**Redfish Service Conformance Check Tool**

**High Level Design and Developers Guide**

Initial code released: 01/2016

Steve Krig - Intel

Fatima Saleem - Intel

© 2015 Intel Corporation

Revision release date: 07/11/16

Redfish Service Conformance Check Tool’s source code licensed under the Apache license: http://www.apache.org/licenses/LICENSE-2.0

Contents

[1. Introduction 3](#_Toc457562907)

[1.1. Purpose and Scope of this Document 3](#_Toc457562908)

[1.2. Terminology 3](#_Toc457562909)

[1.3. Reference Documents 3](#_Toc457562910)

[2. Requirements 4](#_Toc457562911)

[3. Assumptions, Dependencies and To Do List 4](#_Toc457562912)

[3.1. Dependencies 4](#_Toc457562913)

[4. Application High-level Design 4](#_Toc457562914)

[4.1. Application Overview 4](#_Toc457562915)

[4.2. Target Hardware 5](#_Toc457562916)

[4.3. Application Flow Diagram 5](#_Toc457562917)

[4.4. Tool configuration 5](#_Toc457562918)

[4.5. Run Assertions using Tool: 6](#_Toc457562919)

[4.6. Tool logging 7](#_Toc457562920)

[4.7. Key Design Decisions and Alternatives 8](#_Toc457562921)

[5. Modules High Level Design 8](#_Toc457562922)

[5.1. Properties 8](#_Toc457562923)

[5.1.1. SUT configuration: 8](#_Toc457562924)

[5.1.2. Schema File configuration: 8](#_Toc457562925)

[5.2. Client 9](#_Toc457562926)

[5.2.1. Available methods and its usage 9](#_Toc457562927)

[5.3. SUT 10](#_Toc457562928)

[5.3.1. Available methods and its usage 10](#_Toc457562929)

[5.4. Utility 11](#_Toc457562930)

[5.4.1. Available methods and its usage 11](#_Toc457562931)

[5.5. Schema 12](#_Toc457562932)

[5.5.1. Available methods and its usage 12](#_Toc457562933)

[5.6. Logger 13](#_Toc457562934)

[5.6.1. Available methods and its usage 13](#_Toc457562935)

[5.7. Assertions 13](#_Toc457562936)

[5.7.1. rfs\_test folder 13](#_Toc457562937)

Tables

[Table 1 List of terms 4](#_Toc457562938)

[Table 2 Reference documents 4](#_Toc457562939)

[Table 3 Dependencies 5](#_Toc457562940)

Figures

[Figure 1 Tool flow diagram 5](#_Toc457562943)

[Figure 2 Script to run assertions 7](#_Toc457562944)

[Figure 3 rf\_client.py 10](#_Toc457562945)

[Figure 4 rf\_sut.py 11](#_Toc457562946)

[Figure 5 rf\_utility.py 12](#_Toc457562947)

[Figure 6 schema.py 13](#_Toc457562948)

[Figure 7 logger.py 14](#_Toc457562949)

[Figure 8 rfs\_test 14](#_Toc457562950)

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Description | Date | Author |
| 0.1 | Initial version with developers guide | 07/11/2016 | Fatima Saleem |
| 0.2 | Added tools high level design, updated format | 07/25/2016 | Fatima Saleem |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# **Introduction**

## **Purpose and Scope of this Document**

This document aims to provide guidance and samples on how to use different modules of the tool and their functionalities to either use it via python command line or in code/script.

## **Terminology**

Table 1 List of terms

|  |  |
| --- | --- |
| Term | Description |
| SUT | System Under Test , System w/redfish service |
| Client | System used to run the tool |
| Assertions | Tests extracted from RSPMS based on the terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" |

## **Reference Documents**

Table 2 Reference documents

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Document Name | Version | File Location |
| readme | readme |  | Same directory as this file |
| RSPMS | Redfish Scalable Platforms Management API Specification | 1.0.2 | http://redfish.dmtf.org/schemas/DSP0266\_1.0.html |
| Schemas | Redfish Schema Repository | 1.0.2 | http://redfish.dmtf.org/redfish/schema\_index |

# **Requirements**

An SUT with Redfish service enabled is required to run this tool. There are 2 deployment scenarios

* Local: The tool can be deployed and run locally on the SUT, in which case Client is the same as SUT
* Remote: Optionally, this tool can be deployed and run on a Client with remote access to SUT. In this case the tool can run on multiple SUTs in sequence with one instance of the tool or parallel with multiple instances of the tool.

