bases: object

Wraps a URL and formats references to allow for using anchors.

This will work like a string, from the point of view of `sphinx.ext.extlinks`. It takes the URL for the external link and intercepts any string formatting, pulling out the anchor and appending it to the final result.

__init__(*base_url*)

Initialize the class.

Parameters
base_url (unicode) – The URL to wrap. This must contain a %s.

__mod__(*ref*)

Return a URL based on the stored string format and the reference.

Parameters
ref (unicode) – The reference to place into the URL. This may contain an anchor starting with #.

Returns
The formatted URL.

Return type
unicode

__add__(*s*)

Return the concatenated string for the base URL and another string.

Parameters
s (unicode) – A string to concatenate onto this base URL.

Returns
The concatenated string.

Return type
unicode

beanbag_docutils.sphinx.ext.extlinks.**setup_link_roles**(*app*)

Register roles for each external link that's been defined.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application.

beanbag_docutils.sphinx.ext.extlinks.**setup**(*app*)

Set up the Sphinx extension.

This registers the configuration for external links and adds the roles for each when the builder initializes.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application.

Returns
Information about the extension. This is in the same format as what sphinx.ext.extlinks.setup() returns.

Return type
dict

3.1.6 beanbag_docutils.sphinx.ext.github

Sphinx extension to link to source code on GitHub.

This links to source code for modules, classes, etc. to the correct line on GitHub. This prevents having to bundle the source code along with the documentation, and better ties everything together.

Setup

To use this, you'll need to import the module and define your own linkcode_resolve() in your conf.py:

```
from beanbag_docutils.sphinx.ext.github import github_linkcode_resolve

extensions = [
    ...
    'sphinx.ext.linkcode',
    ...
]

def linkcode_resolve(domain, info):
    return github_linkcode_resolve(
        domain=domain,
        info=info,
        allowed_module_names=['mymodule'],
        github_org_id='myorg',
        github_repo_id='myrepo',
        branch='master',
        source_prefix='src/')
```

`source_prefix` and `allowed_module_names` are optional. See the docs for *github_linkcode_resolve()* for more information.

Functions

clear_github_linkcode_caches()	Clear the internal caches for GitHub code linking.
github_linkcode_resolve(domain, info, ...[, ...])	Return a link to the source on GitHub for the given autodoc info.

beanbag_docutils.sphinx.ext.github.**github_linkcode_resolve**(*domain*, *info*, *github_org_id*, *github_repo_id*, *branch*, *source_prefix=''*, *allowed_module_names=[]*, ***, *github_url='https://github.com'*)

Return a link to the source on GitHub for the given autodoc info.

This takes some basic information on the GitHub project, branch, and what modules are considered acceptable, and generates a link to the approprite line on the GitHub repository browser for the class, function, variable, or other object.

Changed in version 2.1: Added `github_host` and `github_scheme` arguments.

Parameters

- **domain** (`unicode`) – The autodoc domain being processed. This only accepts "py", and comes from the original `linkcode_resolve()` call.
- **info** (`dict`) – Information on the item being linked to. This comes from the original `linkcode_resolve()` call.
- **github_org_id** (`unicode`) – The GitHub organization ID.
- **github_repo_id** (`unicode`) – The GitHub repository ID.
- **branch** (`unicode`) – The branch used as a merge base to find the appropriate commit to link to. Callers may want to compute this off of the version number of the project, or some other information.
- **source_prefix** (`unicode`, *optional*) – A prefix for any linked filename, in case the module is not at the top of the source tree.
- **allowed_module_names** (`list` of `unicode`, *optional*) – The list of top-level module names considered valid for links. If provided, links will only be generated if residing somewhere in one of the provided module names.
- **github_url** (`str`, *optional*) – The URL to the base of the GitHub server.

  New in version 2.1.

Returns

The link to the source. This will be `None` if a URL couldn't be calculated.

Return type

`str`

beanbag_docutils.sphinx.ext.github.**clear_github_linkcode_caches**()

Clear the internal caches for GitHub code linking.

This is primarily intended for unit tests.

New in version 2.0.

3.1.7 beanbag_docutils.sphinx.ext.http_role

Sphinx extension to add a *http* role for docs.

