Lucidity

Release 1.1.0

March 08, 2014

Filesystem templating and management.

CHAPTER 1

Guide

Overview and examples of using the system in practice.

1.1 Introduction

Lucidity is a framework for templating filesystem structure.

It works using regular expressions, but hides much of the verbosity through the use of simple placeholders (such as you see in Python string formatting).

Consider the following paths:

```
/jobs/monty/assets/circus/model/high/circus_high_v001.abc
/jobs/monty/assets/circus/model/low/circus_low_v001.abc
/jobs/monty/assets/parrot/model/high/parrot_high_v002.abc
```

A regular expression to describe them might be:

```
'/jobs/(?P<job>[\w_]+?)/assets/(?P<asset_name>[\w_]+?)/model/(?P<lod>[\w_]+?)/(?P<asset_name>[\w_]+?)
```

Meanwhile, the Lucidity pattern would be:

```
'/jobs/{job}/assets/{asset_name}/model/{lod}/{asset_name}_{lod}_v{version}.{filetype}'
```

With Lucidity you store this pattern as a template and can then use that template to generate paths from data as well as extract data from matching paths in a standard fashion.

Read the *Tutorial* to find out more.

1.1.1 Copyright & License

1.2 Installation

Installing Lucidity is simple with pip:

```
$ pip install lucidity
```

If the Cheeseshop (a.k.a. PyPI) is down, you can also install Lucidity from one of the mirrors:

```
$ pip install --use-mirrors lucidity
```

Alternatively, you may wish to download manually from Github where Lucidity is actively developed.

You can clone the public repository:

```
$ git clone git://github.com/4degrees/lucidity.git
```

Or download an appropriate tarball or zipball

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages:

```
$ python setup.py install
```

1.2.1 Dependencies

- Python >= 2.6, < 3
- Bunch >= 1.0.1
- Regex >= 2.4.26

> **Note:** If you are having trouble installing Regex via Pip you might want to install the pre-built extension from http://www.lfd.uci.edu/~gohlke/pythonlibs/

For testing:

- Pytest >= 2.3.5

1.3 Tutorial

This tutorial gives a good introduction to using Lucidity.

First make sure that you have Lucidity *installed*.

1.3.1 Patterns

Lucidity uses patterns to represent a path structure. A pattern is very much like the format you would use in a Python string format expression.

For example, a pattern to represent this filepath:

```
'/jobs/monty/assets/circus/model/high/circus_high_v001.abc'
```

Could be:

```
'/jobs/{job}/assets/{asset_name}/model/{lod}/{asset_name}_{lod}_v{version}.{filetype}'
```

Each `{name}` in braces is a variable that can either be extracted from a matching path, or substituted with a provided value when constructing a path. The variable is referred to as a *placeholder*.

1.3.2 Templates

A `Template` is a simple container for a pattern.

First, import the package:

```
>>> import lucidity
```

Now, construct a template with the pattern above:

```
>>> template = lucidity.Template('model', '/jobs/{job}/assets/{asset_name}/model/{lod}/{asset_name}_
```

Note: The template must be given a name to identify it. The name becomes useful when you have a bunch of templates to manage.

1.3.3 Parsing

With a template defined we can now parse a path and extract data from it:

```
>>> path = '/jobs/monty/assets/circus/model/high/circus_high_v001.abc'
>>> data = template.parse(path)
>>> print data
{
    'job': 'monty',
    'asset_name': 'circus',
    'lod': 'high',
    'version': '001',
    'filetype': 'abc'
}
```

Note: When a group name in a pattern occurs more than once (as with `{asset_name}` and `{lod}` above), the last matching value is used for the returned data.

