nuketemplate Documentation

Release 0.2.0

Florian Einfalt

Jun 28, 2018

Contents

Contents:

CHAPTER 1

Installation

To install `nuketemplate`, type:

```
$ pip install nuketemplate
```

Open Nuke's `init.py` file and add:

```
nuke.pluginAddPath('/path/to/your/local/python/site-packages')
```

CHAPTER 2

Getting Started

2.1 Folder structure

`nuketemplate` is a wrapper around the Jinja 2 templating engine. If you are not familiar with Jinja 2 templates, read the documentation Jinja 2.

Let's start with a simple template structure. Create a folder structure, like so:

```
/attrs
    main.j2
    attrs_write.j2
/nodes
    main.j2
    nodes.j2
```

The `main.j2` file is the root template in which we will include all other sub templates (`attrs_write.j2` and `nodes.j2`).

2.2 Defining attributes

The final format of all attributes must be a valid JSON object so make sure the aggregator in `/attrs/main.j2` is a valid JSON object `{}`.

Add the following code snippet to `/attrs/main.j2`. This code will parse every file in the same folder that has a file name starting with `attrs_` and include it in the final set of attributes. That means it is very easy to extend by simply creating additional files with the same naming pattern.

```
{
    {% for attr_template in attr_templates %}
        {% include attr_template | basename %}
        {% if not loop.last %},{% endif %}
    {% endfor %}
}
```

Add the following code snippet to `/attrs/attrs_write.j2`. Again, this will be included in the JSON object in `/attrs/main.j2`.

```
"output": {
    "flat": {
        "channels": "rgb",
        "file_type": "jpg",
        "_jpeg_quality": 1.0,
        "_jpeg_sub_sampling": "4:4:4",
        "colorspace": "sRGB"
    },
    "layered": {
        "channels": "rgba",
        "file_type": "png",
        "datatype": "8 bit",
        "colorspace": "sRGB"
    }
}
```

2.3 Defining Node Graphs in Templates

Add the following code snippet to `/nodes/main.j2`. This will reference the sub template `nodes.j2` and include it in the list of sub templates.

```
[
    {% include "nodes.j2" %}
]
```

Note: The final format of all nodes must be a valid JSON object so make sure the aggregator in `/nodes/main.j2` is a valid JSON object `[]`.

To add further sub templates simply extend the list with further `include` directives.

Add the following code snippet to `/nodes/nodes.j2`.

```
{
    ">>": "write",
    "write": {
        "type": "Write",
        "inputs": ["cutout"],
        "attr": {{ attrs["output"]["layered"] | tojson }},
        "id": {{ id_data | tojson }}
    },
    "cutout": {
        "type": "Premult",
        "inputs": ["alpha"]
    },
    "alpha": {
        "type": "Shuffle",
        "inputs": ["<<"]
    }
}
```

This is a simple sub template in the standard format. The first item is the `start` indicator, which tells the

AbstractTemplateConverter which node to start with. The standard key to indicate the `start` value is >> but this can be freely defined in the *AbstractTemplateConverter* constructor.

One node in the template will have to have the `end` indicator as one of its inputs (the default is << but this can be redefined in the *AbstractTemplateConverter* constructor). This indicates where the sub graph will be connected to the next sub graph in the list.

Conceptually it is best to think about this node graph as a Nuke graph "upside-down", the `start` node being the last node in the graph.

A node in the template format must implement the following format:

```
node name: {
    type: Nuke node type,
    inputs: List of inputs from the same subgraph,
    attr: {
        Nuke attribute: value
    },
    id: {
        UUID type: UUID
    }
}
```

Note: The `attr` and `id` keys are optional and will be ignored if not specified.

All values can be substituted using Jinja 2 templating, like in the example.

Note: Use the `tojson` filter to ensure values are converted to valid JSON

2.4 Building the Nuke Node Graph

With the templates in place, start Nuke and import the following `nuketemplate` classes:

```
from nuketemplate.template import AbstractTemplate
from nuketemplate.convert import AbstractTemplateConverter
from nuketemplate.build import NukeGraphBuilder
```

Initialise an *AbstractTemplate* like so:

```
template = AbstractTemplate(root='/path/to/template/folder')
```

Since we have specified the variable `id_data` in our template, we will have to supply this data.

```
data = {'id_data': {'uuid': '3a5c2055-e288-4bc2-90cb-dc0fb9ae462e'}}
```