This extension makes it easy to reference HTTP status codes.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.http_role',
    ...
]
```

Roles

.. http-status-codes-format::

Specifies a new format to use by default for any *http* roles. This should include `%(code)s` for the numeric code and `%(name)s` for the name of the HTTP status code.

Call this again without an argument to use the default format.

:http:

References an HTTP status code, expanding to the full status name and linking to documentation on the status in the process.

Configuration

http_status_codes_format

The format string used for the titles for HTTP status codes. This defaults to `HTTP %(code)s %(format)s` and can be temporarily overridden using *http-status-codes-format*.

http_status_codes_url

The location of the docs for the status codes. This expects a string with a `%s`, which will be replaced by the numeric HTTP status code.

Functions

http_role(role, rawtext, text, linenum, inliner)	Implementation of the *http* role.
setup(app)	Set up the Sphinx extension.

Classes

SetStatusCodesFormatDirective(name, ...)	Specifies the format to use for the `:http:` role's text.

class beanbag_docutils.sphinx.ext.http_role.**SetStatusCodesFormatDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: `Directive`

Specifies the format to use for the `:http:` role's text.

required_arguments = 0

Number of required directive arguments.

optional_arguments = 1

Number of optional arguments after the required arguments.

final_argument_whitespace = True

May the final argument contain whitespace?

run()

Run the directive.

Returns
An empty list, always.

Return type
`list`

beanbag_docutils.sphinx.ext.http_role.**http_role**(*role*, *rawtext*, *text*, *linenum*, *inliner*, *options={}*, *content=[]*)

Implementation of the *http* role.

This is responsible for converting a HTTP status code to link pointing to the status documentation, with the full text for the status name.

Parameters

- **rawtext** (`unicode`) – The raw text for the entire role.
- **text** (`unicode`) – The interpreted text content.
- **linenum** (`int`) – The current line number.
- **inliner** (`docutils.parsers.rst.states.Inliner`) – The inliner used for error reporting and document tree access.
- **options** (`dict`) – Options passed for the role. This is unused.
- **content** (`list` of `unicode`) – The list of strings containing content for the role directive. This is unused.

Returns

The result of the role. It's a tuple containing two items:

1) A single-item list with the resulting node.

2) A single-item list with the error message (if any).

Return type
tuple

beanbag_docutils.sphinx.ext.http_role.**setup**(*app*)

Set up the Sphinx extension.

This registers the *http* role, *http-status-codes-format* directive, and the configuration settings for specifying the format and URL for linking to HTTP status codes.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application to register roles and configuration on.

3.1.8 beanbag_docutils.sphinx.ext.image_srcsets

Sphinx extension for srcsets in images.

New in version 2.1.

This extension adds a `sources` option to the standard image directives, enabling responsive image support via srcsets.

These are specified a bit differently from `<img srcset="...">` values. The descriptor goes first, and a comma between entries is optional (a blank line can be used instead). For example:

```
.. image:: path/to/file.png
   :sources: 2x path/to/file@2x.png
             3x path/to/file@3x.png
             100w path/to/file@100w.png
             200h path/to/file@200h.png
```

If `sources` is not explicitly provided, but files with those standard `@`-based suffixes exist alongside the referenced main image, they'll automatically be used to define the srcsets of the image. The `1x` entry is also automatically inserted based on the main image.

If relying on the default of scanning for srcset images, this becomes a zero-configuration, drop-in solution for all Sphinx documentation codebases.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.image_srcsets',
    ...
]
```

Functions

collect_pages(app)	Collect srcset-specified images for use in HTML pages.
collect_srcsets(app, doctree)	Collect all images referenced by image nodes or scanned in directories.
setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.image_srcsets.**collect_srcsets**(*app*, *doctree*)

Collect all images referenced by image nodes or scanned in directories.

This will collect any explicit values defined via our `sources` option for image directives. If `sources` is not specified, but there are files in the directory with `@2x`, `@3x`, `@100w` `@100h`, etc. descriptors, those will be collected instead and associated with the image.

