sphinx-config-schema

Release 0.1.dev1+g350e93f05

Common Partial-Wave Analysis

Nov 12, 2026

Contents

1 Installation **1**

2 Usage **1**
2.1 Referencing options 4
2.2 Other output formats 5

3 Directive reference **5**
3.1 sphinx_config_schema 6

`sphinx-config-schema` is a Sphinx extension that renders a JSON schema as a copyable TOML, YAML, or JSON example. Each key in the example shows the description of that setting as a tooltip, via `sphinx-tippy`, and links to itself so that it can be shared as a permalink.

This is useful for tools that read their configuration from a file, such as a `[tool.*]` table in `pyproject.toml`, and that generate a JSON schema for it, for instance for editor completion.

1 Installation

```
pip install sphinx-config-schema
```

Then add the extension to the `extensions` list in your `conf.py`. It loads `sphinx_tippy` automatically:

Listing 1: conf.py

```
extensions = [
    "sphinx_config_schema",
]
```

2 Usage

The *`config-schema`* directive takes the path to a JSON schema file, relative to the current document or, with a leading `/`, to the documentation source directory. The optional *`table`* option sets the dotted path of the table under which the

settings appear, *caption* adds a caption to the code block, *format* selects the output format, and *header-titles* titles *references* to TOML tables with their header:

MyST

```
```{config-schema} example.schema.json
:caption: pyproject.toml
:table: tool.example
:format: toml
:header-titles:
```
```

reST

```
.. config-schema:: example.schema.json
    :caption: pyproject.toml
    :table: tool.example
    :format: toml
    :header-titles:
```

The *format* option renders the example as `yaml` or `json` instead of the default `toml`. Settings are then nested by indentation instead of grouped under table headers. The tabs below show the same example schema in each format. Hover over a key to see what it does:

TOML

Listing 2: pyproject.toml

```toml
[tool.example]
output = "build"
verbose = false
log-level = "warning"
# color = "auto"
exclude = []
env = {}

[tool.example.server]
host = "example.org"  # required
port = 8080
headers = { Cache-Control = "no-cache", Content-Language = "en" }
```

YAML

Listing 3: config.yaml

```yaml
output: build
verbose: false
log-level: warning
color: null
exclude: []
env: {}
server:
  host: example.org  # required
```

(continues on next page)

(continued from previous page)

```
  port: 8080
  headers: {Cache-Control: no-cache, Content-Language: en}
```

JSON

Listing 4: config.json

```
{
  "output": "build",
  "verbose": false,
  "log-level": "warning",
  "color": null,
  "exclude": [],
  "env": {},
  "server": {
    "host": "example.org",
    "port": 8080,
    "headers": {"Cache-Control": "no-cache", "Content-Language": "en"}
  }
}
```

The schema controls how the configuration is rendered:

- Each setting shows its `default` or, if it has no usable default, its first non-null example or enum choice. Enum choices are also collected from `anyOf` and `oneOf` alternatives and local `$ref` references, unless an alternative accepts any value of its type, as in a nullable boolean. An empty list or table default counts as a placeholder and yields to an example. Settings without a usable value are omitted, or reported as errors when the *`strict`* flag is enabled.
- A free-form table, an object with `additionalProperties` but no `properties`, shows its default or example like any other setting, such as *`server.headers`*. Without one, it appears as an empty table, such as *`env`*, so that readers see that it exists.
- Settings with a `null` default keep that default in YAML and JSON. Since TOML has no null value, the example is commented out there, so copying the block does not enable an optional setting. For instance, *`color`* has a nullable enum and a `null` default, so the TOML tab shows `# color = "auto"`.
- Objects with their own `properties` become nested tables, including objects reached through local `$ref` references such as those generated by pydantic.
- Each key has a tooltip containing its `description`, its actual default, allowed enum values, and any examples other than the value shown. Text in double or single backticks in descriptions renders as inline code, and a blank line starts a new paragraph. The tooltip of the last component of the *`table`* path shows the schema's top-level `description`.
- Keys listed under `required` get an asterisk in the example and a required badge in their tooltip, such as *`server.host`*. The asterisk is not copied with the example. Both can be restyled through the `config-schema-required` and `config-schema-badge` CSS classes.
- Each key links to itself and can be *referenced*.

