
The RobotFramework Testsuites Management

Pollerspoeck Thomas (XC-CI1/ECA3)

Mar 07, 2022

CONTENTS:

1	ROBFW-AIO Testsuite’s Documentation	1
1.1	Introduction:	1
1.2	Features	1
1.2.1	ROBFW project is configured with json files	1
1.2.2	Dotdict features	3
1.2.3	How to use the parameters defined in json configuration file	3
1.3	Feedback	5
2	CConfig module	7
3	Keywords.COnFailureHandle module	9
4	Keywords.CSetup module	11
5	Utils.Events.Event module	13
6	Utils.Events.ScopeEvent module	15
7	Utils.CStruct module	17
8	Utils.LibListener module	19
9	version module	21
	Python Module Index	23

ROBFW-AIO TESTSUITE'S DOCUMENTATION

1.1 Introduction:

The RobotFramework_Testsuites package works together with [JsonPreprocessor](#) python package to provide the enhanced features such as json configuration files, 4 different levels of configuration, config object and global params, schema validation,...

1.2 Features

1.2.1 ROBFW project is configured with json files

RobotFramework_Testsuites supports configuring ROBFW automation test project with json files which allow user adds the comments, imports params from other json files. Adding comments and importing json files are enhanced features which are developed and documented in [JsonPreprocessor](#) python package.

RobotFramework_Testsuites management defines 4 different configuration levels, from level 1 -> level 4, Level 1 is highest priority, and level 4 is lowest priority:

Level 1: Load configuration file while executing robot testsuite by command

This is highest priority configuration level, it is called **configuration level 1**

User can address the json configuration file when executing robot testsuite with input parameter `--variable config_file:"<path_to_json_file>"`

Ex: `robot --variable config_file:"<path_to_json_file>" <path_to_testsuite>`

The level 1 configuration could be set by defined the `${config_file}` in `*** Variables ***`

Ex:

```
*** Variables ***
${config_file}    <Path_to_configuration_file>

*** Settings ***
#Force Tags       atestExcluded
Library           RobotFramework_Testsuites    WITH NAME    testsuites
Suite Setup       testsuites.testsuite_setup
Suite Teardown    testsuites.testsuite_teardown
Test Setup        testsuites.testcase_setup
Test Teardown     testsuites.testcase_teardown
```

Level 2: In case project have many variants, it reads from json file's content to select the corresponding variant configuration

If the **level 1** is not configured, it will check the configuration for **level 2**.

In level 2 configuration, user has to create a json file which contains different variants point to different configuration files. For example, we create the `variants_cfg.json` with content below:

```
//*****
// The file configures the access to all variant dependent robot_config*.json
// files.
//
// The path to the robot_config*.json files depends on the test file location. A
// different number of ../ is required dependend on the directory depth of the test
// case location.
// Therefore we use here three ../ to tell the ROBFW to search from the test
// file location up till the robot_config*.json files are found:
// ./config/robot_config.json
// ../config/robot_config.json
// ../../config/robot_config.json
// ../../../../config/robot_config.json
// and so on.
//*****
{
  "default": {
    "name": "robot_config.json",
    "path": "../config/"
  },
  "variant_0": {
    "name": "robot_config.json",
    "path": "../config/"
  },
  "variant_1": {
    "name": "robot_config_variant_1.json",
    "path": "../config/"
  },
  "variant_2": {
    "name": "robot_config_variant_2.json",
    "path": "../config/"
  }
}
```

User can set configuration level 2 only in testsuite like below:

```
*** Settings ***
Library          RobotFramework_Testsuites    WITH NAME    testsuites
Suite Setup      testsuites.testsuite_setup    <Path_to_the_file_variants_cfg.json>
Suite Teardown   testsuites.testsuite_teardown
Test Setup       testsuites.testcase_setup
Test Teardown    testsuites.testcase_teardown
```

Level 3: Find the config/ folder in testsuite directory, if the config folder is found, it will load configuration file in this folder

In case **level 1** and **level 2** are not configured, it will check the configuration for **level 3**.

If there is the configuration file have the same name with testsuite file (ex: abc.robot & ./config/abc.json), then it will load this configuration file. If the first case doesn't occur, it will load the configuration file ./config/robot_config.json. In case these 2 cases are not matched, it will load the configuration level 4 (default and lowest priority)

Ex:

We have testsuite ./component/abc.robot

In ./component/config/ contains abc.json and robot_config.json, then ./component/config/abc.json will be loaded.