Parameters

- **app** (`sphinx.application.Sphinx`) – The Sphinx application being run.
- **doctree** (`docutils.nodes.document`) – The document tree being processed.

beanbag_docutils.sphinx.ext.image_srcsets.**collect_pages**(*app*)

Collect srcset-specified images for use in HTML pages.

This will go through the images referenced in a document for an HTML page and add any images found in srcsets to the list of images to collect for the page.

Parameters
app (`sphinx.application.Sphinx`) – The Sphinx application to register roles and configuration on.

Returns
An empty list (indicating no additional HTML pages are collected).

Return type
`list`

beanbag_docutils.sphinx.ext.image_srcsets.**setup**(*app*)

Set up the Sphinx extension.

This listens for the events needed to collect and bundle images for srcsets, and update the resulting HTML to specify them.

Parameters
app (`sphinx.application.Sphinx`) – The Sphinx application being run.

3.1.9 beanbag_docutils.sphinx.ext.intersphinx_utils

Sphinx extension to enhance intersphinx support.

This fixes some reference issues with `option` (see https://github.com/sphinx-doc/sphinx/pull/3769 for the equivalent upstream fix).

It also introduces a `.. default-intersphinx::` directive that allows for specifying one or more intersphinx set prefixes that should be tried if a reference could not be found. For example:

```
.. default-intersphinx:: myapp1.5 python

:ref:`some-reference`
```

This does affect the process by which missing references are located. If an unprefixed reference is used, it will only match if the prefix is in the list above, which differs from the default behavior of looking through all intersphinx mappings.

Setup

This extension must be added to `exetnsions` in `conf.py` after the `sphinx.ext.intersphinx` extension is added. For example:

```
extensions = [
    ...
    'sphinx.ext.intersphinx',
    'beanbag_docutils.sphinx.ext.intersphinx',
    ...
]
```

Functions

setup(app)	Set up the Sphinx extension.

Classes

DefaultIntersphinx(name, arguments, options, ...)	Specifies one or more default intersphinx sets to use.

class beanbag_docutils.sphinx.ext.intersphinx_utils.**DefaultIntersphinx**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: `Directive`

Specifies one or more default intersphinx sets to use.

required_arguments = 1

Number of required directive arguments.

optional_arguments = 100

Number of optional arguments after the required arguments.

SPLIT_RE = re.compile(',\\s*')

run()

Run the directive.

Returns
An empty list, always.

Return type
list

beanbag_docutils.sphinx.ext.intersphinx_utils.**setup**(*app*)

Set up the Sphinx extension.

This listens for the events needed to handle missing references, and registers directives.

Parameters

app (sphinx.application.Sphinx) – The Sphinx application building the docs.

3.1.10 beanbag_docutils.sphinx.ext.json_writer

Sphinx extension offering an enhanced JSON document builder.

When enabled, the JSON builder provided by sphinxcontrib-serializinghtml will be augmented with:

1. A `toc` key in `globalcontext.json` containing a structured Table of Contents for the site.
2. An `anchors_html` in the page's context containing HTML-rendered anchors for the page, without the page title being included.

Setup

To use this, add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.json_writer',
    ...
]
```

This will automatically enable the `sphinxcontrib.serializinghtml` extension and set the appropriate overrides.

You can then build the JSON documentation using the `json` builder.

Table of Contents Structure

The `toc` key in `globalcontext.json` contains a list of dictionaries containing:

Keys

- **docname** (str) – The name of the document.
- **title** (str) – The document title.
- **items** (list, *optional*) – The list of child documents under this page in the tree. This uses this same structure.

This will start with the `index.rst` at the top of the project, if it exists. If it does not, it will fall back to the configured Sphinx root document (configured by `root_doc`).

Anchors Structure

Anchors are stored in a `.fjson` page's `anchors_html` key. This will be pre-rendered HTML, following the form of the normal `toc` key. For example:

```
<ul>
 <li><a class="reference internal" href="#section1">Section 1</a></li>
 <li><a class="reference internal" href="#section2">Section 2</a><ul>
  <li><a class="reference internal" href="#section2.1">Section 2.1</a></li>
  <li><a class="reference internal" href="#section2.2">Section 2.2</a></li>
  ...
 </ul></li>
 ...
</ul>
```

Functions

setup(app)	Set up the Sphinx extension.

Classes

JSONBuilder(app[, env])	Sphinx builder for JSON files.

class beanbag_docutils.sphinx.ext.json_writer.**JSONBuilder**(*app: Sphinx*, *env: Optional[BuildEnvironment] = None*)

Bases: `JSONHTMLBuilder`

Sphinx builder for JSON files.

This specializes `sphinxcontrib.serializinghtml.JSONHTMLBuilder`, adding additional state for a structured Table of Contents, available in `glboalcontext.json`.

New in version 2.2.

handle_finish()

Handle finishing the build for all the docs.

beanbag_docutils.sphinx.ext.json_writer.**setup**(*app*)

Set up the Sphinx extension.

This sets up the configuration and event handlers needed for the JSON writer extension.

New in version 2.2.

Parameters

app (`sphinx.application.Sphinx`) – The Sphinx application to register configuration and listen to events on.

3.1.11 beanbag_docutils.sphinx.ext.metadata

Sphinx extension for extracting metadata for the page.

New in version 2.2.

This extension extracts any `meta` information (useful for content like page descriptions for social media) and extracts their information into the document's top-level metadata.

This metadata can then be used when building sites or tools that process the page's compiled documentation (when not simply relying on the generated HTML).

The metadata will be available in the document's `meta` information, keyed off by the name used in the `meta` directive, the content, and any other attributes provided.

If multiple pieces of metadata exist for the same key, the information will be assembled into a list under that key. Otherwise, the information will be assigned directly to the key.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.metadata',
    ...
]
```

Functions

setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.metadata.**setup**(*app*)

Set up the Sphinx extension.

This listens for the event needed to process the document tree for metadata extraction.

New in version 2.2.

Parameters

app (`sphinx.application.Sphinx`) – The Sphinx application being run.

3.1.12 beanbag_docutils.sphinx.ext.ref_utils

Sphinx extension to improve references.

This enhances references, allowing both Python and JavaScript references to break paths (like `foo.bar.MyClass`) across multiple lines.

Setup

To use this, add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.ref_utils',
    ...
]
```

Functions

setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.ref_utils.**setup**(*app*)

Set up the Sphinx extension.

This alters some behavior for the Python domain, allowing newlines in references.

Parameters

app (sphinx.application.Sphinx) – The Sphinx application to extend.

3.1.13 beanbag_docutils.sphinx.ext.retina_images

Sphinx extension for Retina images.

Deprecated since version 2.1: We recommend using *beanbag_docutils.sphinx.ext.image_srcsets* instead. This extension will be removed in a future release.

This extension goes through all the images Sphinx will provide in _images and checks if Retina versions are available. If there are any, they will be copied as well.

Setup

To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.retina_images',
    ...
]
```

Configuration

retina_suffixes:
A list of suffix identifiers for Retina images. Each of these go after the filename and before the extension. This defaults to `['@2x', '@3x']`.

Functions

add_high_dpi_images(app, env)	Add high-DPI images to the list of bundled images.
collect_pages(app)	Collect high-DPI images for use in HTML pages.
setup(app)	Set up the Sphinx extension.

beanbag_docutils.sphinx.ext.retina_images.**add_high_dpi_images**(*app*, *env*)

Add high-DPI images to the list of bundled images.

Any image that has a "@2x" version will be included in the output directory for the docs.

Parameters

- **app** (sphinx.application.Sphinx) – The Sphinx application to register roles and configuration on.
- **env** (sphinx.environment.BuildEnvironment) – The build environment for the generated docs.

beanbag_docutils.sphinx.ext.retina_images.**collect_pages**(*app*)

Collect high-DPI images for use in HTML pages.

This will go through the images referenced in a document for an HTML page and add any high-DPI versions previously found in *add_high_dpi_images()* to the list of images to collect for the page.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application to register roles and configuration on.

Returns
An empty list (indicating no additional HTML pages are collected).

Return type
list

beanbag_docutils.sphinx.ext.retina_images.**setup**(*app*)

Set up the Sphinx extension.

This listens for the events needed to collect and bundle high-DPI images.

