eds4jinja2

Release 0.1.36

Eugeniu Costetchi

May 06, 2022

CONTENTS

An easy way to reports generation with Jinja2 templates. With Embedded Datasource Specifications inside Jinja2 templates, you can fetch the data you need on the spot.

You can specify the file data source in your JINJA templates as follows. The *path* can be specified as absolute or relative to the running script.

```
{% set content, error = from_file(path).fetch_tree() %}
content:  {{ content }}
error: {{ error }}
```

In case you need to fetch data from a SPARQL endpoint, define a sparql_endpoint (usually in the global configuration) and a custom sparql query_text. Use them in the template like this:

```
{% set query_string = "select * where {?s ?p ?o} limit 10" %}
{% set content, error =
    from_endpoint(endpoint).with_query(query_string).fetch_tabular() %}
content:  {{ content }}
error: {{ error }}
```

Currently supported data sources are:

- from file - tabular (CSV, Excel, etc.) and tree-structured (JSON, YAML, etc.)
- from SPARQL endpoint - through a select query or describe request

Not yet supported data sources are:

- from a XML file
- from a REST API
- from a relational database
- from a NoSQL database, e.g. MongoDB
- from a graph database, e.g. TinkerPop, Neo4j

CHAPTER

ONE

BUT WHY?

Imagine, for example, that you need to build a report from multiple SPARQL endpoints and a configuration data provided in a local JSON file.

What you would normally do is retrieve first all the data and then pass it to the rendered template.

Alternatively, with **eds4jinja2** you can simply specify how the data shall be retrieved and then just use it in the template.

1. instantiate the template from eds4jinja2 environment

```
from eds4jinja2.builders.jinja_builder import build_eds_environment
from jinja2 import PackageLoader

loader=PackageLoader('your_application', 'templates')
env = build_eds_environment(loader=loader)
```

2. create a template file 'mytemplate.txt' that looks like this

```
{% set config_content, error = from_file("path/to/the/config/file.json").fetch_tree() %}
The configuration content:
{{ config_content }}

{% set query_string = "select * where {?s ?p ?o} limit 10" %}
{% set endpoint_content, error = from_endpoint(endpoint).with_query(query_string).fetch_
↪tree() %}
content: {{ endpoint_content }}
error: {{ error }}
```

3. render the template with no context. The context is dynamically generated during the template rendering. Bingo!

```
rendered_text = template.render()
```

CHAPTER

TWO

SO, WHAT ARE THE BENEFITS?

- your python code is agnostic of what data the template displays
- data fetching functionality is no longer part of the python context-building logic
- the queries and the template to visualise the query result set are tightly coupled and easy to modify
- this allows for building quickly custom visualisation templates (or modifying existent ones), before you even decide what the final query looks like

CHAPTER

THREE

REPORTBUILDER CLASS USAGE

`ReportBuilder` accepts 4 parameters when instantiating:

`target_path` (required) - the folder where the required resources are found.

`config_file` (optional) - the name of the configuration file (defaults `config.json`).

`output_path` (optional) - the output folder where the result of the rendering will be created.

`additional_config` (optional) - additional config parameters that are added to the default ones and are overwritten (deep update) in the project `config.json`.

Example:

```
target_path = 'some/path/for/template'
config_file = 'other.json'
output_path = 'some/other/path'
additional_config = {"default_endpoint": 'http://localhost:9999'}
report_builder = ReportBuilder(target_path=location,
                               config_file=config_file,
                               output_path=output_path,
                               additional_config=additional_config)
```

CHAPTER

FOUR

CLI USAGE

The command to run the report generation is *mkreport*

CHAPTER

FIVE

THE COMMAND LINE INTERFACE HAS THREE ARGUMENTS:

- **–target** (optional): the directory where eds4jinja2 will load the content from, for rendering a template; this directory has a mandatory layout (see below)
- **–output** (optional): the directory where eds4jinja2 will place the rendered output; if this parameter is not specified, it will be placed in an "output" subdirectory that will be created where the "–target" parameter points to
- **–config** (optional): the name of the configuration file from where eds4jinja2 will load the configuration that's passed to Jinja2; default "config.json"

Example:

```
mkreport --target=template --output=report_location --config=report_config.json
```

CHAPTER

SIX

TARGET DIRECTORY LAYOUT:

By convention, the target directory **must** contain:

1. a configuration file in JSON format which serves two purposes:
   - it specifies the main template for eds2jinja to start with (this template may refer to other additional templates)
   - it specifies a list of variables needed to render the aforementioned template(s); the list may contain anything you need in your templates
2. a directory named "templates" where all of your templates reside
3. if your template(s) need additional static resources (such as CSS/JS/etc files) a directory named "static" where all of these resources must reside; the contents of this directory will be copied to the output folder and its tree structure preserved

Example:

```
{
    "template": "main.html",
    "template_flavour_syntax": "html",
    "conf":
    {
        "default_endpoint" : "http://example.com/path/sparqlendpoint",
        "title": "Pretty printed relevant information",
        "type": "report",
        "author": "Your name here"
        "nexted_properties": {
            "graph": "http://publications.europa.eu/resources/authority/lam/
↪DocumentProperty"
        },
    }
}
```

CHAPTER

SEVEN

LATEX TEMPLATES:

It is possible to write templates with LaTex documents. To do so, first make sure you have specified the template flavour in the config file

```
{
    "template": "main.tex",
    "template_flavour_syntax": "latex",
    "conf":
    {
        "title": "Pretty printed relevant information",
        ...
```

Next write your templates using the following conventions:

- Blocks: `\BLOCK{block block_name}\BLOCK{endblock}`
- Line statement: `%- line instruction`
- Variables: `\VAR{variable_name}`
- Comments (long form): `\#{This is a long-form Jinja comment}`
- Comments (short form): `%# This is a short-form Jinja comment`

An example `jinja-test.tex` is available below:

```
\documentclass{article}
\begin{document}
\section{Example}
An example document using \LaTeX, Python, and Jinja.

% This is a regular LaTeX comment
\section{\VAR{section1}}
\#{This is a long-form Jinja comment}
\begin{itemize}
\BLOCK{ for x in range(0, 3) }
  \item Counting: \VAR{x}
\BLOCK{ endfor }
\end{itemize}

\section{\VAR{section2}}
%# This is a short-form Jinja comment
\begin{itemize}
%- for x in range(0, 3)
  \item Counting: \VAR{x}
```

(continues on next page)

(continued from previous page)

```
%- endfor
\end{itemize}

\end{document}
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search