Merging structured data with \textit{jsonmerge}

Tomaž Šolc
tomaz.solc@tablix.org

Python meetup, Ljubljana, 20 July 2017
About me

- I'm an electrical engineer (Univ. of Ljubljana)
  - Currently a research assistant at JSI
- I've been using Python since around 2007
  - These days mostly for numerical simulations and visualizations (numpy, matplotlib)
- In previous life I was doing semantic web, natural language processing, linked data.
- I sometimes help with opendata.si projects.
Structured what now?

- In short: data you can serialize to JSON format.

```python
>>> data = {
    "title": "Alice's adventures in Wonderland",
    "publication": {
        "year": 1865,
    },
    "characters": ["Alice", "White rabbit"]
}
```
A more complicated structure
This Python module allows you to merge a series of JSON documents into a single one.

Supports Python 2.7 and 3.x
- dependencies: jsonschema

pip install jsonmerge

https://github.com/avian2/jsonmerge
- 1.4.0 released on 5 June 2017
- MIT license
Why is this useful?

- Gathering data about an object from different sources
  - For example, data from web scraping
  - Structured open data from different organizations (e.g. https://github.com/open-contracting)

- Data that gets updated over time
  - Merging changes from different authors
  - Recording how a document changed over time
  - jsonmerge borrows some terminology from git
Merging dicts with stdlib

```python
>>> base = {
    "publication": { "year": 1865 },
    "title": "Alice's adventures in Wonderland",
    "characters": [ "Alice" ]
}

>>> head = {
    "publication": { "publisher": "Macmillan" },
    "characters": [ "White rabbit" ],
    "author": "Lewis Carroll"
}

>>> base.update(head)

>>> base
{'author': 'Lewis Carroll',
 'characters': ['White rabbit'],
 'publication': {'publisher': 'Macmillan'},
 'title': "Alice's adventures in Wonderland"}
```
Merging with jsonmerge

```python
from jsonmerge import merge
base = merge(base, head)
print(base)

{'author': 'Lewis Carroll',
 'characters': ['White rabbit'],
 'publication': {
 'publisher': 'Macmillan',
 'year': 1865
 },
 'title': "Alice's adventures in Wonderland"}
```
Specifying a schema

```python
>>> schema = {
    "properties": {
        "characters": {
            "mergeStrategy": "append"
        }
    }
}
>>> base = merge(base, head, schema)
>>> base
{'author': 'Lewis Carroll',
 'characters': ['Alice', 'White rabbit'],
 'publication': {
     'publisher': 'Macmillan',
     'year': 1865
 },
 'title': "Alice's adventures in Wonderland"
}
```
But I thought JSON is schema-less?

- JSON can be very XML-ish if you want it to be
- JSON schema ≈ XML schema
  - [http://json-schema.org](http://json-schema.org)
  - Python validator: [https://github.com/Julian/jsonschema](https://github.com/Julian/jsonschema)
  - jsonmerge extends JSON schema with the "mergeStrategy" keyword
- JSON pointer (RFC 6901) ≈ XPath
  - (including fun parts like external references...)
- JSON merge patch (RFC 7386) ≈ XML Patch
  - not related to jsonmerge and serves a different purpose
Example JSON schema

```python
>>> schema = {
    "type": "object",
    "properties": {
        "characters": {
            "type": "array",
            "elements": {
                "type": "string"
            }
        },
        "title": {
            "type": "string"
        },
        "publication": {
            "properties": {
                "year": {
                    "type": "number"
                }
            }
        }
    }
}
```
Deeply nested structures

- **Without a schema**, `jsonmerge` will
  - recursively merge objects,
  - replace other types with newer values.

- **With a schema**, various merge strategies can be defined in (very) complex ways.
  - Each part of the document can be merged in a different way.
  - `jsonmerge` can automatically generate a validation schema for the resulting document.
Merge strategies

- **Built-ins**
  - overwrite
  - append
  - arrayMergeById (treat arrays as sets)
  - objectMerge (recursive "dict.update")
  - version (new values are appended to an array)

- **Add your own by subclassing MergeStrategy**
Example: versioning

```python
>>> schema = { "type": "string",
             "mergeStrategy": "version" }
>>> from jsonmerge import Merger
>>> merger = Merger(schema)

>>> base = None
>>> base = merger.merge(base, "Hello, World!",
                      meta={"version": 1})
>>> base = merger.merge(base, "Howdy, World!",
                      meta={"version": 2})

>>> base
[{
   "value": "Hello, World!", "version": 1},
  {
   "value": "Howdy, World!", "version": 2}
]```
Example: get_schema()

```python
click
>>> base

[{
    "value": "Hello, World!", "version": 1},
    {
    "value": "Howdy, World!", "version": 2}
]

>>> merged_schema = merger.get_schema()

>>> merged_schema

{
    'type': 'array',
    'items': {
        'properties': {
            'value': {
                'type': 'string'
            }
        }
    }
}

>>> import jsonschema

>>> jsonschema.validate(base, merged_schema)
```
Example: array as a set

```python
>>> schema = {
    "mergeStrategy": "arrayMergeById",
    "mergeOptions": {"idRef": "/"},
}

>>> from jsonmerge import Merger
>>> merger = Merger(schema)

>>> base = [1, 2]
>>> head = [2, 3]
>>> merger.merge(base, head)
[1, 2, 3]
```
Example: all arrays with appends

```
{
    "oneOf": [
        {
            "type": "array",
            "mergeStrategy": "append"
        },
        {
            "type": "string"
        },
        {
            "type": "object",
            "additionalProperties": {
                "$ref": "#"
            }
        }
    ]
}
```
Questions?

tomaz.solc@tablix.org

https://github.com/avian2/jsonmerge