Parameters
app (sphinx.application.Sphinx) – The Sphinx application to listen to events on.

3.2 Release Notes

3.2.1 beanbag-docutils 2.3

Release date: October 15, 2023

Packaging / Compatibility

- Added support for Sphinx 7.2.

beanbag_docutils.sphinx.ext.image_srcsets

- Images with widths/heights now have `width` and `height` attributes set.

  Sphinx turns widths and heights into `style="width: ...; height: ..."` attributes, but does not set the dedicated `width` and `height` attributes. These attributes are helpful for the browser when computing layout and aspect ratios, and are not deprecated or strictly interchangeable with the CSS styles.

  This extension now sets these attributes and strips them out of any CSS styling.

Contributors

- Christian Hammond
- David Trowbridge
- Michelle Aubin

3.2.2 beanbag-docutils 2.2

Release date: June 19, 2023

Packaging / Compatibility

- Added support for Python 3.12.
- Added support for Sphinx 6 and 7.

New Extensions

- Added a new *metadata* Sphinx extension for extracting additional page metadata.

  This extension extracts any `meta` information (useful for content like page descriptions for social media) and extracts their information into the document's top-level metadata.

  This metadata can then be used when building sites or tools that process the page's compiled documentation (when not simply relying on the generated HTML).

  The metadata will be available in the document's `meta` information, keyed off by the name used in the `meta` directive, the content, and any other attributes provided.

  If multiple pieces of metadata exist for the same key, the information will be assembled into a list under that key. Otherwise, the information will be assigned directly to the key.

- Added a new *json_writer* Sphinx extension for enhanced JSON writing.

  This augments the JSON writer typically found in the sphinxcontrib-serializinghtml package to include two new pieces of information:

  1. A structured representation of the Table of Contents for the whole doc bundle.
  2. Rendered HTML for the anchors in the page (similar to the "toc" available in the page, but without the page's own header).

  These can be used by consumers to pull out this structured information without having to parse the docsets themselves, helping provide a dynamic Table of Contents.

Contributors

- Christian Hammond
- David Trowbridge
- Michelle Aubin

3.2.3 beanbag-docutils 2.1.1

Release date: February 28, 2023

beanbag_docutils.sphinx.ext.image_srcsets

- Fixed automatically scanning and collecting multi-DPI images for the generated documentation.

Contributors

- Christian Hammond
- David Trowbridge

3.2.4 beanbag-docutils 2.1

Release date: February 4, 2023

Packaging / Compatibility

- Added support for Sphinx 6.
- Updated the Sphinx dependency to require version 4 or higher.
- Removed a dependency on `six`.

beanbag_docutils.sphinx.ext.autodoc_utils

- Fixed performing incremental builds.

  Due to a bug, all builds were being treated as full builds.

- `__annotations__` are now excluded from generated docs by default.

beanbag_docutils.sphinx.ext.github

- Added support for custom GitHub URLs.

  A GitHub URL (useful for GitHub Enterprise) can now be set by passing `github_url=` to *`github_linkcode_resolve()`*.

- Fixed generating links to GitHub when failing to resolve tracking branches.

  These links were appearing broken before. Now, if a tracking branch can't be resolved, the links will point to the file on GitHub without a stable SHA.

beanbag_docutils.sphinx.ext.image_srcsets

- Added the new *`image_srcsets`* Sphinx extension.

  This extension enables multi-DPI/responsive images to be used in generated documentation.

  It's intended to replace `image` and can be used like:

```
.. image:: path/to/file.png
   :sources: 2x path/to/file@2x.png
             3x path/to/file@3x.png
             100w path/to/file@100w.svg
             200h path/to/file@200h.svg
```

Contributors

- Christian Hammond
- David Trowbridge
- Michelle Aubin

3.2.5 beanbag-docutils 2.0

Release date: August 17, 2022

Compatibility

- Dropped support for Python 2.7 and added Python 3.11.