In ./component/config/ contains only robot_config.json, then ./component/config/robot_config.json will be loaded.

If there is no ./component/config/ or the directory ./component/config/ doesn't have abc.json or robot_config.json, then configuration level 4 will be set.

Level 4: Lowest priority level, it reads default configuration file

The default configuration file (robot_config.json) in installation directory:

```
python39\Lib\site-packages\RobotFramework_Testsuites-0.1.0-py3.9.egg\
RobotFramework_Testsuites\Config\robot_config.json
```

The default configuration file just contains some basic parameters:

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "TargetName" : "gen3flex@dlt"
}
```

1.2.2 Dotdict features

User can access dictionary object in robot test script by called \${dict}[abc][def] or \${dict.abc.def}

Note: In case a parameter name contains a ".", then we could not use dotdict but the traditional way \${dict}[abc][def] is still working.

1.2.3 How to use the parameters defined in json configuration file

We design the special format of json configuration file, so users can define the global variables for Robot project:

```
/** *****
//
// File: robot_config.json
// Initialized by ROBFW-AIO team
//
/** *****
```

(continues on next page)

(continued from previous page)

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  // Version control information.
  "version": {
    "majorversion": "0",
    "minorversion": "1",
    "patchversion": "1"
  },
  "params": {
    "global": {
      ...
      // The objects define here will become robot global variables
    }
  },
  "preprocessor": {
    "definitions": {
      ...
      // The objects define here will become robot global variables
    }
  },
  "Project": "G3g-variant_2"
}
```

All parameters which are defined in `params.global` and `preprocessor.definitions` will be used as the global variables in robot script, and used directly in robot script. The other parameters will be defined in the `{CONFIG}` variables, and we can use them by calling `{CONFIG}[abc]` or `{CONFIG}.abc`.

Ex: If we create json configuration like below:

```
{
  "Project": "G3g",
  "WelcomeString": "Hello... ROBFW is running now!",
  "params": {
    "global": {
      "variable_01": 1
    }
  },
  "preprocessor": {
    "definitions": {
      "preprocessor_var": "definition"
    }
  },
  "Project": "G3g-variant_2"
}
```

Then, in robot script you can call `{variable_01}` and `{preprocessor_var}` to get the value 1 and definition. But to get the `WelcomeString` value you have to call `{CONFIG>WelcomeString}` or `{CONFIG}[WelcomeString]`

1.3 Feedback

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

CCONFIG MODULE

```
class RobotFramework_Testsuites.Config.CConfig.CConfig(*args, **kwargs)
```

Bases: object

Defines the properties of configuration Holds the identified config files. Level1 is highest priority, Level4 is lowest priority.

(remaining content needs to be fixed and restored)

```
class CJsonDotDict
```

Bases: object

The CJsonDotDict class converts json configuration object to dotdict

```
dotdictConvert(oJson)
```

Method: dotdictConvert converts json object to dotdict

Args: oJson: dict

Returns: CConfig.ddictJson: dotdict

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

```
bConfigLoaded = False
```

```
bLoadedCfg = True
```

```
static bValidateMaxVersion(tCurrentVersion, tMaxVersion)
```

Validate current version with required maximum version.

```
static bValidateMinVersion(tCurrentVersion, tMinVersion)
```

Validate current version with required minimum version.

```
static bValidateSubVersion(sVersion)
```

Validate the format of provided sub version and parse it into sub tuple for version comparison.

```
ddictJson = {}
```

```
iSuiteCount = 0
```

```
iTestCount = 0
```

```
iTotalTestcases = 0
```

```
static loadCfg(self)
```

```
oConfigParams = {}
```

```
rConfigFiles = <RobotFramework_Testsuites.Utills.CStruct.CStruct object>
```

```
rMetaData = <RobotFramework_Testsuites.Utills.CStruct.CStruct object>
```

```
static sCalcAbsPath(self, relativePath)
```

Staticmethod: sCalcAbsPath

Args: `relativePath`: String

Returns: `absolutePath`: String

`sConfigFileName` = None

`sConfigName` = 'default'

`sLoadedCfgError` = ''

`sMaxVersion` = ''

`sMinVersion` = ''

static `sNormalizePath(sPath)`
staticmethod `sNormalizePath`:
(remaining content needs to be fixed and restored)

`sProjectName` = None

`sTargetName` = None

`sTestCfgFile` = ''

`sTestSuiteCfg` = ''

`sTestcasePath` = ''

`sWelcomeString` = None

static `tupleVersion(sVersion)`
Return a tuple which contains the (major, minor, patch) version.
(remaining content needs to be fixed and restored)

updateCfg()

staticmethod `updateParams`: This method updates preprocessor, global or local params base on ROBFW local config or any json config file according to purpose of specific testsuite.

