JSONTas Documentation

Release unknown

Oct 15, 2021

Contents

JSONTas is a tool for generating dynamic JSON structures.

CHAPTER 1

Contents

1.1 JSONTas

JSONTas is a tool for generating dynamic JSON data.

1.1.1 Description

JSONTas adds conditionals and logic to JSON files in order to create dynamic JSON data depending on which dataset you supply.

It opens up the possibility to create generic tools where most of the operations are done by executing JSONTas on the data.

Documentation: https://jsontas.readthedocs.io/en/latest

1.1.2 Features

- Simple yet powerful syntax.
- HTTP requests on parse
- Separation of environments by providing different datasets.

1.1.3 Installation

Install the project by running:

```
pip install jsontas
```

1.1.4 Examples

First we create two datasets. One for our 'dev' environment and one for our 'prod' environment.

Dataset 'dev.json'

```
{
   "mode": "dev",
   "database": "dev_db"
}
```

Dataset 'prod.json'

```
{
   "mode": "prod",
   "database": "prod_db"
}
```

JSONTas JSON file

Next up, let's create our JSONTas file.

```
{
   "database": {
      "host": "myawesomedb.example.com",
      "database": "$database"
   },
   "message": {
      "$condition": {
         "if": {
            "key": "$mode",
            "operator": "$eq",
            "value": "dev"
         },
         "then": "This is the DEV server.",
         "else": "This is the PROD server."
      }
   }
}
```

JSONTas execute with 'dev' dataset

```
jsontas -d dev.json data.json
```

```
{
   "database": {
      "host": "myawesomedb.example.com",
      "database": "dev_db"
   },
   "message": "This is the DEV server."
}
```

JSONTas execute with 'prod' dataset

```
jsontas -d prod.json data.json
```

```
{
   "database": {
      "host": "myawesomedb.example.com",
      "database": "prod_db"
   },
   "message": "This is the PROD server."
}
```

These examples only show the bare minimum. For more examples look at our documentation at: https://jsontas.readthedocs.io/en/latest

1.1.5 Contribute

- Issue Tracker: https://github.com/AxisCommunications/jsontas/issues
- Source Code: https://github.com/AxisCommunications/jsontas

1.1.6 Support

If you are having issues, please let us know. Email tobias.persson@axis.com or just write an issue.

1.2 Examples

In order to run these examples you need to create two JSON-files; one for the dataset and one for the actual JSONTas JSON file.

Run the examples by executing

```
jsontas -d dataset.json jsonfile.json
```

1.2.1 Condition

jsontas.data_structures.condition

Set value in JSON on a condition. Supported operators are defined here: *jsontas.data_structures.operator.Operator*

Dataset

```
{
  "name": "John Doe"
}
```

JSON

```
{
   "occupation": {
      "$condition": {
         "if": {
             "key": "$name",
             "operator": "$eq",
             "value": "John Doe",
         },
         "then": "Engineer",
         "else": "Unemployed"
      }
   }
}
```

Result

```
{
   "occupation": "Engineer"
}
```

1.2.2 Condition List

jsontas.data_structures.condition

Set a value in JSON on multiple conditions. Supported operators are defined here: *jsontas.data_structures.operator.Operator*

Dataset

```
{
   "name": "John Doe",
   "occupation": "Engineer"
}
```

JSON

```
{
   "team": {
      "$condition": {
          "if": [
              {
                  "key": "$name",
                  "operator": "$eq",
                  "value": "John Doe",
              },
              {
                  "key": "$occupation",
                  "operator": "$in",
                  "value": ["Engineer", "Manager"]
```

(continues on next page)

(continued from previous page)

```
                }
            ],
            "then": "The Best Team",
            "else": "The Worst Team"
        }
    }
}
```

Result

```
{
    "team": "The Best Team"
}
```

Note that condition list is an 'AND' check so all conditions must be True for the 'then' field to execute.

1.2.3 Expand

jsontas.data_structures.expand

Expand a value into a list of a certain number of elements.