  Beanbag Docutils now supports 3.6 through 3.11.

beanbag_docutils.sphinx.ext.autodoc_utils

- This module now sets a handful of default Sphinx configuration settings:

```
extensions = [
    'sphinx.ext.autodoc,',
    'sphinx.ext.intersphinx',
    'sphinx.ext.napoleon',
]

autodoc_member_order = 'bysource'
autoclass_content = 'class'
autodoc_default_options = {
    'members': True,
    'special-members': True,
    'undoc-members': True,
    'show-inheritance': True,
}
```

  To disable these defaults, set:

```
use_beanbag_autodoc_defaults = False
```

- Added support for a `Keys` section.

  This is used to document dictionaries and other similar content. It works as a section nested within a paragraph and resembles `Args` or `Attributes`.

  For example:

```
Here is a description of some dictionary:

Keys:
    key1 (str):
       Description of the key.

    key2 (collections.OrderedDict, optional):
       And a description of another key.
```

- Added support for a `Tuple` section.

  This is used to document tuples. It also works as a section nested within a paragraph, and resembles `Args` or `Attributes`.

  For example:

```
Here is a description of some tuple:

Tuple:
    1 (int):
```

(continues on next page)

(continued from previous page)

```
      Description of the first item.

   2 (bool):
      And a description of the second.

   3 (django.db.models.Model):
      And finally the third.
```

- Added support for a `Type` section.

  This is used to document the type of an attribute or property. It works similarly to a `Returns` section, with the description being optional.

  For example:

```
This attribute does a thing.

Type:
    path.to.MyObject
```

- Fixed including list items in `Version Added`, `Version Changed`, and `Deprecated` sections.

beanbag_docutils.sphinx.ext.github

- Improved several aspects of linking to Python code on GitHub.

  This can now handle linking to attributes, variables, and dynamically-generated objects like named tuples.

  The entire way of linking has been redone, now taking advantage of the Python AST to determine the correct line numbers to content, rather than an older approach that only supported typical classes, functions, and methods.

- Fixed regressions in linking to code on GitHub when building on Python 3.

Contributors

- Christian Hammond
- David Trowbridge

3.2.6 beanbag-docutils 1.9

Release date: July 26, 2022

Compatibility Improvements

- Added compatibility with Sphinx 5.x.
- Updated the Sphinx dependency to not specify a range. It's now completely in the hands of the consuming codebase.

Contributors

- Christian Hammond

3.2.7 beanbag-docutils 1.8.1

Release date: August 1, 2021

Compatibility Improvements

- Added compatibility with Sphinx 4.x+.

Bug Fixes

- Fixed the *beanbag_docutils.sphinx.ext.extlinks* module on Sphinx 4+.

Contributors

- Christian Hammond

3.2.8 beanbag-docutils 1.8

Release date: February 27, 2021

Compatibility Improvements

- Added compatibility for Django 3.x.
- Added compatibility with Sphinx 3.x+.

  The correct version is now used depending on the version of Python.

Project Changes

- Universal wheels are now built for Python 2 and 3 compatibility.
- Added documentation for beanbag-docutils.
- Added a unit test suite.

Bug Fixes

- Fixed a syntax error with *http* with HTTP 418 on Python 3.

Contributors

- Christian Hammond

3.2.9 beanbag-docutils 1.7.1

Release date: August 5, 2020

Compatibility Changes

- Capped the Sphinx version to >=1.7.1, <2.0.

Bug Fixes

- Fixed a crash in *beanbag_docutils.sphinx.ext.autodoc_utils* on newer versions of Sphinx.

Contributors

- Christian Hammond

3.2.10 beanbag-docutils 1.7

Release date: June 14, 2018

New Features

- Added compatibility with Python 3.

  This package now supports Python 2.7 and 3.4 through 3.6.

- Added new documentation sections for the Beanbag documentation format.

  This is enabled when using *beanbag_docutils.sphinx.ext.autodoc_utils* with `napoleon_beanbag_docstring = True`.

  This adds:

  `Deprecated:`
  : A version in which something is deprecated, and an optional description. This is similar to the ReST `deprecated` directive.

  `Version Added:`
  : A version in which something is added, and an optional description. This is similar to the ReST `versionadded` directive.

  `Version Changed:`
  : A version in which something is changed, and an optional description. This is similar to the ReST `versionchanged` directive.

- Added a Sphinx extension (*beanbag_docutils.sphinx.ext.ref_utils*) for improving code references.

