Peppercorn Documentation

Release 0.6

Pylons Developers

April 22, 2022

Contents

A library for converting a token stream into a data structure comprised of sequences, mappings, and scalars, developed primarily for converting HTTP form post data into a richer data structure. It runs on Python 2.7, 3.4, 3.5, 3.6 and 3.7.

Example "bare" usage:

```
>>> fields = [
... ('name', 'project1'),
... ('title', 'Cool project'),
... ('__start__', 'series:mapping'),
... ('name', 'date series 1'),
... ('__start__', 'dates:sequence'),
... ('__start__', 'date:sequence'),
... ('day', '10'),
... ('month', '12'),
... ('year', '2008'),
... ('__end__', 'date:sequence'),
... ('__start__', 'date:sequence'),
... ('day', '10'),
... ('month', '12'),
... ('year', '2009'),
... ('__end__', 'date:sequence'),
... ('__end__', 'dates:sequence'),
... ('__start__', ':ignore'),
... ('selectall', ''),
... ('__end__', ''),
... ('__end__', 'series:mapping'),
... ]
 >>> from peppercorn import parse
 >>> return pprint.pprint(parse(fields))
 {'series':
   {'name':'date series 1',
    'dates': [['10', '12', '2008'], ['10', '12', '2009']]},
  'name': 'project1',
  'title': 'Cool project'}
```

A `__start__` token pushes a data structure onto the stack. Its value is composed of a name and a type, separated by a colon (e.g. `date:sequence`). Four `__start__` token types exist:

- `sequence`: begins a sequence. Subsequent data elements will be added as positional elements in the sequence.
- `mapping`: begins a mapping. Subsequent data elements will be added as key/value pairs in the mapping.
- `rename`: begins a special mode. The value of the first subsequent data element in the stream will be used within its parent sequence or mapping; any remaining data elements until the corresponding `__end__` marker are ignored.

  If the parent is a mapping, the key used in the mapping will be the name of the `rename` token (when `value="something:rename"`, the key will be `something`). The value will be the value of the first data element.

  If the parent is a sequence, the `rename` token name is ignored, and the value of the first data element is placed into the sequence.

  `rename` is mostly for radio controls; we surround sets of radio controls in a `rename` in order to provide a surrogate naming for a group of radio control elements. Radio control `name` attributes are used by the browser to perform grouping, so each radio control that is a member of a the same group must share a `name` attribute value. Moreover, this group name must be unique amongst all controls on the form to prevent "select bleeding" between radio controls. However, on the server side, we're uninterested in participating in this disambiguation process and it's easier to not know about it when the form is posted. We just want the selected value back in the pstruct to be recorded under a well-known name. This name will be the name of the `rename` token surrounding some radio controls.

- `ignore`: The subsequent data elements will be ignored (not added to the mapping or sequence) until the next `__end__` token. Useful when forms include a field designed for client side scripting, such as a "select all" checkbox in the middle of a series of checkboxes.

`__start__` markers can be unnamed; they are unnamed when their value does not contain a colon. For example, the start marker `('__start__', 'mapping')` begins a mapping with the implied name `''` (the empty string).

A sequence or mapping is closed when the corresponding `__end__` token for its `__start__` token is processed. Mappings and sequences can be nested arbitrarily. The value of an `__end__` token is optional; it is useful as documentation, but they are not required by Peppercorn.

The data structure returned from *`peppercorn.parse()`* will always be a mapping.

To use Peppercorn in a web application, create a form that has the tokens in order. For instance, the below form will generate the above token stream:

```html
<form action="." method="post" enctype="multipart/form-data">
  <input name="name"/>
  <input name="title"/>
  <input type="hidden" name="__start__" value="series:mapping"/>
  <input name="name"/>
  <input type="hidden" name="__start__" value="dates:sequence"/>
  <input type="hidden" name="__start__" value="date:sequence"/>
  <input name="day"/>
  <input name="month"/>
  <input name="year"/>
  <input type="hidden" name="__end__"/>
  <input type="hidden" name="__start__" value="date:sequence"/>
  <input name="day"/>
  <input name="month"/>
  <input name="year"/>
  <input type="hidden" name="__start__" value="sex:rename"/>
  <input type="radio" name="sex1" value="male"/>
  <input type="radio" name="sex1" value="female"/>
  <input type="hidden" name="__end__"/>
  <input type="hidden" name="__end__"/>
  <input type="hidden" name="__end__"/>
  <input type="hidden" name="__end__"/>
</form>
```

Then when the web post reaches the application, call the *`peppercorn.parse()`* function with the ordered field pairs. For a *WebOb* request, this means using the `items` method of a Webob MultiDict such as `request.POST`:

```python
fields = request.POST.items()
peppercorn.parse(fields)
```

The `list` attribute of a Python `cgi.FieldStorage` object can also be used as a source of information:

```python
fields = []
if fs.list:
    for field in fs.list:
        if field.filename:
            fields.append((field.name, field))
        else:
            fields.append((field.name, field.value))
```

(continues on next page)

(continued from previous page)

```

peppercorn.parse(fields)
```

CHAPTER 1

API Documentation

`peppercorn.`**`parse`**`(`*tokens*`)`

Infer a data structure from the ordered set of fields and return it.

CHAPTER 2

Glossary

WebOb WebOb is a WSGI request/response library created by Ian Bicking.

CHAPTER 3

Resources

See the plope.com article about the genesis of Peppercorn.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index