Dataset

```
{
    "likes": 2,
    "upvotes": 3
}
```

JSON

```
{
    "upvotes": {
        "$expand": {
            "value": {
                "upvote": true
            },
            "to": "$upvotes"
        }
    },
    "likes": {
        "$expand": {
            "value": "Like",
            "to": "$likes"
        }
    }
}
```

Result

```
{
   "upvotes": [
      {"upvote": true},
      {"upvote": true},
      {"upvote": true}
   ],
   "likes": ["Like", "Like"]
}
```

1.2.4 Filter

jsontas.data_structures.filter

Remove items that do not match a certain filter. Supported operators are defined here: *jsontas.data_structures.operator.Operator*

Dataset

```
{
   "employees": [
      {
         "name": "John Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Smith",
         "occupation": "Manager"
      }
   ]
}
```

JSON

```
{
   "engineers": {
      "$filter": {
         "items": "$employees",
         "filters": [
            {
               "key": "occupation",
               "operator": "$eq",
               "value": "Engineer"
            }
         ]
      }
   }
}
```

Result

```
{
   "engineers": [
      {
         "name": "John Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Doe",
         "occupation": "Engineer"
      }
   ]
}
```

1.2.5 From

jsontas.data_structures.from_item

Get a value from a dictionary.

Dataset

```
{
   "manager": {
      "name": "Jane Smith",
      "occupation": "Manager"
   }
}
```

JSON

```
{
   "manager": {
      "$from": {
         "item": "$manager",
         "get": "name"
      }
   }
}
```

Result

```
{
   "manager": "Jane Smith"
}
```

1.2.6 List

jsontas.data_structures.list

While List is not supposed to be used directly inside a JSON structure one can operate on lists in a dataset like this

Dataset

```
{
   "employees": [
      {
         "name": "John Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Smith",
         "occupation": "Manager"
      }
   ]
}
```

JSON

```
{
   "first_employee": "$employees.0",
   "last_employee": "$employees.-1",
   "first_two_employees": "$employees.:2"
}
```

Result

```
{
   "first_employee": {
      "name": "John Doe",
      "occupation": "Engineer"
   },
   "last_employee": {
      "name": "Jane Smith",
      "occupation": "Manager"
   },
   "first_two_employees": [
      {
         "name": "John Doe",
         "occupation": "Engineer"
      },
      {
         "name": "Jane Doe",
         "occupation": "Engineer"
      }
   ]
}
```

1.2.7 Operator

jsontas.data_structures.operator

Dataset

```
{
   "employee": {
      "name": "Jane Doe",
      "occupation": "Engineer"
   }
}
```

JSON

```
{
   "is_manager": {
      "$operator": {
         "key": "$employee.occupation",
         "operator": "$eq",
         "value": "Manager"
      }
   }
}
```

Result

```
{
   "is_manager": false
}
```

Available operators are further explained in *jsontas.data_structures.operator* They are:

- **$eq** *jsontas.data_structures.operator.Operator._equal*
- **$in** *jsontas.data_structures.operator.Operator._in*
- **$notin** *jsontas.data_structures.operator.Operator._notin*
- **$startswith** *jsontas.data_structures.operator.Operator._startswith*
- **$regex** *jsontas.data_structures.operator.Operator._regex*

1.2.8 Reduce

jsontas.data_structures.reduce

Reduce a list from end to beginning (from right) to a specific value.

Dataset

```
{
   "max_list_length": 2
}
```

JSON

```
{
   "reduced_list": {
      "$reduce": {
         "list": [
            "element 1",
            "element 2",
            "element 3"
         ],
         "to": "$max_list_length"
      }
   }
}
```

Result

```
{
   "reduced_list": [
         "element 1",
         "element 2"
   ]
}
```

1.2.9 Request

jsontas.data_structures.request

Make HTTP requests and get JSON values from the response. Useful for when the dataset is located on a website or if one wants to parse JSON based APIs.

