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 the python3 package which allows programmer to handle some additional features in json file such as comment, import, override, etc for configuring purpose. The json file containing comment, import,… will be handled by JsonPreprocessor package then returned the final dictionary object for python program.

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

The JsonPreprocessor allows adding comments into json file, a comment could be added follow **"//"**.

This is required becuase some big projects require a large json configuration file which contains a huge number of configuration parameters for the different features. Therefore, having the need of adding comments into json file to clarify the different configuration parameters.

Note: This package is not allow commented a block of json code, each comment must be added with **"//"**.

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,
      "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 allows user merges the content of other json files into the json file, it also allows the nested importing means we can use import feature in the imported files.

The import feature helps user distinguish the configuration parameters of each functions, purposes, variants, and so forth into the separate json files. Therefore, we can easy to understand and manage the configuration parameters of big project.

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
//************************************************************************************
{
  "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",
    "patchversion": "1"
  },
  "params": {
    "global": {
      "gGlobalIntParam" : 1,
      "gGlobalFloatParam" : 1.332,
      "gGlobalString"   : "This is a string",
      "gGlobalStructure": {
        "general": "general"
        }
    }
  },
  "preprocessor": {
    "definitions": {
```

(continues on next page)

(continued from previous page)

```
      "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"
```

(continues on next page)

(continued from previous page)

```
      }
    },
  "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": {
         "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",
```

(continues on next page)

(continued from previous page)

```
    "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