# **Assumptions, Dependencies and To Do List**

## **Dependencies**

Table 3 Dependencies

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| 1 | Python 2.7 or 3.4 | Python environment setup on Client |
| 2 | openpyxl library | To run the tool as-is, install openpyxl using pip |
| 3 | assertions folder | To run the tool as-is , assertions folder should be present in the same directory as rf\_client.py with rf-assertions-run.xlsx file in the folder |
| 4 | properties json file | To run the tool as-is, properties.json should be present in the same directory as rf\_client.py with proper settings provided in the section 4.3 of this document |
| 5 | Schemas | If remote fetching of the redfish schemas is disabled in the tool (refer section 4.3 for enable/disable details), then schema files should be present in a separate folder in the same directory as rf\_client.py, with folder name correctly set in properties.json. For folder name configuration in the tool refer section 4.3 |

# **Application High-level Design**

## **Application Overview**

The tool provides implementation to explore the operational Redfish Service and verifies it’s conformance to the normative statements in the RSPMS. Assertion coverage is growing (development in process) and future revisions of the tool will increase coverage of the Assertions. This tool provides command line interface only.

## **Target Hardware**

Server with redfish service deployed

## **Application Flow Diagram**

Figure 1 Tool flow diagram

As shown in the [Application Flow Diagram](#_Application_Flow_Diagram), the tool starts with the Client with the support of properties.json, it gets the SUT properties from properties.json. The property object is passed to the setup\_tool() function which with the help of SUT API sets up the SUT object with it’s relevant Redfish service related data used throughout the Assertions. Some of the SUT API functions internally calls the HTTP and Restful Utility module. Client also uses Schema Model to serialize the CSDL Schema files in appropriate structures used throughout the Assertions. Once the set up completes the Client calls the run() function in Conformance Test Suite to run the Assertions. Conformance Test suite uses the Logger (required) and some functions in the HTTP and Restful Utility module.

## **Tool configuration**

To setup the tool:

1. Install openpyxl using pip, (if openpyxl is already installed, skip this step)

With proxy:

C:\your\_python> pip --proxy <hostname>:<port> install openpyxl  
Without proxy:

C:\your\_python> pip install openpyxl

1. Properties.json configuration:
   * SUTs configuration:

Make sure required SUT properties are set in properties.json. Each SUT is represented as a dictionary within the SUTs list. The required properties for each SUT are as follows: DisplayName, DnsName, LoginName and Password (refer section 5.1 for description of keys).

* + RedfishServiceCheckTool\_SchemaFiles configuration:

Provide the folder name where Schema files reside/should reside. This folder should be placed in the same directory as rf\_client.py. The key for this property is “LocalSchemaDirectoryFolder”. 2 folders are expected within this parent Schema folder, one for json schemas named “json-schema” and csdl schemas folder named “metadata”

If schema is required to be downloaded from DMTF Repository hosted on DMTF’ site, Set ‘’RetrieveDMTFSchemas” to “yes”, else change it to “no”

If proxy is required to bypass firewall, set appropriate “https\_proxy” or ‘http\_proxy’ according to the requirement of URL. If no proxy is required, set both “https\_proxy” and http\_proxy to “none”

*Note:   
It is also possible to download the Zip folder of the Schemas hosted on the DMTF site. Refer Section 4.6 for details.*

## **Run Assertions using Tool:**

Quickest way to run the tool is by having proper settings in properties.json (see section 4.4) and running rf\_client.py.

To write a script to run assertions on each SUT provided in properties.json, make sure properties.json has required values and follow these steps:

Figure 2 Script to run assertions

*Note: Use python functions dir() and help() in command line for more info on objects and functions.*

## **Tool logging**

Tool logging is based on 2 types of logging file formats:

* **Spreadsheet logging for Assertions**  
  Once the tool runs it creates Log folder in the same directory as rf\_client.py and a subfolder within it using the SUT’s DisplayName, it copies rf\_assertions-run.xlsx file in this subfolder, Current timestamp is appended to the file name. Assertion numbers in the spreadsheet are marked with colors ‘Green’, ‘Yellow’ and ‘RED’ indicating ‘PASS’, ‘WARN’ and ‘FAIL’ respectively. Comments are placed against each assertions whenever appropriate.
* **Text file logging**  
  Text file is also created with more detailed log for assertions placed in the same folder as the logged spreadsheet

## **Key Design Decisions and Alternatives**

# **Modules High Level Design**

## **Properties**

Properties used in the tool for SUT and Schema configuration are placed in a json file ‘properties.json’.