Dataset

```
{
   "userdata": "https://jsonplaceholder.typicode.com/users/1"
}
```

JSON

```
{
   "user": {
      "$request": {
         "url": "$userdata",
         "method": "GET"
      }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Result


```
{
   "user": {
     "address": {
         "city": "Gwenborough",
         "geo": {
             "lat": "-37.3159",
             "lng": "81.1496"
         },
         "street": "Kulas Light",
         "suite": "Apt. 556",
         "zipcode": "92998-3874"
     },
     "company": {
         "bs": "harness real-time e-markets",
         "catchPhrase": "Multi-layered client-server neural-net",
         "name": "Romaguera-Crona"
     },
     "email": "Sincere@april.biz",
     "id": 1,
     "name": "Leanne Graham",
     "phone": "1-770-736-8031 x56442",
     "username": "Bret",
     "website": "hildegard.org"
   }
}
```


Getting a specific response from the response will be further explained below in segment *Nested*

1.2.10 Wait

jsontas.data_structures.wait

Waiting for a query tree to evaluate to true. A query tree is the full, uresolved, JSON structure that is currently being resolved. This is mostly used for when utilizing the *Request* datastructure as a way to wait for an API to respond with the data required.

Example will look similar to the *Request* example as this is how the *Wait* structure is normally used.

Dataset

```
{
   "userdata": "https://jsonplaceholder.typicode.com/users/this_does_not_exist"
}
```

JSON

```
{
   "user": {
      "$wait": {
         "for": {
            "$request": {
               "url": "$userdata",
               "method": "GET"
            }
          },
          "interval": 1,
          "timeout": 5,
          "else": "No user found"
      }
   }
}
```

Result

Result will come after 5 seconds.

```
{
   "user": "No user found"
}
```

1.2.11 Nested

Now that we know how all the data structures work in isolation we can start nesting data structures and create more advanced logic.

This example will get the title of all the posts from the user 'Leanne Graham' at https://jsonplaceholder.typicode.com/users

Dataset

```
{
   "users_api": "https://jsonplaceholder.typicode.com/users",
   "posts_api": "https://jsonplaceholder.typicode.com/posts",
   "username": "Leanne Graham",
   "accepted_status_codes": [200]
}
```

JSON

```
{
   "user_id": {
      "$from": {
         "item": {
            "$filter": {
               "items": {
```

(continues on next page)

(continued from previous page)

```
                    "$wait": {
                        "for": {
                            "$condition": {
                                "then": {
                                    "$request": {
                                        "url": "$users_api",
                                        "method": "GET"
                                    }
                                },
                                "if": {
                                    "key": "$response.status_code",
                                    "operator": "$in",
                                    "value": "$accepted_status_codes"
                                },
                                "else": null
                            }
                        },
                        "interval": 1,
                        "timeout": 20,
                        "else": {}
                    }
                },
                "filters": [
                    {
                        "key": "name",
                        "operator": "$eq",
                        "value": "$username"
                    }
                ]
            }
        },
        "get": "id"
    }
},
"posts": {
    "$from": {
        "item": {
            "$request": {
                "url": "$posts_api",
                "method": "GET",
                "params": {
                    "userId": "$this.user_id.0"
                }
            }
        },
        "get": "title"
    }
}
}
```

Result

```
{
   "posts": [
      "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
```

(continues on next page)

(continued from previous page)

```
      "qui est esse",
      "ea molestias quasi exercitationem repellat qui ipsa sit aut",
      "eum et est occaecati",
      "nesciunt quas odio",
      "dolorem eum magni eos aperiam quia",
      "magnam facilis autem",
      "dolorem dolore est ipsam",
      "nesciunt iure omnis dolorem tempora et accusantium",
      "optio molestias id quia eum"
   ],
   "user_id": [
       1
   ]
}
```

1.2.12 Conclusion

There are many crazy ways of utilizing JSONTas and it's quite impossible to write examples for each and every use-case. If there are any questions then please to not hestitate contacting the maintainers or writing an issue in github. We encurage you to play around with it and send a PR with new examples.

1.3 License

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.
```

(continues on next page)

(continued from previous page)

```
   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
```

(continues on next page)

(continued from previous page)

```
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
```

(continues on next page)

(continued from previous page)

```
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
```

(continues on next page)

(continued from previous page)

```
limitations under the License.
```

1.4 Maintainers

- Tobias Persson <tobias.persson@axis.com>
- Fredrik Fristedt <fredrik.fristedt@axis.com>

1.5 Contributing

Thank you for showing interest in improving JSONTas, it is appreciated!