2.1 Referencing options

Use the *config-schema:option* role or its short alias *cfg* to link to an option. Both render the key as inline code and, without an explicit title, use the key path below the *table* as the title, such as *server.host*. The standard ref role accepts the same labels but needs an explicit title and renders it as plain text, as in *the output option*. Unnamed examples use labels qualified by the document name and the format:

MyST

Option

```
{config-schema:option}`index:tool.example.output-toml`
```

cfg

```
{cfg}`index:tool.example.output-toml`
```

ref

```
{ref}`the output option <index:tool.example.output-toml>`
```

reST

Option

```
:config-schema:option:`index:tool.example.output-toml`
```

cfg

```
:cfg:`index:tool.example.output-toml`
```

ref

```
:ref:`the output option <index:tool.example.output-toml>`
```

The full *table* path remains part of each label, with dots separating the levels of the hierarchy and a hyphen before the format. Repeated examples with the same format on one page receive numeric suffixes, such as `index:output-yaml-2`. These references resolve to *output* in the TOML tab and *output* in the JSON tab above.

Set *name* to give an example a stable namespace instead. The YAML tab above does this with `:name: settings`, so its *output* option is referenced with:

MyST

```
{config-schema:option}`settings.output`
```

reST

```
:config-schema:option:`settings.output`
```

As in Sphinx's Python domain, a leading `.` matches every label whose key path ends with the target, and `~` shortens the title to the last key. For instance, `{cfg}`.tool.example.server.host`` renders as *server.host* and `{cfg}`~.tool.example.server.host`` as *host*, where `tool.example` selects the TOML tab over the YAML and JSON tabs.

When a suffix matches several labels, those on the current page take precedence, followed by those whose complete key path equals the target. If several candidates remain, Sphinx warns and lists them. Targets without a leading dot are exact labels. In both cases, keys can be spelled as in the schema, such as `log_level`, or as in the label, such as `log-level`.

A reference to the table of an example shows its full path, such as `tool.example`, also when that table has no settings of its own and only appears in the headers of its subtables. With *`header-titles`*, the tables of a TOML example are titled with their header instead, as in *`[tool.example]`* for the TOML tab above.

When the PyData Sphinx Theme or a theme derived from it is used, the tooltips take its colors and follow light and dark mode.

2.2 Other output formats

The example is built from regular docutils nodes, so it also appears in other output formats, such as plain text and LaTeX. There, each key becomes a label that can be referenced. In LaTeX output, the example is a highlighted code block with its caption, like any other code block. Tooltips only appear in HTML. Instead of the asterisk, the line of a required key ends in a `# required` comment, except in JSON, which has no comments.

3 Directive reference

`.. config-schema::` `path`

Render the JSON schema at `path` as a configuration example. The path is relative to the current document or, with a leading `/`, to the documentation source directory.

`:caption: text (text)`

Caption of the code block, typically the name of the configuration file.

`:format: toml, yaml or json (``toml`` (default), ``yaml`` or ``json``)`

Serialization format of the example.

`:header-titles: (flag)`

Title references to the tables of a TOML example with their header, such as `[tool.example.server]`, instead of the key path. Has no effect in YAML and JSON, which have no table headers.

`:name: text (text)`

Name the example so it can be referenced with an explicit title using the *`config-schema:option`* role. Option and nested-table labels use this name as a prefix instead of adding a format suffix. Names must be unique across the documentation; reusing a name on the same page produces a warning.

`:strict: (flag)`

Report settings that have no `default`, `examples`, or `enum` as an error instead of leaving them out.