  This currently enhances references, allowing both Python and JavaScript references to break paths (like `foo.bar.MyClass`) across multiple lines.

  To install this extension, add the following to your `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.ref_utils',
    ...
]
```

Bug Fixes

- Fixed handling of multi-line argument types when using the Beanbag documentation format support.

  This addresses a case where spaces after a type (such as `int, optional`) would be lost.

Contributors

- Christian Hammond

3.2.11 beanbag-docutils 1.6

Release date: November 16, 2017

New Features

- Added custom titles for HTTP error codes in *http*.

Contributors

- Christian Hammond

3.2.12 beanbag-docutils 1.5

Release date: November 10, 2017

New Features

- Added configuration options for available Retina suffixes like `@2x` and `@3x` in *beanbag_docutils.sphinx.ext.retina_images*.

Bug Fixes

- Fixed handling multiple images with the same filenames in *beanbag_docutils.sphinx.ext.retina_images*.

Contributors

- Christian Hammond

3.2.13 beanbag-docutils 1.4.1

Release date: October 4, 2017

New Features

- Added new HTTP status codes for the *http* role.

  This includes:

  - HTTP 208 Already Reported
  - HTTP 308 Permanent Redirect
  - HTTP 428 Precondition Required
  - HTTP 429 Too Many Requests
  - HTTP 431 Request Header Fields Too Large
  - HTTP 444 No Response
  - HTTP 451 Unavailable For Legal Reasons
  - HTTP 499 Client Closed Request
  - HTTP 508 Loop Detected
  - HTTP 511 Network Authentication Required
  - HTTP 598 Network Read Timeout Error
  - HTTP 599 Network Connect Timeout Error

Contributors

- Christian Hammond

3.2.14 beanbag-docutils 1.4

Release date: May 25, 2017

New Features

- Added a parser for the Beanbag documentation format.

  This is part of *beanbag_docutils.sphinx.ext.autodoc_utils*, and is enabled when setting `napoleon_beanbag_docstring = True` in `conf.py`. It replaces the Google and Numpy docstring formats, but requires `sphinx.ext.napoleon`.

  This currently supports:

  `Context:`
  : Describes what happens within the context of a context manager.

Model Attributes:
: Describes the attributes on a Backbone.js model.

Option Args:
: Describes options available to a function in JavaScript.

This parser also allows types (used in `Args` and other sections) to be wrapped across lines.

Contributors

- Christian Hammond

3.2.15 beanbag-docutils 1.3

Release date: May 23, 2017

New Features

- Added *beanbag_docutils.sphinx.ext.intersphinx_utils*, which adds new features for intersphinx lookups.

  This adds a `default-intersphinx` directive that allows for specifying one or more intersphinx set prefixes that should be tried if a reference could not be found. For example:

```
.. default-intersphinx:: myapp1.5, python

:ref:`some-reference`
```

  This does affect the process by which missing references are located. If an unprefixed reference is used, it will only match if the prefix is in the list above, which differs from the default behavior of looking through all intersphinx mappings.

  This also works around an issue with `option` (see https://github.com/sphinx-doc/sphinx/pull/3769 for the equivalent upstream fix).

Contributors

- Christian Hammond

3.2.16 beanbag-docutils 1.2

Release date: January 21, 2017

New Features

- Added *beanbag_docutils.sphinx.ext.collect_files* for collecting additional files in a build.

  This is used to copy files (indicated by glob patterns) from the source directory into the destination build directory. Each destination file will be in the same relative place in the tree.

  This is useful when you have non-ReST/image files that you want part of your built set of files, perhaps containing metadata or packaging that you want to ship along with the documentation.

  To use this, you just need to add the extension in `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.collect_files',
    ...
]
```

  And then configure `collect_file_patterns` to be a list of filenames/glob patterns, like:

```
collect_file_patterns = ['metadata.json', '*.pdf']
```

Contributors

- Christian Hammond

3.2.17 beanbag-docutils 1.1

Release date: November 18, 2016

New Features

- Added *beanbag_docutils.sphinx.ext.extlinks* for defining anchor-aware external links.