Args: `sUpdateCfgFile`: str

Returns: None

verifyRbfwVersion()
Validate the current robotframework version with maximum and minimum version (if provided in the configuration file). In case the current version is not between min and max version, then the execution of testsuite is terminated with “unknown” state

versioncontrol_error(reason, version1, version2)
Wrapper version control error log: Log error message of version control due to reason and set to unknown state. *reason* can only be “conflict_min”, “conflict_max” and “wrong_minmax”.

KEYWORDS.CONFAILUREHANDLE MODULE

```
class RobotFramework_Testsuites.Keywords.COnFailureHandle.COnFailureHandle
    Bases: object
    is_noney(item)
    register_keyword_run_on_failure(keyword)
        TBD
```


KEYWORDS.CSETUP MODULE

class RobotFramework_Testsuites.Keywords.CSetup.CGeneralKeywords

Bases: object

Definition setup keywords

get_config()

oConfigParams: is the dictionary consist of some configuration params which are return to user from
get_config_params keyword

load_json(*jsonfile, level=1, variant='default'*)

This keyword uses to load json file then return json object.

- Level = 1 -> loads the content of jsonfile.
- level != 1 -> loads the json file which is set with variant (likes loading config level2)

class RobotFramework_Testsuites.Keywords.CSetup.CSetupKeywords

Bases: object

Definition setup keywords

testcase_setup()

testcase_teardown()

testsuite_setup(*sTestsuiteCfgFile=""*)

testsuite_teardown()

update_config(*sCfgFile*)

UTILS.EVENTS.EVENT MODULE

```
class RobotFramework_Testsuites.Utils.Events.Event.Event  
    Bases: object  
    abstract trigger(*args, **kwargs)
```


UTILS.EVENTS.SCOPEEVENT MODULE

```
class RobotFramework_Testsuites.Utls.Events.ScopeEvent.ScopeEnd(scope, action, *args, **kwargs)
    Bases: RobotFramework_Testsuites.Utls.Events.ScopeEvent.ScopeEvent
    name = 'scope_end'

class RobotFramework_Testsuites.Utls.Events.ScopeEvent.ScopeEvent(scope, action, *args,
                                                                    **kwargs)
    Bases: RobotFramework_Testsuites.Utls.Events.Event.Event
    trigger(*args, **kwargs)

class RobotFramework_Testsuites.Utls.Events.ScopeEvent.ScopeStart(scope, action, *args,
                                                                    **kwargs)
    Bases: RobotFramework_Testsuites.Utls.Events.ScopeEvent.ScopeEvent
    name = 'scope_start'
```


UTILS.CSTRUCT MODULE

This class provides the “struct” functionality of “C/C++” in python. It simply helps to organize data which belongs logically together.

Usage: `oStruct=CStruct(attribute_1=value_1,... attribute_n=value_n)` `oStruct.attribute_1=“....”`

class `RobotFramework_Testsuites.Utils.CStruct.CStruct(*args, **kwargs)`

Bases: `object`

Constructor `__init__` creates the given attributes dynamically at runtime.

Args:

Attributes to be created with the initial value

Returns:

Accessible attributes

UTILS.LIBLISTENER MODULE

```
class RobotFramework_Testsuites.Utils.LibListener.LibListener
    Bases: object
    Define some hook methods
    ROBOT_LIBRARY_SCOPE = 'GLOBAL'
    ROBOT_LISTENER_API_VERSION = 2
```


VERSION MODULE

RobotFramework_Testsuites.version.**robfwaio_version()**

Return testsuitemanagement version as Robot framework AIO version

PYTHON MODULE INDEX

r

`RobotFramework.Testsuites.Config.CConfig`, [7](#)
`RobotFramework.Testsuites.Keywords.COnFailureHandle`,
 [9](#)
`RobotFramework.Testsuites.Keywords.CSetup`, [11](#)
`RobotFramework.Testsuites.Utls.CStruct`, [17](#)
`RobotFramework.Testsuites.Utls.Events.Event`,
 [13](#)
`RobotFramework.Testsuites.Utls.Events.ScopeEvent`,
 [15](#)
`RobotFramework.Testsuites.Utls.LibListener`,
 [19](#)
`RobotFramework.Testsuites.version`, [21](#)