If a template's pattern does not match the path then `parse()` will raise a `ParseError`:

```
>>> print template.parse('/other/monty/assets')
ParseError: Input '/other/monty/assets' did not match template pattern.
```

Anchoring

By default, a pattern is anchored at the start, requiring that the start of a path match the pattern:

```
>>> job_template = lucidity.Template('job', '/job/{job}')
>>> print job_template.parse('/job/monty')
{'job': 'monty'}
>>> print job_template.parse('/job/monty/extra/path')
{'job': 'monty'}
>>> print job_template.parse('/other/job/monty')
ParseError: Input '/other/job/monty' did not match template pattern.
```

The anchoring can be changed when constructing a template by passing an *anchor* keyword in:

```
>>> filename_template = lucidity.Template(
...     'filename',
...     '{filename}.{index}.{ext}',
...     anchor=lucidity.Template.ANCHOR_END
... )
>>> print filename_template.parse('/some/path/to/file.0001.dpx')
{'filename': 'file', 'index': '0001', 'ext': 'dpx'}
```

The anchor can be one of:

- `ANCHOR_START` - Match pattern at the start of the string.
- `ANCHOR_END` - Match pattern at the end of the string.
- `ANCHOR_BOTH` - Match pattern exactly.
- `None` - Match pattern once anywhere in the string.

1.3.4 Formatting

It is also possible to pass a dictionary of data to a template in order to produce a path:

```
>>> data = {
...     'job': 'monty',
...     'asset_name': 'circus',
...     'lod': 'high',
...     'version': '001',
...     'filetype': 'abc'
... }
>>> path = template.format(data)
>>> print path
/jobs/monty/assets/circus/model/high/circus_high_v001.abc
```

In the example above, we haven't done more than could be achieved with standard Python string formatting. In the next sections, though, you will see the need for a dedicated `format()` method.

If the supplied data does not contain enough information to fill the template completely a `FormatError` will be raised:

```
>>> print template.format({})
FormatError: Could not format data {} due to missing key 'job'.
```

1.3.5 Nested Data Structures

Often the data structure you want to use will be more complex than a single level dictionary. Therefore, Lucidity also supports nested dictionaries when both parsing or formatting a path.

To indicate a nested structure, use a dotted notation in your placeholder name:

```
>>> template = lucidity.Template('job', '/jobs/{job.code}')
>>> print template.parse('/jobs/monty')
{'job': {'code': 'monty'}}
>>> print template.format({'job': {'code': 'monty'}})
/jobs/monty
```

Note: Unlike the standard Python format syntax, the dotted notation in Lucidity always refers to a nested item structure rather than attribute access.

1.3.6 Custom Regular Expressions

Lucidity works by constucting a regular expression from a pattern. It replaces all placeholders with a default regular expression that should suit most cases.

However, if you need to customise the regular expression you can do so either at a template level or per placeholder.

At The Template Level

To modify the default regular expression for a template, pass it is as an additional argument:

```
>>> template = lucidity.Template('name', 'pattern',
                                 default_placeholder_expression='[^/]+')
```

Per Placeholder

To alter the expression for a single placeholder, use a colon `:` after the placeholder name and follow with your custom expression:

```
>>> template = lucidity.Template('name', 'file_v{version:\d+}.ext')
```

Above, the *version* placeholder expression has been customised to only match one or more digits.

Note: If your custom expression requires the use of braces ({ }) you must escape them to distinguish them from the placeholder braces. Use a preceding backslash for the escape (\{, \}).

And of course, any custom expression text is omitted when formatting data:

```
>>> print template.format({'version': '001'})
file_v001.ext
```

1.4 Managing Multiple Templates

Representing different path structures requires the use of multiple templates.

Lucidity provides a few helper functions for dealing with multiple templates.

1.4.1 Template Discovery

Templates can be *discovered* by searching a list of paths for *mount points* that register template instances. By default, the list of paths is retrieved from the environment variable `LUCIDITY_TEMPLATE_PATH`.

To search and load templates in this way:

```
>>> import lucidity
>>> templates = lucidity.discover_templates()
```

To specify a specific list of paths just pass them to the function:

```
>>> templates = lucidity.discover_templates(paths=['/templates'])
```

By default each path will be recursively searched. You can disable this behaviour by setting the `recursive` keyword argument:

```
>>> templates = lucidity.discover_templates(recursive=False)
```

1.4.2 Template Mount Points

To write a template mount point, define a Python file containing a `register` function. The function should return a list of instantiated `Template` instances:

```
# templates.py

from lucidity import Template

def register():
    '''Register templates.'''
    return [
        Template('job', '/jobs/{job.code}'),
        Template('shot', '/jobs/{job.code}/shots/{scene.code}_{shot.code}')
    ]
```

Place the file on one of the search paths for `discover_templates()` to have it take effect.

