JsonPreprocessor

v. 0.2.3

Mai Dinh Nam Son

04.05.2023

Contents

Chapter 1

Introduction

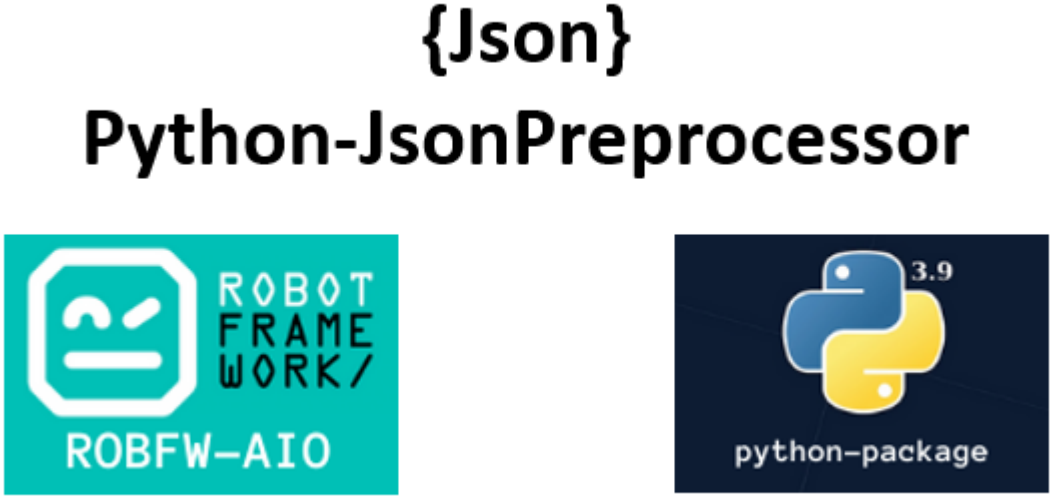
1.1 Json Preprocessor documentation

This is the documentation for Python JsonPreprocessor

Json is a format used to represent data and becomes the universal standard of data exchange. Today many software projects are using configuration file in Json format. For a big or a complex project there is a need to have some enhanced format in Json file such as adding the comments, importing other Json files, etc.

Based on that needs, we develop JsonPreprocessor package:

- Gives the possibility to comment out parts of the content. This feature can be used to explain the meaning of the parameters defined inside the configuration files.
- Has ability to import other Json files. This feature can be applied for complex project, users can create separated Json files then importing them to other Json file.
- Allows users using the defined parameter in Json file.
- Accepts **"True"**, **"False"**, and **"None"** in Json syntax



Chapter 2

Description

2.1 How to install

JsonPreprocessor can be installed in two different ways.

1. Installation via PyPi (recommended for users)

   ```
   pip install JsonPreprocessor
   ```

   JsonPreprocessor in PyPi

2. Installation via GitHub (recommended for developers)
   Clone the **JsonPreprocessor** repository to your machine.

   ```
   git clone https://github.com/test-fullautomation/python-jsonpreprocessor.git
   ```

   JsonPreprocessor in GitHub
   Use the following command to install **JsonPreprocessor**:

   ```
   setup.py install
   ```

2.2 Features

2.2.1 Basic Json format

Users can use JsonPreprocessor to handle the json file with its original format.

Example:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "param_1" : "value_1",
      "param_2" : value_2,
      "structure_param": {
        "general": "general"
      }
    }
```

```
  },
  "device" : "device_name"
}
```

2.2.2 Adding the comments

Often large projects require a lot of configuration parameters. So adding comments to json files is useful in case of more and more content is added, e.g. because of a json file has to hold a huge number of configuration parameters for different features. Comments can be used here to clarify the meaning of these parameters or the differences between them.

Every line starting with "//", is commented out. Therefore a comment is valid for singles lines only.

Comment out a block of several lines with only one start and one end comment string, is currently not supported.

Example:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
{
  "Project": "name_of_prject",
  // <adding comment>
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // <adding comment>
    "global": {
      "param_1" : "value_1",
      "param_2" : value_2,  // <adding comment>
      "structure_param": {
        "general": "general"
      }
    }
  },
  "device" : "device_name"
}
```

2.2.3 Imports other json files

This import feature enables developers to take over the content of other json files into the current json file. A json file that is imported into another json file, can contain imports also (allows nested imports).

A possible usecase for nested imports is to handle configuration parameters of different variants of a feature or a component within a bunch of several smaller files, instead of putting all parameter into only one large json file.

Example:

Suppose we have the json file `params_global.json` with the content:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
//
```

```
//  This is to distinguish the different types of resets
{
  "import_param_1" : "value_1",

  "import_param_2" : "value_2",

  "import_structure_1": {   // <adding comment>
    "general": "general"
  }
}
```

And other json file `preprocessor_definitions.json` with content:

```
//*****************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*****************************************************************************
{
  "import_param_3" : "value_3",

  "import_param_4" : "value_4",

  // <adding comment>

  "import_structure_2": {
     "general": "general"
   }
}
```

Then we can import these 2 files above to the json file `config.json` with the [import] statement:

```
//*****************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*****************************************************************************
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
           "[import]": "<path_to_the_imported_file>/params_global.json"
     }
   },
  "preprocessor": {
    "definitions": {
        "[import]": "<path_to_the_imported_file>/preprocessor_definitions.json"
      }
  },
  "device" : "device_name"
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
        }
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
      "import_structure_2": {
         "general": "general"
       }
    }
  },
  "device" : "device_name"
}
```

2.2.4 Add new or overwrites existing parameters

This JsonPreprocessor package also provides developers ability to add new as well as overwrite existing parameters. Developers can update parameters which are already declared and add new parameters or new element into existing parameters. The below example will show the way to do these features.

In case we have many different variants, and each variant requires a different value assigned to the parameter, users can use this feature to add new parameters and update new values for existing parameters of existing configuation object.

Example:

Suppose we have the json file `params_global.json` with the content:

```
//*************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*************************************************************************
//
//  This is to distinguish the different types of resets
{
  "import_param_1" : "value_1",

  "import_param_2" : "value_2",

  "import_structure_1": {   // <adding comment>
    "general": "general"
  }
}
```

Then we import `params_global.json` to json file `config.json` with content:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
       "[import]": "<path_to_the_imported_file>/params_global.json"
      }
    },
  "device" : "device_name",
  // Overwrite parameters
  "${params}['global']['import_param_1']": "new_value_1",
  "${version}['patch']": "2",
  // Add new parameters
  "new_param": {
       "abc": 9,
           "xyz": "new param"
  },
  "${params}['global']['import_structure_1']['new_structure_param']":
↪  "new_structure_value"
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "2"
  },
  "params": {
    "global": {
      "import_param_1" : "new_value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general",
            "new_structure_param": "new_structure_value"
       }
     }
  },
  "device" : "device_name",
  "new_param": {
      "abc": 9,
      "xyz": "new param"
  }
}
```

2.2.5 Using defined parameters

With JsonPreprocessor package, users can also use the defined parameters in Json file. The value of the defined parameter could be called with syntax `${<parameter_name>}`

Example:

Suppose we have the json file `config.json` with the content:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_1",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
       }
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
        "ABC": "param_ABC",
      "import_structure_1": {
         "general": "general"
       }
    }
  },
  "device" : "device_name",
  // Using the defined parameters
  "${params}['global'][${preprocessor}['definitions']['ABC']]": True,
  "${params}['global']['import_param_1']":
↪  ${preprocessor}['definitions']['import_param_4']
}
```

After all imports are resolved by the JsonPreprocessor, this is the resulting of data structure:

```
{
  "Project": "name_of_prject",
  "version": {
    "major": "0",
    "minor": "1",
    "patch": "1"
  },
  "params": {
    "global": {
      "import_param_1" : "value_4",
      "import_param_2" : "value_2",
      "import_structure_1": {
        "general": "general"
        },
        "param_ABC": True
    }
  },
  "preprocessor": {
    "definitions": {
      "import_param_3" : "value_3",
      "import_param_4" : "value_4",
        "ABC": "param_ABC",
      "import_structure_1": {
         "general": "general"
       }
    }
```

```
  },
  "TargetName" : "device_name"
}
```

2.2.6 Accepted `True`, `False`, and `None`

Some keywords are different between Json and Python syntax:

- Json syntax: **"true"**, **"false"**, **"null"**
- Python syntax: **"True"**, **"False"**, **"None"**

To facilitate the usage of configuration files in Json format, both ways of syntax are accepted.

Chapter 3

CJsonPreprocessor.py

3.1 Class: CSyntaxType

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CSyntaxType
```

3.2 Class: CPythonJSONDecoder

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CPythonJSONDecoder

Add python data types and syntax to json. ``True``, ``False`` and ``None`` will be a
↪  accepted as json syntax elements.
```

Args:

json.JSONDecoder (*object*)

Decoder object provided by `json.loads`

3.2.1 Method: custom_scan_once

3.3 Class: CJsonPreprocessor

Imported by:

```
from JsonPreprocessor.CJsonPreprocessor import CJsonPreprocessor
```

CJsonPreprocessor extends the syntax of json.

Features are

- Allow c/c++-style comments within json files.
  // single line or part of single line and /* */ multline comments are possible
- Allow to import json files into json files
  `"[import]" : "relative/absolute path"`, imports another json file to exactly this location.
- Allow use of the defined paramaters within json files
  In any place the syntax `${basenode.subnode. ... nodename}` allows to reference an already existing parameter.

  - Example:

```
{
    "basenode" : {
                        subnode : {
                                        "myparam" : 5
                                    },

                    },

    "myVar" : ${basenode.subnode.myparam}
}
```

- Allow Python data types `True`, `False` and `None`

3.3.1 Method: jsonLoad

This function is the entry point of JsonPreprocessor.

It loads the json file, preprocesses it and returns the preprocessed result as data structure.

Args:

jFile (*string*)

Relative/absolute path to main json file.
`%envvariable%` and `${envvariable}` can be used, too in order to access environment variables.

Returns:

oJson (*dict*)

Preprocessed json file(s) as dictionary data structure

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	JsonPreprocessor
Version	0.2.3
Date	04.05.2023
Description	Preprocessor for json files
Package URL	python-jsonpreprocessor
Author	Mai Dinh Nam Son
Email	son.maidinhnam@vn.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 5

History

0.1.0	01/2022
Initial version	
0.1.4	09/2022
Updated documentation	