### **SUT configuration:**

One or more SUTs can be defined and properties for each are defined in “SUTs” list. Following keys can be set:

{

"AllowAction\_LogServiceClearLog": "yes",

"DisplayName": "server1",

"DnsName": "foo.domain.com",

"LoginName": “Foo”,

"Password": "Bar",

"RedfishVersion": "v1"

}

Where:

* DisplayName: String Name identifying the SUT
* DnsName: Domain name or IP address of the SUT
* LoginName: Login id of the SUT
* Password: Password of the SUT
* RedfishVersion: Current version expected for the SUT
* AllowAction\_LogServiceClearLog: Set value “yes” or “no”, where yes indicates the assertions related to log clear action can be run, no indicates otherwise. If no value is provided, default value is “no”.

### **Schema File configuration:**

Schema files are required for this tool to run various assertions. There are 2 types of schema files, json schemas and csdl schemas. They are both provided by DMTF. Schema configuration properties are as follows:

"RedfishServiceCheckTool\_SchemaFiles": {

“Description”: “”,

"LocalSchemaDirectoryFolder": "redfish-schemas-v1",

"RetrieveDMTFSchemas”: "no",

"DMTF\_SPMFSchemas": {

"SchemaRepository": "http://redfish.dmtf.org/schemas",

"SchemaZipFileName”: "DSP8010\_1.0.0.zip",

"SchemaVersion": "v1",

"ClientProxy”: {

"https\_proxy": "https://proxyurl:port",

"http\_proxy": "http://proxyurl:port"

}

}

}

Where:

* LocalSchemaDirectoryFolder: Folder name where schemas reside or folder destination for schemas download
* RetrieveDMTFSchemas: Set value “yes” or “no”, where yes indicates tool should attempt to download schemas from DMTF site, no indicates tool should expect the schemas files in the folder and should not fetch new copies
* SchemaRepository: URL for Redfish Schema Repository from where the tool should attempt to fetch the schema files
* SchemaVersion: Version of the schema files the tool should expect
* ClientProxy: Set proxy url and port to bypass firewall
  + https\_proxy: https proxy url:port or “none” if no proxy is required
  + http\_proxy: http proxy url:port or “none” if no proxy is required
* Description or Long Description : Can be provided for detailed description of the above keys

## **Client**

### **Available methods and its usage**

Client module provides the functionalities shown in Figure 3. For more details on each function, its parameters and return values, please see rf\_client.py

Figure 3 rf\_client.py

## **SUT**

### **Available methods and its usage**

SUT module provides the functionalities shown in Figure 4. For more details on each function, its parameters and return values, please see rf\_sut.py

Figure 4 rf\_sut.py

Following are some of the examples of using rf\_sut.py functions:

Code: version, service\_root = sut\_obj.parse\_protocol\_version(“/redfish/”)

Get request headers

Code: headers = sut\_obj.get\_headers()

Perform GET

Code: res\_body, res\_header, status = sut\_obj.http\_GET(“/redfish/”, headers, “off”)

Get top level uris by parsing odata document

Code: odata\_context, top\_level\_uris = sut\_obj.parse\_odatadoc\_payload(“/redfish/v1/odata”)

## **Utility**

### **Available methods and its usage**

Utility module provides the functionalities shown in Figure 5. For more details on each function, its parameters and return values, please see rf\_utility.py

Figure 5 rf\_utility.py

Following are some of the examples of using rf\_utility.py functions:

*To request GET for a resource uri:*

Code: sut\_prop = {“DisplayName”: “sample”, “DnsName” : “foo.domain.com”, “LoginName” : “foo”, “Password” : “bar” }

rq\_header = rf\_service.create\_request\_headers()

res\_body, res\_header, res\_status = rf\_service.http\_\_GET(sut\_prop, “/redfish/v1/”, rq\_header, “on”)

*To convert status int code to string:*

Code: rf\_service.HTTP\_status\_string(status)

*Find if key exists in the payload and return key’s value*Code: found\_key, value = rf\_service.json\_get\_key\_value(‘@odata.type’)

## **Schema**

### **Available methods and its usage**

Figure 6 schema.py

## **Logger**

### **Available methods and its usage**

Figure 7 logger.py

## **Assertions**

### **rfs\_test folder**

Figure 8 rfs\_test