1.5.1 Proposing changes

Anyone can propose changes by just writing an issue in GitHub. These proposals will be reviewed by our maintainers in order to make sure that it's feasable.

When writing proposals for JSONTas, be as precise as possible and include arguments as to why this change is beneficial and possible downsides.

Note that our maintainers will implement submitted proposals in due time and that you are welcome to implement the proposal yourself for faster integration. During implementation, it is required to assign yourself to the issue and following the contribution guide below.

1.5.2 How to contribute

Make contributions by forking the project, make the change and create a pull request. Note that it is required to link an issue in the pull request unless it is a very minor change (i.e. spelling corrections etc.).

Every pull request is appreciated but note that all changes might not be accepted. If this is the case, the reasons shall be communicated in the pull request. If there is no communication or the communicated reasons are not clear enough, do not hesitate to contact the maintainers at jsontas-maintainers@google-groups.com

License

All contributions must be licensed under the Apache License 2.0. This means that a license notice shall be included in every file in the following format:

> Copyright <Year(s)> <Copyright holder of original contribution [and others].>
>
> For a full list of idividual contributors, please see the commit history.

1.6 Contributor Covenant Code of Conduct

1.6.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size,

disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

1.6.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

1.6.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

1.6.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

1.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at oss-conduct@axis.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

1.6.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.7 Changelog

1.7.1 Version 1.0.0

- JSONTas is a tool for generating dynamic JSON structures.

1.8 jsontas

1.8.1 jsontas package

Subpackages

jsontas.data_structures package

Submodules

jsontas.data_structures.condition module

Condition datastructure.

class jsontas.data_structures.condition.**Condition**(*jsonkey*, *datasubset*, *dataset*, ***data*)

Bases: *jsontas.data_structures.datastructure.DataStructure*

Condition datastructure.

Example:

```
{
    "$condition": {
        "if": {
            "key": "Key to match",
            "operator": "$eq",
            "value": "something",
        },
        "then": "returnvalue",
        "else": "Default value"
    }
}
```

Example:

```
{
    "$condition": {
        "if": [
```

(continues on next page)

(continued from previous page)

```
            {
                "key": "Key to match",
                "operator": "$eq",
                "value": "something",
            },
            {
                "key": "Another key to match",
                "operator": "$in",
                "value": "somethingelse"
            }
        ],
        "then": "returnvalue",
        "else": "Default value"
    }
}
```

Supported operators defined here: *jsontas.data_structures.operator.Operator*

_if(*operator*)
If operator.

Parameters **operator** (*dict*) – Data to check "if" on.

Returns Operator result.

Return type bool

static _else(*data*)
Else operator. Just return the data.

Note that this method is added for readability in *execute* only.

Parameters **data** (*any*) – Data to just return.

Returns Condition.

Return type str

execute()
Execute data.

Returns None and value from either 'else' or 'then'.

Return type tuple

jsontas.data_structures.datastructure module

Base datastructure.

class jsontas.data_structures.datastructure.**DataStructure**(*jsonkey*, *datasubset*, *dataset*, ***data*)

Bases: object

Base datastructure class.

execute()
Execute datastructure.

Implement this.

jsontas.data_structures.expand module

Expand datastructure.

class jsontas.data_structures.expand.**Expand**(*jsonkey*, *datasubset*, *dataset*, ***data*)

Bases: *jsontas.data_structures.datastructure.DataStructure*

Expand datastructure.

