

---

# **sphinxcontrib.datatemplates** **Documentation**

*Release 0.0.0*

**Doug Hellmann**

**Jun 09, 2019**



---

## Contents

---

<b>1</b>	<b>Using datatemplate</b>	<b>3</b>
1.1	Template Context . . . . .	3
1.2	Template Helpers . . . . .	4
<b>2</b>	<b>JSON Samples</b>	<b>5</b>
2.1	Data File . . . . .	5
2.2	Template File . . . . .	5
2.3	Rendered Output . . . . .	6
<b>3</b>	<b>YAML Samples</b>	<b>7</b>
3.1	Data File . . . . .	7
3.2	Template File . . . . .	7
3.3	Rendered Output . . . . .	8
<b>4</b>	<b>CSV Samples</b>	<b>9</b>
4.1	Data File . . . . .	9
4.2	Template File . . . . .	9
4.3	Code . . . . .	10
4.4	Rendered Output . . . . .	10
<b>5</b>	<b>XML Samples</b>	<b>11</b>
5.1	Data File . . . . .	11
5.2	Template File . . . . .	11
5.3	Rendered Output . . . . .	12
<b>6</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



`sphinxcontrib.datatemplates` helps you use static data in machine readable format in your documentation by letting you define Jinja2 templates to turn JSON, YAML, XML, or CSV data into reStructuredText for Sphinx to render as part of its output.

- Repo: <https://github.com/sphinxcontrib/sphinxcontrib.datatemplates>
- Docs: <http://sphinxcontribdatatemplates.readthedocs.io/>



---

## Using datatemplate

---

The `datatemplate` directive is the interface between the data source and the rendering template. It requires two parameters.

.. **datatemplate::**

**source** The source file, relative to the documentation build directory.

**template** The name of a template file on the Sphinx template search path.

**csvheader optional flag, CSV only** Set to use `csv.DictReader` for reading the file. If not set `csv.reader()` is used.

**csvdialect optional, CSV only, either auto or one from `csv.list_dialects()`** Set to select a specific `csv.Dialect`. Set to `auto`, to try autodetection. If not set the default dialect is used.

### 1.1 Template Context

When a `datatemplate` directive is processed, the data is passed to the template through its context so that the symbol data is available as a global variable.

Format	data type
JSON	the object returned by <code>json.safe_load()</code> (varies based on input)
YAML	the object returned by the <a href="#">PyYAML</a> function <code>yaml.safe_load()</code> (varies based on input)
XML	<code>xml.etree.ElementTree.ElementTree</code>
CSV	list of tuple or dict (when <code>csvheaders</code> option is specified)

The data is loaded from the source and passed directly to the template. No pre-processing is done on the data, so the template needs to handle aspects like `None` values and fields that have values that may interfere with parsing `reStructuredText`.

## 1.2 Template Helpers

These helper functions are exposed using their short name (without the module prefix) in the template context.

`sphinxcontrib.datatemplates.helpers.make_list_table` (*headers*, *data*, *title=""*,  
*columns=None*)

Build a list-table directive.

### Parameters

- **headers** – List of header values.
- **data** – Iterable of row data, yielding lists or tuples with rows.
- **title** – Optional text to show as the table title.
- **columns** – Optional widths for the columns.

`sphinxcontrib.datatemplates.helpers.make_list_table_from_mappings` (*headers*,  
*data*, *title*,  
*columns=None*)

Build a list-table directive.

### Parameters

- **headers** – List of tuples containing header title and key value.
- **data** – Iterable of row data, yielding mappings with rows.
- **title** – Optional text to show as the table title.
- **columns** – Optional widths for the columns.



### 2.1 Data File

```
{
  "key1": "value1",
  "key2": [
    "list item 1",
    "list item 2",
    "list item 3"
  ],
  "nested-list": [
    ["a", "b", "c"],
    ["A", "B", "C"]
  ],
  "mapping-series": [
    {"cola": "a", "colb": "b", "colc": "c"},
    {"cola": "A", "colb": "B", "colc": "C"}
  ]
}
```

### 2.2 Template File

```
.. -*- mode: rst -*-

Static Heading
-----

Individual Item
~~~~~

{{ data['key1'] }}
```

(continues on next page)

```
List of Items
```

```
~~~~~
```

```
{% for item in data['key2'] %}
- {{item}}
{% endfor %}
```

```
Nested List Table
```

```
~~~~~
```

Rendering a table **from a list** of nested sequences using hard-coded headers.