  Sphinx ships with a `sphinx.ext.extlinks` extension, which allows for defining roles that map to external links. These take a URL that work as a format string, containing a `%s`. The reference is then passed into that, and the result is a URL embedded in the page.

  This is very useful, but has a flaw in that anchors do not work with these links. If the format string has any content after the `%s`, then that content will appear after the anchor. Support for this is being tracked upstream by Sphinx, but as of yet there isn't any indication of a plan to implement anchor support.

  This extension adds that anchor support, building upon the logic in Sphinx's extension and adding a shim that handles anchor-safe string formatting. It's a drop-in replacement that can be used without changing any configuration, meaning we can remove it down the road without any real work if Sphinx ends up fixing this upstream.

Contributors

- Christian Hammond

3.2.18 beanbag-docutils 1.0

Release date: July 19, 2016

Initial release, featuring:

autodoc_utils

Enhances autodoc support to allow for excluding content from docs.

A module can define top-level `__autodoc_excludes__` or `__deprecated__` lists. These are in the same format as `__all__`, in that they take a list of strings for top-level classes, functions, and variables. Anything listed here will be excluded from any autodoc code.

`__autodoc_excludes__` is particularly handy when documenting an `__init__.py` that imports contents from a submodule and re-exports it in `__all__`. In this case, autodoc would normally output documentation both in `__init__.py` and the submodule, but that can be avoided by setting:

```
__autodoc_excludes = __all__
```

Excludes can also be defined globally, filtered by the type of object the docstring would belong to. See the documentation for autodoc-skip-member for more information. You can configure this in `conf.py` by doing:

```
autodoc_excludes = {
    # Applies to modules, classes, and anything else.
    '*': [
        '__dict__',
        '__doc__',
        '__module__',
        '__weakref__',
    ],
    'class': [
        # Useful for Django models.
        'DoesNotExist',
        'MultipleObjectsReturned',
        'objects',

        # Useful forms.
        'base_fields',
        'media',
    ],
}
```

That's just an example, but a useful one for Django users.

To install this extension, add the following to your `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.autodoc_utils',
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

django_utils

Adds some improvements when working with Django-based classes in autodocs, and when referencing Django documentation.

First, this will take localized strings using `ugettext_lazy()` and turn them into actual strings, which is useful for forms and models.

Second, this adds linking for setting-based documentation, allowing custom settings (from `django.conf.settings`) to be documented and referenced, like so:

```
.. setting:: MY_SETTING

Settings go here.

And then to reference it: :setting:`MY_SETTING`.
```

To install this extension, add the following to your `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.django_utils',
    ...
]
```

github_linkcode

Links source code for modules, classes, etc. to the correct line on GitHub. This prevents having to bundle the source code along with the documentation, and better ties everything together.

To use this, simply add the following to `conf.py`:

```
from beanbag_docutils.sphinx.ext.github import github_linkcode_resolve

extensions = [
    ...
    'sphinx.ext.linkcode',
    ...
]

linkcode_resolve = github_linkcode_resolve
```

http_role

Provides references for HTTP codes, linking to the matching docs on Wikipedia.

To create a link, simply do:

```
This is :http:`404`.
```

If you want to use a different URL, you can add the following to `conf.py`:

```
http_status_codes_url = 'http://mydomain/http/%s'
```

Where `%s` will be replaced by the HTTP code.

To install this extension, add the following to your `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.http_role',
    ...
]
```

retina_images

Copies all Retina versions of images (any with a `@2x` filename) into the build directory for the docs. This works well with scripts like retina.js.

To install this extension, add the following to your `conf.py`:

```
extensions = [
    ...
    'beanbag_docutils.sphinx.ext.retina_images',
    ...
]
```

Contributors

- Christian Hammond
- Other Projects:
  - Review Board - Our extensible open source code review product.
  - RBCommons - Our Review Board SaaS.
  - Django Evolution - Advanced schema migration for Django, compatible with Django's migrations.
  - Djblets - A set of utilities and infrastructure for Django-based projects.
  - kgb - Function spies for Python unit tests.
  - RBTools - Command line tools for Review Board and RBCommons.

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

J

M

O

P

Q

R

S