Expand a value into a list of a certain number of elements.

Example:

```
{
    "a_list": {
        "$expand": {
            "value": {
                "hello": "world"
            },
            "to": 5
        }
    }
}
```

Result:

```
{
    "a_list": [
        {
            "hello": "world"
        },
        {
            "hello": "world"
        },
        {
            "hello": "world"
        },
        {
            "hello": "world"
        },
        {
            "hello": "world"
        }
    ]
}
```

Example:

```
{
    "another_list": {
        "$expand": {
            "value": "something"
            "to": 5
        }
    }
}
```

Result:

```
{
    "another_list": [
        "something",
        "something",
        "something",
        "something",
        "something"
    ]
}
```

execute()
Execute expand.

Returns None and a list of values.

Return type tuple

jsontas.data_structures.filter module

Filter datastructure.

class jsontas.data_structures.filter.**Filter**(*jsonkey*, *datasubset*, *dataset*, ***data*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

Filter datastructure.

Example:

```
{
    "data": {
        "$filter": {
            "items": [
                {
                    "status": "success",
                    "value": "1"
                },
                {
                    "status": "failure",
                    "value": "2"
                },
                {
                    "status": "success",
                    "value": "3"
                }
            ],
            "filters": [
                {
                    "key": "status",
                    "operator": "$eq",
                    "value": "success"
                }
            ]
        }
    }
}
# {"data": [{"status": "success", "value": "1"}, {"status": "success", "value": "3
↪"}]}
```

execute()
Execute the filter datastructure.

Returns Key and the value(s) found.

Return type tuple

filter(*item*)
Execute the filtering list against item.

The filtering list is the list of key,operator,value dictionaries following the format of *jsontas.data_structures.operator.Operator* (in fact, the *jsontas.data_structures.operator.Operator* datastructure is used in this method to validate the filters)

Note that when running the filter, the dataset is appended with an "item" which is a single item from the "items" filter list (can be seen above).

The dataset could then be this:

```
{
    **current_dataset,
    "item": { "status": "success", "value": "1"}
}
```

With the filter being:

```
{
    "key": "status",
    "operator": "$eq",
    "value": "success"
}
```

And the filter would be True (the status of the item is 'success'). Note that this method will only ever run on the "item". If there is not "$item" in the "key", the method will add it. That is why it's possible to write:

```
{"key": "status"}
```

instead of:

```
{"key": "$item.status"}
```

But it would also mean that if one were to write:

```
{"key": "$anotherkey.something.else"}
```

it would become:

```
{"key": "$item.$anotherkey.something.else"}
```

which would fail.

This is a design choice. The filter shall not utilize any data aside from the 'item' key in the dataset.

Parameters **item** (*dict*) – Item to filter.

Returns Whether or not all items in filter evaluates to True.

Return type bool

jsontas.data_structures.from_item module

From datastructure.

class jsontas.data_structures.from_item.**From**(*jsonkey*, *datasubset*, *dataset*, ***data*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

From datastructure.

Example:

```
{
    "hello" {
        "$from": {
            "item": {
                "value": "something",
                "text": "world"
            },
            "get": "text"
        }
    }
}
# {"hello": "world"}
```

execute()
Execute the $from datastructure.

Returns Key and the value(s) found.

Return type tuple

jsontas.data_structures.list module

List datastructure.

class jsontas.data_structures.list.**List**(*jsonkey*, *datasubset*, *dataset*, ***data*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

List datastructure.

Not to be used directly in JSON structure. Use it with the .int operator.

index - Index 0:

```
{
    "something": "$value.0"
}
```

index - Last index:

```
{
    "something": "$value.-1"
}
```

slice - Everything after index 1:

```
{
    "something": "$value.1:"
}
```

slice - Everything before index 4:

```
{
    "something": "$value.:4"
}
```

slice - Everything between index 2 and 4:

```
{
    "something": "$value.2:4"
}
```

execute()
Execute the list datastructure.