```
{{ make_list_table(
    ['One', 'Two', 'Three'],
    data['nested-list'],
    title='Table from nested lists',
) }}
```

```
Mapping Series Table
```

```
~~~~~
```

Rendering a table **from a list** of nested dictionaries using dynamic headers.

```
{{ make_list_table_from_mappings(
    [('One', 'cola'), ('Two', 'colb'), ('Three', 'colc')],
    data['mapping-series'],
    title='Table from series of mappings',
) }}
```

## 2.3 Rendered Output

### 3.1 Data File

```
---
key1: value1
key2:
  - list item 1
  - list item 2
  - list item 3
nested-list:
  - ['a', 'b', 'c']
  - ['A', 'B', 'C']
mapping-series:
  - cola: a
    colb: b
    colc: c
  - cola: A
    colb: B
    colc: C
```

### 3.2 Template File

```
.. -*- mode: rst -*-

Static Heading
-----

Individual Item
~~~~~

{{ data['key1'] }}
```

(continues on next page)

```
List of Items
```

```
~~~~~
```

```
{% for item in data['key2'] %}
- {{item}}
{% endfor %}
```

```
Nested List Table
```

```
~~~~~
```

Rendering a table **from a list** of nested sequences using hard-coded headers.

```
{{ make_list_table(
    ['One', 'Two', 'Three'],
    data['nested-list'],
    title='Table from nested lists',
) }}
```

```
Mapping Series Table
```

```
~~~~~
```

Rendering a table **from a list** of nested dictionaries using dynamic headers.

```
{{ make_list_table_from_mappings(
    [('One', 'cola'), ('Two', 'colb'), ('Three', 'colc')],
    data['mapping-series'],
    title='Table from series of mappings',
) }}
```

### 3.3 Rendered Output

## 4.1 Data File

a	b	c	
Eins		Zwei	Drei
1	2	3	
I	II	III	

## 4.2 Template File

```
.. -*- mode: rst -*-

Static Heading
-----

Individual Cell in Row
~~~~~~~~~~~~~~~~~~~~~

{{ data[0].a }}

List of Cells in Row
~~~~~~~~~~~~~~~~~~~~~

{% for item in data[0].items() %}
- {{item[0]}}: {{item[1]}}
{% endfor %}

Mapping Series Table
~~~~~~~~~~~~~~~~~~~~~

Rendering a table from a list of nested dictionaries using dynamic
```

(continues on next page)

(continued from previous page)

```
headers.  
  
{{ make_list_table_from_mappings(  
    [('One', 'a'), ('Two', 'b'), ('Three', 'c')],  
    data,  
    title='Table from series of mappings',  
    ) }}
```

## 4.3 Code

```
.. datatemplate::  
   :source: sample.csv  
   :template: csv-sample.tmpl  
   :csvheaders:  
   :csvdialect: excel-tab
```

## 4.4 Rendered Output

## 5.1 Data File

```
<sample>
  <key1>value1</key1>
  <key2>
    <item>list item 1</item>
    <item>list item 2</item>
    <item special='yes'>list item 3</item>
  </key2>
  <mappingseries>
    <mapping>
      <cola special='yes'>a</cola>
      <colb>b</colb>
      <colc>c</colc>
    </mapping>
    <mapping>
      <cola>A</cola>
      <colb special='yes'>B</colb>
      <colc>C</colc>
    </mapping>
  </mappingseries>
</sample>
```

## 5.2 Template File

```
.. -*- mode: rst -*-
```

```
Static Heading
```

```
-----
```

(continues on next page)

(continued from previous page)

```
Individual Item
~~~~~

{{ data.find('key1').text }}

List of Items
~~~~~

{% for item in data.find('key2') %}
- {{item.text}}
{% endfor %}

XPath for Items
~~~~~

See `XPath support <https://docs.python.org/3/library/xml.etree.elementtree.html
↪#xpath-support>`_

{% for item in data.findall("./*[@special='yes']") %}
- {{item.text}}
{% endfor %}
```

## 5.3 Rendered Output



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**S**

`sphinxcontrib.datatemplates.helpers`, 4



## D

`datatemplate` (*directive*), 3

## M

`make_list_table()` (*in module sphinxcontrib.datatemplates.helpers*), 4

`make_list_table_from_mappings()` (*in module sphinxcontrib.datatemplates.helpers*), 4

## S

`sphinxcontrib.datatemplates.helpers` (*module*), 4