`:table: dotted.path (text)`

Dotted path of the table under which the settings appear, such as `tool.example`.

`:config-schema:option:`

Link to a key of a *`config-schema`* example as inline code. See *Referencing options* for label syntax and title formatting.

`:cfg:`

Short alias of *`config-schema:option`*.

3.1 sphinx_config_schema

```
import sphinx_config_schema
```

Render a JSON schema as a copyable TOML, YAML, or JSON example in Sphinx.

OPTION_ROLE_ALIAS = 'cfg'

Short name under which *config-schema:option* is also registered.

class ConfigSchemaDirective(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: SphinxDirective

Render a copyable TOML, YAML, or JSON example from a JSON schema.

Each key in the example is an anchor that sphinx-tippy turns into a tooltip with the description of that setting and the values that the example does not show.

Submodules and Subpackages

codeblock

```
import sphinx_config_schema.codeblock
```

Build the example as a highlighted code block in which each key is an anchor.

create_literal_block(*settings: Sequence[*Entry*]*, *fmt: str*, *rawsource: str*) → Element

Build the example as a code block in which each key is an anchor.

The anchor is what sphinx-tippy attaches the tooltip to; it refers to itself so that the key doubles as a permalink to the setting.

The example is highlighted as a whole, so that the lexer sees the full context, and the tokens of each key are then wrapped in its anchor.

Sphinx highlights a literal block again when its `rawsource` equals its text, which would drop the inline nodes built here. A `rawsource` that differs from the text, such as the directive name, marks the block as a parsed literal instead. The surrounding containers give the block the class names that themes and sphinx-copybutton expect from a regular code block.

render

```
import sphinx_config_schema.render
```

Render the settings as TOML, YAML, or JSON text.

FORMATS = ('toml', 'yaml', 'json')

Formats in which the example can be rendered.

render(*settings: Sequence[*Entry*]*, *fmt: str = 'toml'*) → str

Render the example as plain text.

```
>>> from sphinx_config_schema.schema import Setting
>>> settings = [Setting(("tool", "demo"), "transport", "auto", "")]
>>> print(render(settings))
[tool.demo]
transport = "auto"
>>> print(render(settings, "yaml"))
```

(continues on next page)

(continued from previous page)

```
tool:
  demo:
    transport: auto
```

to_toml_value(*value: Any*) → str

Render a JSON value as an inline TOML value.

```
>>> to_toml_value({"name": "demo", "paths": ["a", "b"], "strict": True})
'{ name = "demo", paths = ["a", "b"], strict = true }'
```

to_yaml_value(*value: Any*) → str

Render a JSON value as a YAML flow value.

```
>>> to_yaml_value({"name": "demo", "paths": ["a", "b"], "strict": True, "v": "1.0"})
'{name: demo, paths: [a, b], strict: true, v: "1.0"}'
```

schema

```
import sphinx_config_schema.schema
```

Collect the settings of a JSON schema that the example shows.

NO_DEFAULT

Default of a *Setting* whose schema has none, as None is a valid default.

class Setting(*table: tuple[str, ...]*, *name: str*, *value: Any*, *description: str*, *examples: tuple[Any, ...] = ()*, *allowed: tuple[Any, ...] = ()*, *default: Any = NO_DEFAULT*, *value_source: Literal['default', 'example', 'enum', 'placeholder'] | None = None*, *required: bool = False*, *_anchor_prefix: str = ''*, *_anchor_suffix: str = ''*)

Bases: object

One key-value pair in the generated example.

table: tuple[str, ...]