Now, pass the data to the *render()* function, like so:

```
template.render(**data)
```

After template rendering, initialise a *AbstractTemplateConverter* and run the *convert()* function, like so:

```
converter = AbstractTemplateConverter(template.template)
converter.convert()
```

To build the Nuke node graph, initialise a *NukeGraphBuilder* and run the *build()* function, like so:

```
builder = NukeGraphBuilder(converter.result)
builder.build()
```

CHAPTER 3

API Documentation

3.1 Template module

class nuketemplate.template.**AbstractTemplate**(*root*, *attrs='attrs'*, *nodes='nodes'*, *template=[]*)

Template class, automates Jinja2 loader and environment generation, wraps Jinja2 rendering and JSON encoding.

Parameters

- **root** (*str*) – Template root location
- **nodes** (*str*) – Node templates' folder name, (default: `nodes`)
- **attrs** (*str*) – Attribute templates' folder name, (default: `attrs`)

render(*template='main.j2'*, ***kwargs*)

Compile the main node `template` to JSON and store as an instance attribute

Parameters

- **template** (*str*) – Main template name
- ****kwargs** (*dict*) – Template data

save(*filename='template.json'*)

Given a `root` folder and a `filename`, save the template JSON encoded to a file.

Parameters

- **root** (*str*) – Saving location
- **filename** (*str*) – Filename

3.2 Graph module

class nuketemplate.graph.**AbstractGraph**(*nx_graph*, *start=None*, *end=None*, *end_slot=0*)
Abstraction of a NetworkX Directed Graph, adds `start`, `end` attributes for simplified graph combination.

Parameters

- **nx_graph** (`Graph`) – NetworkX directed graph
- **start** (*GenericNode* or *NukeNode*) – Start node
- **end** (*str*) – *GenericNode* or *NukeNode*
- **end_slot** (*int*) – End node input slot, for nodes with multiple inputs

class nuketemplate.graph.**GenericNode**(*name*)
Generic Node, used with *AbstractGraph*

Parameters **name** (*str*) – Node name

class nuketemplate.graph.**NukeNode**(*name*, *type*, *attr*, *id*, *nuke_node=None*, *nuke_name=None*)
Nuke Node, used with *AbstractGraph*, inherits from *GenericNode*

Parameters

- **name** (*str*) – Node name
- **_type** (*str*) – Nuke node type
- **_attr** (*dict*) – Nuke node attributes
- **_id** (*dict*) – Node UUIDs
- **_nuke_node** (`Node`) – Nuke node, generated by *build()*
- **_nuke_name** (*str*) – Nuke node name, generated by *build()*

build()
Build and return the Nuke node.

Returns Nuke node

Return type `Node`

nuketemplate.graph.**is_node_in_nx_graph**(*instance*, *attribute*, *value*)
Input validator for *AbstractGraph*. Check whether start and end node inputs are part of the graph.

Parameters

- **instance** (*AbstractGraph*) – *AbstractGraph* instance
- **_type** (`Attribute`) – Attribute
- **_attr** (*GenericNode* or *NukeNode*) – Input node

3.3 Convert module

class nuketemplate.convert.**AbstractTemplateConverter**(*template*, *start='>>'*, *end='<<'*, *subgraphs=[]*, *result=None*)
Template to Graph Converter

Parameters

- **template** (*list, dict*) – JSON Template
- **start** (*str*) – Start characters, default: >>
- **end** (*str*) – End characters, default: <<

convert()
Convert the JSON template into an *AbstractGraph*, if the template consists of multiple sub graphs, convert and combine otherwise convert in one pass.

to_dot(*dot_filename='graph.dot'*)
Save the converted graph to a dot file at location `dot_filename`

Parameters **dot_filename** (*str*) – Filename, default: `graph.dot`

to_png(*png_filename='graph.png'*, *dot_executable='/usr/local/bin/dot'*)
Save the converted graph to a png file at location `png_filename`

Parameters

- **png_filename** (*str*) – Filename, default: `graph.png`
- **dot_executable** (*str*) – Path to the dot executable, default: `/usr/local/bin/dot`

3.4 Build module

class nuketemplate.build.**NukeGraphBuilder**(*abstract_graph*)
NukeGraphBuilder, convert an *AbstractGraph* to a Nuke compositing script

Parameters **abstract_graph** (*AbstractGraph*) – Abstract graph

build()
Build the Nuke node graph from the object's `abstract_graph`.

Returns Number of nodes built

Return type int

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index