Returns Key and the value(s) found.

Return type tuple

static index(*data*, *index*)
Get an index from data.

Parameters

- **data** (*list, set or tuple*) – Data to get index from.
- **index** (*int*) – Index to get.

Returns Value on index.

Return type any

static slice(*data*, *first*, *second*)
Slice a string based on first and second integers.

Parameters

- **data** (*list, set or tuple*) – Data to get slice from.
- **first** (*int or None*) – Number to put before ':' if not None.
- **second** (*int or None.*) – Number to put after ':' if not None.

Returns Sliced list.

Return type list

static split(*value*)
Split a string on ':' and return first and second value.

Parameters **value** (*str*) – Value to split.

Returns First and second value. Any of them can be None.

Return type tuple

jsontas.data_structures.operator module

Operator datastructure.

class jsontas.data_structures.operator.**Operator**(**args*, ***kwargs*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

Operator datastructure.

Example:

```
{
    "$operator": {
        "key": "key to match",
        "operator": "$eq",
        "value": "value to match against key"
    }
}
```

Supported operators:

- **$eq** *_equal*
- **$in** *_in*
- **$notin** *_notin*
- **$startswith** *_startswith*
- **$regex** *_regex*

_equal()
Operator '==='.

Example:

```
{
    "$operator": {
        "key": "key to match",
        "operator": "$eq",
        "value": "value to match against key"
    }
}
```

_in()
Operator 'in'.

Example - In list:

```
{
    "$operator": {
        "key": "key",
        "operator": "$in",
        "value": ["key", "anotherkey"]
    }
}
```

Example - In string:

```
{
    "$operator": {
        "key": "k",
        "operator": "$in",
        "value": "key"
    }
}
```

_notin()
Operator 'not in'.

Example - In list:

```
{
    "$operator": {
        "key": "key",
        "operator": "$notin",
        "value": ["key", "anotherkey"]
    }
}
```

Example - In string:

```
{
    "$operator": {
        "key": "k",
        "operator": "$notin",
        "value": "key"
    }
}
```

_startswith()
Operator 'str.startswith()'.

Example:

```
{
    "$operator": {
        "key": "key",
        "operator": "$startswith",
        "value": "k"
    }
}
```

_regex()
Operator 're.match'.

Example:

```
{
    "$operator": {
        "key": "key",
        "operator": "$regex",
        "value": "^[A-Za-z]+$"
    }
}
```

execute()
Execute operator.

Returns None and whether key matches value or not.

Return type tuple

jsontas.data_structures.reduce module

Reduce datastructure.

class jsontas.data_structures.reduce.**Reduce**(*jsonkey*, *datasubset*, *dataset*, ***data*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

Reduce a list from end to beginning (from right) to a specific value

Example:

```
{
    "reduced_list": {
        "$reduce": {
            "list": [
                "element 1",
                "element 2",
                "element 3"
            ],
            "to": 2
        }
    }
}
```

Result:

```
{
    "reduced_list": [
        "element 1",
        "element 2"
    ]
}
```

execute()
Execute reduce.

Returns None and a list of values.

Return type tuple

jsontas.data_structures.request module

Request datastructure.

class jsontas.data_structures.request.**Request**(*jsonkey*, *datasubset*, *dataset*, ***data*)
Bases: *jsontas.data_structures.datastructure.DataStructure*

HTTP request datastructure.

Example:

```
{
    "$request" {
        "url": "http://localhost:8000/something.json",
        "method": "GET"
    }
}
```

Example:

```
{
    "$request" {
        "url": "http://localhost:8000/something.json",
        "method": "POST",
        "json": {
```

(continues on next page)

(continued from previous page)

```
            "my_json": "hello"
        },
        "headers": {
            "Content-Type": "application/json"
        },
        "auth": {
            "username": "admin",
            "password": "admin",
            "type": "basic"
        }
    }
}
```

Example getting response after:

```
# Assume response from request is: {"hello": "world"}
{
    "data": {
        "$request" {
            "url": "http://localhost:8000/something.json",
            "method": "GET"
        }
    },
    "text": "$response.json.hello"
}
# Resulting JSON will be:
{
    "data": {
        "hello": "world"
    },
    "text": "world"
}
```

execute()
Execute data.