Keys of the enclosing tables, one per nesting level, such as ("tool", "demo") for [tool.demo]; empty at the top level.

name: str

Unquoted key of the setting within its table.

value: Any

Value shown in the example, which *value_source* traces back to the schema.

description: str

Description from the schema for the tooltip, in which inline literals become code.

examples: tuple[Any, ...] = ()

Example values from the schema, listed in the tooltip unless *value* shows them.

allowed: tuple[Any, ...] = ()

The values that the schema allows for this setting through enum.

default: `Any` = `NO_DEFAULT`

The actual schema default, kept separate from the displayed example.

value_source: `Literal['default', 'example', 'enum', 'placeholder'] | None` = `None`

Origin of *value*; unset for settings constructed without schema metadata.

A `"placeholder"` is the empty mapping shown for a free-form table, an object with `additionalProperties` but neither `properties` nor a value to show.

required: `bool` = `False`

Whether the enclosing object lists the key under `required`.

property path: `tuple[str, ...]`

Full path of the key, including its *name*.

property anchor: `str`

HTML ID of the key, from which its `ref` label is formed.

class Table(*table: tuple[str, ...]*, *name: str*, *description: str*, *required: bool = False*, *_anchor_prefix: str = ''*, *_anchor_suffix: str = ''*)

Bases: `object`

A nested table in the generated example, which holds settings of its own.

table: `tuple[str, ...]`

Keys of the enclosing tables, as in *Setting.table*.

name: `str`

Unquoted key of the table within its parent.

description: `str`

Description from the schema for the tooltip, as in *Setting.description*.

required: `bool` = `False`

Whether the enclosing object lists the table under `required`.

property path: `tuple[str, ...]`

Full path of the table, including its *name*.

property anchor: `str`

HTML ID of the key, from which its `ref` label is formed.

Entry = sphinx_config_schema.schema.Setting | sphinx_config_schema.schema.Table

A key in the generated example.

iter_settings(*schema: dict[str, Any]*, *table: tuple[str, ...] = ()*, ** (Keyword-only parameters separator (PEP 3102))*, *strict: bool = False*) → Iterator[*Entry*]

Collect the settings with a default or example value from a JSON schema.

Settings without a usable value are skipped, or reported with a `ValueError` after all other settings have been collected if `strict` is set. Nested tables are yielded as a *Table*, so that their key can carry a description as well. So is the last part of `table`, with the description of the schema itself.

```
>>> schema = {
...     "description": "Demo settings.",
...     "properties": {
...         "name": {"description": "Project name.", "examples": ["demo"]},
...         "optional": {"type": "string"},
```

(continues on next page)

(continued from previous page)

```
...     },
... }
>>> for entry in iter_settings(schema, table=("tool", "demo")):
...     print(entry)
Table(table=('tool',), name='demo', description='Demo settings.')
Setting(table=('tool', 'demo'), name='name', value='demo', description='Project␣
↪name.')
>>> list(iter_settings(schema, strict=True))
Traceback (most recent call last):
    ...
ValueError: Settings without a default or example: optional
```

tooltips

```
import sphinx_config_schema.tooltips
```

Create the tooltips of the keys and hand them to sphinx-tippy.

to_tooltip(*description: str*) → str

Convert a schema description to the HTML shown in its tooltip.

Inline literals become code, and so does interpreted text in single backticks. Projects that document Python usually render the latter as cross-references, which Sphinx shows as code, and Markdown renderers of the schema show it as code as well. A role prefix is dropped, and so are the ~ and ! modifiers of a reference. A reference with an explicit title shows only the title. Paragraphs separated by a blank line become separate paragraphs.

```
>>> to_tooltip("Arguments for ``pytest``, such as ``-n auto``.")
'<p>Arguments for <code>pytest</code>, such as <code>-n auto</code>.</p>'
>>> to_tooltip("Requires `host`.")
'<p>Requires <code>host</code>.</p>'
>>> to_tooltip("Run :program:`pwa` with :py:obj:`~module.Settings`.")
'<p>Run <code>pwa</code> with <code>Settings</code>.</p>'
>>> to_tooltip("First.\n\nSecond.")
'<p>First.</p><p>Second.</p>'
```