1.4.3 Operations Against Multiple Templates

Lucidity also provides two top level functions to run a `parse` or `format` operation against multiple candidate templates using the first correct result found.

Given the following templates:

```
>>> import lucidity
>>> templates = [
...     lucidity.Template('model', '/jobs/{job}/assets/model/{lod}'),
...     lucidity.Template('rig', '/jobs/{job}/assets/rig/{rig_type}')
... ]
```

To perform a parse:

```
>>> print lucidity.parse('/jobs/monty/assets/rig/anim', templates)
({'job': 'monty', 'rig_type': 'anim'},
 Template(name='rig', pattern='/jobs/{job}/assets/rig/{rig_type}'))
```

To format data:

```
>>> print lucidity.format({'job': 'monty', 'rig_type': 'anim'}, templates)
('/jobs/monty/assets/rig/anim',
 Template(name='rig', pattern='/jobs/{job}/assets/rig/{rig_type}'))
```

Note: The return value is a tuple of `(result, template)`.

If no template could provide a result an appropriate error is raised (`ParseError` or `FormatError`).

CHAPTER 2

Reference

API reference providing details on the actual code.

2.1 lucidity

lucidity.**discover_templates**(*paths=None*, *recursive=True*)
Search *paths* for mount points and load templates from them.

paths should be a list of filesystem paths to search for mount points. If not specified will try to use value from environment variable `LUCIDITY_TEMPLATE_PATH`.

A mount point is a Python file that defines a 'register' function. The function should return a list of instantiated `Template` objects.

If *recursive* is True (the default) then all directories under a path will also be searched.

lucidity.**parse**(*path*, *templates*)
Parse *path* against *templates*.

path should be a string to parse.

templates should be a list of `Template` instances in the order that they should be tried.

Return `(data, template)` from first successful parse.

Raise `ParseError` if *path* is not parseable by any of the supplied *templates*.

lucidity.**format**(*data*, *templates*)
Format *data* using *templates*.

data should be a dictionary of data to format into a path.

templates should be a list of `Template` instances in the order that they should be tried.

Return `(path, template)` from first successful format.

Raise `FormatError` if *data* is not formattable by any of the supplied *templates*.

lucidity.**get_template**(*name*, *templates*)
Retrieve a template from *templates* by *name*.

Raise `NotFound` if no matching template with *name* found in *templates*.

2.1.1 template

class lucidity.template.**Template**(*name*, *pattern*, *anchor=1*, *default_placeholder_expression='[\w_.\-]+'*)

Bases: object

A template.

__init__(*name*, *pattern*, *anchor=1*, *default_placeholder_expression='[\w_.\-]+'*)

Initialise with *name* and *pattern*.

anchor determines how the pattern is anchored during a parse. A value of ANCHOR_START (the default) will match the pattern against the start of a path. ANCHOR_END will match against the end of a path. To anchor at both the start and end (a full path match) use ANCHOR_BOTH. Finally, None will try to match the pattern once anywhere in the path.

name

Return name of template.

pattern

Return template pattern.

parse(*path*)

Return dictionary of data extracted from *path* using this template.

Raise ParseError if *path* is not parseable by this template.

format(*data*)

Return a path formatted by applying *data* to this template.

Raise FormatError if *data* does not supply enough information to fill the template fields.

ANCHOR_BOTH = 3

ANCHOR_END = 2

ANCHOR_START = 1

2.1.2 error

Custom error classes.

exception lucidity.error.**ParseError**

Bases: exceptions.Exception

Raise when a template is unable to parse a path.

exception lucidity.error.**FormatError**

Bases: exceptions.Exception

Raise when a template is unable to format data into a path.

exception lucidity.error.**NotFound**

Bases: exceptions.Exception

Raise when an item cannot be found.

CHAPTER 3

Glossary

LUCIDITY_TEMPLATE_PATH
: Environment variable defining paths to search for template mount points. Can be multiple paths separated by the appropriate path separator for your operating system.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

e

l

t