The Json Preprocessor

Pollerspoeck Thomas (XC-CI1/ECA3)

Feb 10, 2022

CONTENTS:

CHAPTER

ONE

JSON PREPROCESSOR'S FEATURE DOCUMENTATION

1.1 Introduction:

The JsonPreprocessor is a Python3 package which allows programmers to handle additional features in json files such as

- add comments
- import other json files
- overwrite already existing parameters with new values

These json files will be handled by the JsonPreprocessor package which returns as result a dictionary object of the deserialized data.

1.1.1 New features

Adding comments to Json file

Import the contents from other json files

Overwrite existing and add new parameters

Nested parameters

1.2 Features in details

1.2.1 Adding comments to Json file

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.

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.

Example:

```
//*****************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*****************************************************************************
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // Global parameters
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 1.332,  // This parameter is used to configure for ....
      "gGlobalString"   : "This is a string",
      "gGlobalStructure": {
        "general": "general"
      }
    }
  },
  "preprocessor": {
    "definitions": {
      // FEATURE switches
      "gPreprolIntParam" : 1,
```

(continues on next page)

(continued from previous page)

```
      "gPreproFloatParam" : 1.332,
       // The parameter for feature ABC
      "gPreproString"   : "This is a string",
      "gPreproStructure": {
                           "general": "general"
                          }
      }
  },
  "TargetName" : "gen3flex@dlt"
}
```

1.2.2 Import the contents from 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 similar 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
{
  "gGlobalIntParam" : 1,

  "gGlobalFloatParam" : 1.332, // This parameter is used to configure for ....

  "gGlobalString"   : "This is a string",

  "gGlobalStructure": {
    "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
//*************************************************************************************
{
```

(continues on next page)

(continued from previous page)

```
  "gPreprolIntParam" : 1,

  "gPreproFloatParam" : 1.332,
  // The parameter for feature ABC
  "gPreproString"   : "This is a string",

  "gPreproStructure": {
                         "general": "general"
                       }
}
```

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

```
//*****************************************************************************
//  Author: ROBFW-AIO Team
//
//  This file defines all common global parameters and will be included to all
//  test config files
//*****************************************************************************
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // Global parameters
    "global": {
               "[import]": "<path_to_the_imported_file>/params_global.json"
      }
    },
  "preprocessor": {
    "definitions": {
      // FEATURE switches
        "[import]": "<path_to_the_imported_file>/preprocessor_definitions.json"
      }
  },
  "TargetName" : "gen3flex@dlt"
}
```

The `config.json` file is handled by JsonPreprocessor package, then return the dictionary object for a program like below:

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  "version": {
    "majorversion": "0",
    "minorversion": "1",
```

(continues on next page)

(continued from previous page)

```
    "patchversion": "1"
  },
  "params": {
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 1.332,
      "gGlobalString"   : "This is a string",
      "gGlobalStructure": {
        "general": "general"
       }
    }
  },
  "preprocessor": {
    "definitions": {
      "gPreprolIntParam" : 1,
      "gPreproFloatParam" : 1.332,
      "gPreproString"   : "This is a string",
      "gPreproStructure": {
                           "general": "general"
                         }
    }
  },
  "TargetName" : "gen3flex@dlt"
}
```

1.2.3 Overwrite existing and add new parameters

This package also provides user ability to overwrite or update as well as add new parameters. User 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. This feature could help us update new value for existing parameters, it also supports to add new parameters to existing configuation object.

Example:

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

```
{
  "gGlobalIntParam" : 1,

  "gGlobalFloatParam" : 1.332, // This parameter is used to configure for ....

  "gGlobalString"   : "This is a string",

  "gGlobalStructure": {
    "general": "general"
  }
}
```

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

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // Global parameters
    "global": {
                "[import]": "<path_to_the_imported_file>/params_global.json"
      }
    },
  "TargetName" : "gen3flex@dlt",
  // Overwrite parameters
  "${params}['global']['gGlobalFloatParam']": 9.999,
  "${version}['patchversion']": "2",
  "${params}['global']['gGlobalString']": "This is the new value for the already␣
↪existing parameter.",
  // Add new parameters
  "${newParam}": {
                                "abc": 9,
                                "xyz": "new param"
  },
  "${params}['global']['gGlobalStructure']['newGlobalParam']": 123
}
```

The `config.json` file is handled by JsonPreprocessor package, then return the dictionary object for a program like below:

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "2"
  },
  "params": {
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 9.999,
      "gGlobalString"   : "This is the new value for the already existing parameter.",
      "gGlobalStructure": {
        "general": "general",
               "newGlobalParam": 123
        }
      }
  },
  "TargetName": "gen3flex@dlt",
  "newParam": {
```

(continues on next page)

(continued from previous page)

```
        "abc": 9,
        "xyz": "new param"
  }
}
```

1.2.4 Nested parameters

With JsonPreprocessor package, user can also use nested parameters as example below:

Example:

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

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    // Global parameters
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 1.332, // This parameter is used to configure for ....
      "gGlobalString"   : "This is a string",
      "gGlobalStructure": {
        "general": "general"
       }
    }
  },
  "preprocessor": {
    "definitions": {
      "gPreprolIntParam" : 1,
      "gPreproFloatParam" : 9.664,
         "ABC": "checkABC",
      "gPreproString"   : "This is a string",
      "gPreproStructure": {
                          "general": "general"
                        }
    }
  },
  "TargetName" : "gen3flex@dlt",
  // Nested parameter
  "${params}['global'][${preprocessor}['definitions']['ABC']]": true,
  "${params}['global']['gGlobalFloatParam']": "${preprocessor}['definitions'][
↪'gPreproFloatParam']"
}
```

The `config.json` file is handled by JsonPreprocessor package, then return the dictionary object for a program like below:

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 9.664,
      "gGlobalString"   : "This is a string",
      "gGlobalStructure": {
        "general": "general"
       },
        "checkABC": true
    }
  },
  "preprocessor": {
    "definitions": {
      "gPreprolIntParam" : 1,
      "gPreproFloatParam" : 9.664,
         "ABC": "checkABC",
      "gPreproString"   : "This is a string",
      "gPreproStructure": {
                           "general": "general"
                         }
    }
  },
  "TargetName" : "gen3flex@dlt"
}
```

1.3 Feedback

To give us a feedback, you can send an email to Thomas Pollerspöck or RBVH-ECM-Automation_Test_Framework-Associates

In case you want to report a bug or request any interesting feature, please don't hesitate to raise a ticket on our Jira

1.4 References

For more information please refer to our Bosch Connect Community

CHAPTER

TWO

JSONPREPROCESSOR PACKAGE

2.1 Module contents

class CJsonPreprocessor.**CJsonPreprocessor**(*syntax='json'*, *currentCfg={}*)

Bases: object

CJsonPreprocessor helps to handle configuration file as json format:

- Allow comment within json file
- Allow import json file within json file

jsonLoad(*jFile*, *masterFile=True*)

Method: jsonLoad loads the json file then parses to dict object

Args: jFile: string, json file input

Returns: oJson: dict

class CJsonPreprocessor.**CPythonJSONDecoder**(**args*, ***kwargs*)

Bases: json.decoder.JSONDecoder

Add below python values when scanning json data

True	True
False	False
None	None

custom_scan_once(*string*, *idx*)

class CJsonPreprocessor.**CSyntaxType**

Bases: object

json = 'json'

python = 'python'

PYTHON MODULE INDEX