Returns None and response as JSON (or None).

Return type Tuple

request(*url*, *method*, *json=None*, *headers=None*, ***requests_parameters*)
Make an HTTP request.

Parameters

- **url** (*str*) – URL to request.
- **method** (*str*) – HTTP method.
- **json** (*dict*) – Optional JSON data to request.
- **headers** (*dict*) – Optional extra headers to request.
- **requests_parameters** (*dict*) – Extra parameters to python requests.

Returns Wait generator for getting responses from request.

Return type generator

static wait(*method*, *timeout=None*, *interval=5*, ***kwargs*)
Iterate over result from method call.

Parameters

- **method** – Method to call.
- **timeout** (*int or None*) – How long, in seconds, to iterate.
- **interval** (*int*) – How long, in seconds, to wait between method calls.
- **kwargs** (*dict*) – Keyword arguments to pass to method call.

jsontas.data_structures.wait module

Wait datastructure.

class jsontas.data_structures.wait.**Wait**(*jsonkey*, *datasubset*, *dataset*, ***data*)

Bases: *jsontas.data_structures.datastructure.DataStructure*

Wait datastructure.

Wait for a query tree to evaluate to True.

Example:

```
{
    "$wait": {
        "for": {
            "$request": {
                "url": "http://example.com",
                "method": "GET"
            }
        },
        "interval": 1,
        "timeout": 20,
        "else": {}
    }
}
```

Request example.com until a non-null response. Maximum 1 request/s for 20s

Due to the storage of the 'query_tree' in dataset it is possible to nest queries:

Example:

```
{
    "response": {
        "$from": {
            "item": {
                "$wait": {
                    "for": {
                        "$condition": {
                            "then": {
                                "$request": {
                                    "url": "http://example.com",
                                    "method": "GET"
                                }
                            },
                            "if": {
                                "key": "$response.status_code",
                                "operator": "$eq",
                                "value": 200
```

(continues on next page)

(continued from previous page)

```
                            },
                            "else": null
                        }
                    },
                    "interval": 1,
                    "timeout": 20,
                    "else": {}
                }
            },
            "get": "items"
        }
    }
}
```

Wait for example.com to respond with status_code 200 and a non-null response and get the 'items' key from the response. Maximum 1 request/s for 20s

execute()
Execute wait datastructure.

Waiting for will requires a 'query_tree' which is a collection of all requests that are 'below' the 'wait' datastructure. This is to keep track of the queries that have executed before (but not the results of these queries).

This query tree is then executed in the *jsontas.jsontas.JsonTas.resolve()* method, which can be considered weird. We're executing JsonTas inside of a JsonTas query. But this is the only way to re-run a query tree.

Returns None and the result of re-running a query tree.

Return type Tuple

static wait(*method*, *timeout*, *interval*, ***kwargs*)
Iterate over result from method call.

Parameters

- **method** – Method to call.
- **timeout** (*int or None*) – How long, in seconds, to iterate.
- **interval** (*int*) – How long, in seconds, to wait between method calls.
- **kwargs** (*dict*) – Keyword arguments to pass to method call.

Module contents

Datastructure.

Submodules

jsontas.dataset module

Dataset module.

class jsontas.dataset.**Dataset**
Bases: object

JSONTas dataset object. Used for lookup of $ notated strings in a JSON file.

add(*key*, *value*)
Add a new dataset value and key.

Parameters

- **key** (*str*) – Dictionary key for global dataset dict.
- **value** (*any*) – Dictionary value for global dataset dict.

copy()
Make a copy of this dataset.

Returns A dataset object with a copy of the internal dataset in it.

Return type *Dataset*

get(*key*, *default=None*)
Get a key from dataset global dictionary.

Parameters

- **key** (*any*) – Key to get from global dataset dictionary.
- **default** (*any*) – Default value if the key does not exist.

Returns Value from key in global dataset dictionary or default.

Return type any

get_or_getattr(*datasubset*, *key*)
Get a key from a datasubset. Either using 'get' or 'getattr'.

Raises AttributeError if attribute could not be found with either 'get' or 'getattr'.

Parameters

- **datasubset** (*any*) – Dataset to attempt to get key from.
- **key** (*any*) – Key to get.

Returns Value from key.

Return type any

logger = <Logger Dataset (WARNING)>

lookup(*query_string*, *parameters*)
Lookup JSONTas query string against dataset.

Query string is a dot separated string of keywords which is split and iterated over.

If the query string is "$something.another.third":

```
# Iterate through the words
["something", "another", "third"]
```

These keywords are then matched against the dataset very simply

1. Try to get keyword from dataset, either via ".get" or "getattr".
2. If keyword exists, inspect what the type of the keyword value is.
3. If it's a *jsontas.data_structures.datastructure.DataStructure*, call its 'execute' method with the parameters and keyword value.
4. If it's a function, call the function with the parameters and keyword value.
5. If neither of the above are True, set dataset to the value.

6. Loop to the next keyword.

If at any point of this, the keyword value becomes None, return without changing the key. (the key is, by default, the query_string)

If the keyword value is not None after the loop is finished, return the key and value. Note that the *jsontas.data_structures.datastructure.DataStructure* and functions must return a key, value pair.

Value will be put as the 'value' in the JSON dict when returned, key is what the key will be changed to.

Assume the following dataset for the JSON below:

```
{
    "query": {
        "call": "Chaos"
    }
}
```

Input:

```
{
    "Text": "$query.call"
}
```

Output:

```
{
    "Text": "Chaos"
}
```

In most cases, as well as in these examples, the key returned is None. It is expected that the caller handles this situation by setting the correct key when None is returned.

The "correct" key is generally the key that was already there. In the examples above the the key is "Text", and this method returns None, so JSONTas will keep the key as 'Text'

However, if the key returned is not None, the key is overwritten by JSONTas. This is not a very common use-case, but it is possible to implement in a *jsontas.data_structures.datastructure.DataStructure*

This should be used with care! In fact, the only built-in use-case for this behavior is when there's an exception in the lookup or the keyword value becomes None at any point in order to have the key stay as the query_string in case of problems.

Parameters

- **query_string** (*str*) – JSONTas query string.
- **parameters** (*any*) – Parameters that exist nested below the query_string in JSON data. For instance {"$querystring": {"some": "data"}} parameters would be {"some": "data"}

Returns New key and value as defined by dataset.

Return type tuple

merge(*dataset*)

Merge a dataset with this dataset.

Parameters **dataset** (*dict*) – Dataset to merge with this one.

regex = re.compile('[\\$\\-\\w!,:]+')

jsontas.jsontas module

JSONTas module.

class jsontas.jsontas.**JsonTas**(*dataset=None*)
Bases: object

JSONTas resolver.

logger = <Logger JSONTas (WARNING)>

resolve(*json_data*, *query_tree=None*)
Resolve JSONTas queries. Takes a JSON structure and resolve all values against dataset.

This is a recursive method.

Parameters

- **json_data** (*any*) – JSON data to iterate through and resolve.
- **query_tree** (*any*) – Used in recursion to keep track of query_tree.

Returns New JSON structure with resolved values.

Return type any

run(*json_data=None*, *json_file=None*, *copy=True*)
Run JSONTas. This should be the main entry to JSONTas.

Parameters

- **json_data** (*file*) – JSON data to run JSONTas on.
- **json_file** – JSON file to run JSONTas on.

Returns Resolved JSON structure.

Return type OrderedDict

Module contents

Distribution name and version.

Python Module Index

j

Index

Symbols

A

C

D

E

F

G

I

J

L

M

O

R

S

W