python-redfish Documentation

Release 0.4.2

Bruno Cornec, Vincent Misson, René Ribaud

Dec 31, 2049

CONTENTS

Contents:

CHAPTER

ONE

THE PYTHON-REDFISH PROJECT

This repository will be used to house the python-redfish library, a reference implementation to enable Python developers to communicate with the Redfish API (http://www.dmtf.org/standards/redfish).

NOTE:

```
STATUS: Work in progress, ready for proof of concept.

The current Redfish specification revision is 1.0.0 - Note that the mockup
is still at version 0.99.0a and may not reflect what the standard provides
fully
```

1.1 Documentation

The full documentation is available at http://pythonhosted.org/python-redfish/installation.html

1.2 Project Structure

This project follows the same convention as OpenStack projects, eg. using pbr for build and test automation:

```
doc/             # Documentation
doc/source       # The doc source files live here
doc/build/html   # Output of building any docs will go here
dmtf             # Reference documents and mockup provided by the DMTF
examples/        # Any sample code using this library, eg. for education
                 # should be put here
pbconf           # Project builder file to build rpm/deb packages for
                 # distributions
redfish/         # The redfish library itself
redfish/tests/   # python-redfish unit test suite
redfish-client   # Client tool to manage redfish devices
```

1.3 Requirements

To use the enclosed examples, you will need Python 2.7 or Python 3.4 (https://www.python.org/downloads/). Note that Python 2.7.9 enforces greater SSL verification requiring server certificates be installed. Parameters to relax the requirements are available in the library, but these configurations are discouraged due to security.

Python requirements are listed in requirements.txt; additional requirements for running the unit test suite are listed in test-requirements.txt.

Note: Running tests requires Docker engine.

Note: The program was tested with Python 2.7.10 and 3.4.2 however it might work as well with all Python 3 releases.

1.4 Get the source code

The source code is available on github and can be retrieved using:

```
git clone https://github.com/openstack/python-redfish.git
```

1.5 Installation

Please refer to the following link.

http://pythonhosted.org/python-redfish/installation.html

1.6 Contacts

Distribution list: python-redfish@mondorescue.org

1.7 Further References

Please look at dmtf/README.rst file.

CHAPTER

TWO

INSTALLATION

The following instructions are ordered by ease of use, and our project recommendations.

2.1 Using rpm packages

There is currently no official Linux distribution packages.

The upstream project provides packages for a limited set of Linux distributions.

There are available at ftp://ftp.project-builder.org

As an example for Fedora 23 use the following:

1. As root get the repo file:

```
cd /etc/yum.repos.d && wget ftp://ftp.project-builder.org/fedora/23/x86_64/python-
↪redfish.repo
```

2. Install using dnf:

```
dnf install python-redfish
```

2.2 Using pip and virtualenv

1. Install virtualenv and virtualenvwrapper:

   Fedora 22:

```
dnf install python-virtualenv python-virtualenvwrapper
```

   Ubuntu 15.04:

```
apt-get install python-virtualenv virtualenvwrapper
```

2. Source virtualenvwrapper.sh:

```
. /usr/bin/virtualenvwrapper.sh
```

   or:

```
. /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

3. Create a redfish virtual environement:

```
mkvirtualenv redfish
```

4. Install using pip:

```
pip install python-redfish
```

   All files are installed under your virtualenv.

2.3 Using pip

Use:

```
sudo pip install python-redfish
```

Pip will install :

1. The library and all dependencies into prefix/lib/pythonX.Y/site-packages directory
2. redfish-client conf file into prefix/etc/redfish-client.conf. If prefix = '/usr' then force the configuration file to be in /etc
3. Data files (templates) into prefix/share/redfish-client/templates

Point 2 and 3 above need root access to your system. If you don't have root access on your system, please follow *Using pip and virtualenv* section.

2.4 Using source code

1. Follow get the source code section to retrieve it.
2. Install from the source code using:

```
python setup.py install --prefix="/usr/local"
```

2.5 Building your own rpm packages

Inside the project tree there is a mechanism to build rpm packages for distributions.

The mechanism is based on project builder tool.

1. Follow get the source code section to retrieve it.
2. Download project builder for your distribution from ftp://ftp.project-builder.org.
3. Clone the project to your own github account.
4. Create a .pbrc with the following content, replace "/workspace/python/redfish" and "uggla" with your own directory and account:

```
$ cat .pbrc
pbdefdir python-redfish = $ENV{'HOME'}/workspace
pbconfdir python-redfish = $ENV{'HOME'}/workspace/python-redfish/pbconf
```

```
pbconfurl python-redfish = git+ssh://git@github.com:uggla/python-redfish.git
pburl python-redfish = git+ssh://git@github.com:uggla/python-redfish.git
```

5. Build the project:

```
pb -p python-redfish sbx2pkg
```

or:

```
pb -p python-redfish sbx2pkg2ins
```

6. All packages (srpm/rpm) should be available into the build directory, then install the package using rpm:

```
rpm -Uvh python-redfish/build/RPMS/python-redfish-devel20160213182552.rpm
```

CHAPTER

THREE

INVENTORY FILE CONFIGURATION

1. Verify redfish-client is working correclty:

```
redfish-client -h
```

2. Create a default entry to use the mockup:

```
redfish-client config add default default http://localhost:8000/redfish/v1
```

3. Verify the entry is correctly registered:

```
redfish-client config showall
```

Note: The inventory file is created in $HOME/.redfish

CHAPTER

FOUR

MOCKUP INSTALLATION

1. Follow get the source code section to retrieve it.
2. Install docker using your distribution packages or the docker procedure (docker provides more recent packages):

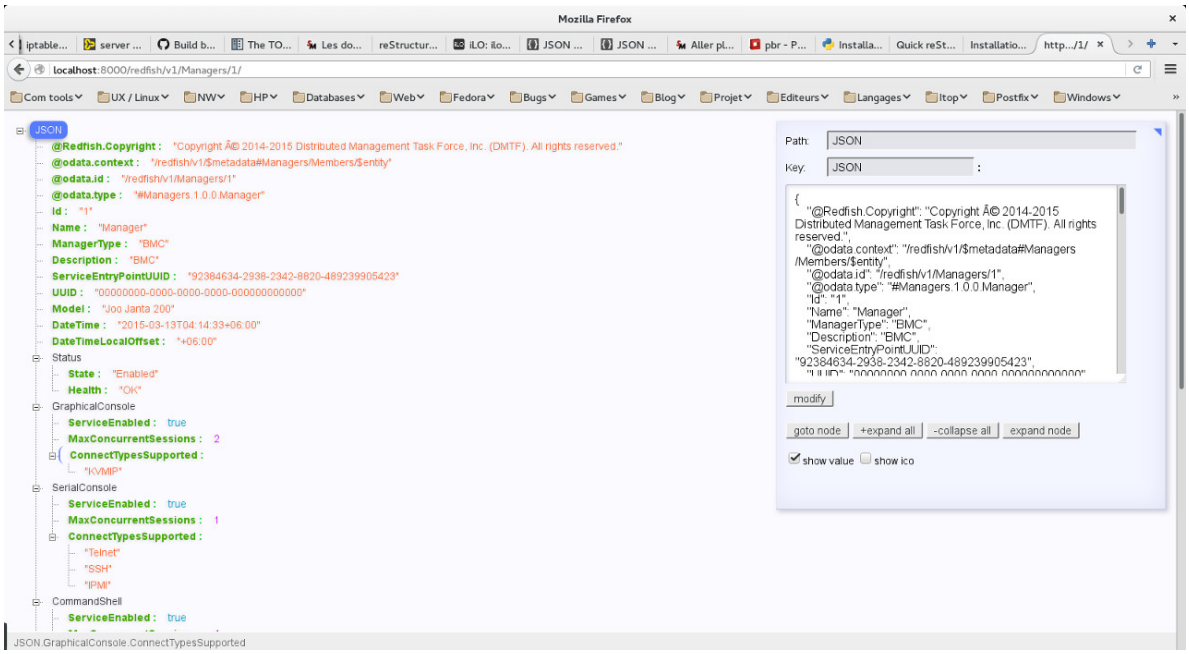
As an example for Fedora 23 use the following:

```
dnf install docker
systemctl enable docker.service
systemctl start docker.service
systemctl status docker.service
```

1. Jump into the dmtf directory.
2. Run ./buildImage.sh and ./run-redfish-simulator.sh
3. Check that a container is running and listening on port 8000:

```
$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
↪           STATUS              PORTS                    NAMES
9943ff1d4d93        redfish-simulator:latest    "/bin/sh -c /tmp/redf"   3 weeks
↪ago          Up 2 days           0.0.0.0:8000->80/tcp   redfish-simulator
```

4. Try to connect using a navigator to http://localhost:8000 the following screen should apear.



Note : in the above screenshot, firefox JSON-handle extension is used. If you want the same presentation install the extension and refresh the page.

CHAPTER

FIVE

TESTING AGAINST THE MOCKUP

1. Follow *Inventory file configuration* and *Mockup installation* section.
2. Run the following command:

```
redfish-client manager getinfo
```

The result should be like this:

```
$ redfish-client manager getinfo
Gathering data from manager, please wait...

Redfish API version :  1.00
Root Service

Managers information :
======================

Manager id 1:
UUID : 00000000-0000-0000-0000-000000000000
Type : BMC
Firmware version : 1.00
State : Enabled
Ethernet Interface :
    This manager has no ethernet interface
Managed Chassis :
    1
Managed System :
    1
----------------------------

Manager id 2:
UUID : 00000000-0000-0000-0000-000000000000
Type : EnclosureManager
Firmware version : Not available
State : Enabled
Ethernet Interface :
    This manager has no ethernet interface
Managed Chassis :
    Enc1
Managed System :
    2
----------------------------

Manager id 3:
UUID : 00000000-0000-0000-0000-000000000000
```

```
Type : EnclosureManager
Firmware version : Not available
State : Enabled
Ethernet Interface :
    This manager has no ethernet interface
Managed Chassis :
    Enc1
Managed System :
    2
----------------------------
```

CHAPTER

SIX

BUILDING LOCAL DOCUMENTATION

Building the html documentation locally.

1. Follow get the source code section to retrieve it.
2. Jump in the doc directory:

```
cd doc
```

3. Build the html documentation:

```
make html
```

If you want to build the documentation in pdf.

1. Get texlive full distribution, e.g. on Fedora 23:

```
dnf install texlive-scheme-full
```

2. Build the documentation:

```
make latexpdf
```

CHAPTER

SEVEN

USAGE

7.1 Example using the mockup

example/simple-simulator.py provide a simple library usage to interact with the redfish mockup.

7.2 Example using a proliant

example/simple-proliant.py provide a simple library usage to interact with a HPE ProLiant BL460C G9 server. However this example should work on any server supplier following redfish standard.

7.3 redfish-client usage

The client usage can be display using:

```
redfish-client -h
```

This is also available at http://pythonhosted.org/python-redfish/redfish-client.html.

CHAPTER

EIGHT

DEVELOPER SETUP

1. Follow get the source code section to retrieve the sources.
2. Follow using pip and virtualenv section to create your environment.

To setup a development environment, the DMTF has published a mockup environment to simulate a Redfish based system so it is possible to write programs without real Redfish compliant hardware platform.

Please look at _‘DMTF Documentation <further_ref.html#docker-container>‘_ for detailed steps. for setting up the environment.

You can start hacking the code now.

CHAPTER

NINE

RUNNING TESTS

9.1 redfish module tests

Tests are not functional for the redfish module yet.

9.2 redfish-client tests

1. Create your development environment following Developer setup.
2. Install docker using the procedure.
3. Ensure you can use docker with your current user.
4. Jump into the python-redfish directory containing the source code.
5. Depending of your distribution, you may have to upgrade setuptools:

```
pip install --upgrade setuptools
```

6. Install required modules for testings:

```
pip install -t test-requirements.txt
```

7. Run the test:

```
tox
```

or:

```
py.test redfish-client
```

CHAPTER

TEN

CLASSES DOCUMENTATION

10.1 python-redfish classes

10.1.1 config class

redfish.config.**initialize_logger**(*REDFISH_LOGFILE*, *CONSOLE_LOGGER_LEVEL*, *FILE_LOGGER_LEVEL*, *logger_name=None*)

Initialize a global logger to track application behaviour

Parameters

- **redfish_logfile** (*str*) – Log filename
- **screen_logger_level** (*logging constant or string*) – Console log level (logging.DEBUG, logging.ERROR, ..) or nolog
- **file_logger_level** (*logging constant*) – File log level

Returns logging object

10.1.2 exception class

exception redfish.exception.**RedfishException**(*message*, ***kwargs*)

Base class for redfish exceptions

10.1.3 main class

STARTING ASSUMPTIONS

On URIs:

The Redfish RESTful API is a "hypermedia API" by design. This is to avoid building in restrictive assumptions to the data model that will make it difficult to adapt to future hardware implementations. A hypermedia API avoids these assumptions by making the data model discoverable via links between resources.

A URI should be treated by the client as opaque, and thus should not be attempted to be understood or deconstructed by the client. Only specific top level URIs (any URI in this sample code) may be assumed, and even these may be absent based upon the implementation (e.g. there might be no /redfish/v1/Systems collection on something that doesn't have compute nodes.)

The other URIs must be discovered dynamically by following href links. This is because the API will eventually be implemented on a system that breaks any existing data model "shape" assumptions we may make now. In particular, clients should not make assumptions about the URIs for the resource members of a collection. For instance, the URI

of a collection member will NOT always be /redfish/v1/.../collection/1, or 2. On systems with multiple compute nodes per manager, a System collection member might be /redfish/v1/Systems/C1N1.

This sounds very complicated, but in reality (as these examples demonstrate), if you are looking for specific items, the traversal logic isn't too complicated.

On Resource Model Traversal:

Although the resources in the data model are linked together, because of cross link references between resources, a client may not assume the resource model is a tree. It is a graph instead, so any crawl of the data model should keep track of visited resources to avoid an infinite traversal loop.

A reference to another resource is any property called "href" no matter where it occurs in a resource.

An external reference to a resource outside the data model is referred to by a property called "extref". Any resource referred to by extref should not be assumed to follow the conventions of the API.

On Resource Versions:

Each resource has a "Type" property with a value of the format Tyepname.x.y.z where * x = major version - incrementing this is a breaking change to the schema y = * minor version - incrementing this is a non-breaking additive change to the * schema z = errata - non-breaking change

Because all resources are versioned and schema also have a version, it is possible to design rules for "nearest" match (e.g. if you are interacting with multiple services using a common batch of schema files). The mechanism is not prescribed, but a client should be prepared to encounter both older and newer versions of resource types.

On HTTP POST to create:

When POSTing to create a resource (e.g. create an account or session) the guarantee is that a successful response includes a "Location" HTTP header indicating the resource URI of the newly created resource. The POST may also include a representation of the newly created object in a JSON response body but may not. Do not assume the response body, but test it. It may also be an ExtendedError object.

HTTP REDIRECT:

All clients must correctly handle HTTP redirect. We (or Redfish) may eventually need to use redirection as a way to alias portions of the data model.

FUTURE: Asynchronous tasks

In the future some operations may start asynchonous tasks. In this case, the client should recognized and handle HTTP 202 if needed and the 'Location' header will point to a resource with task information and status.

JSON-SCHEMA:

The json-schema available at /redfish/v1/Schemas governs the content of the resources, but keep in mind: * not every property in the schema is implemented in every implementation. * some properties are schemed to allow both null and another type like string * or integer.

Robust client code should check both the existence and type of interesting properties and fail gracefully if expectations are not met.

GENERAL ADVICE:

Clients should always be prepared for: * unimplemented properties (e.g. a property doesn't apply in a particular case) * null values in some cases if the value of a property is not currently known * due to system conditions HTTP status codes other than 200 OK. Can your code * handle an HTTP 500 Internal Server Error with no other info? URIs are case * insensitive HTTP header names are case insensitive JSON Properties and Enum * values are case sensitive A client should be tolerant of any set of HTTP * headers the service returns

class redfish.main.**ConnectionParameters**
Store connection parameters.

class redfish.main.**RedfishConnection**(*url*, *user*, *password*, *simulator=False*, *enforceSSL=True*, *verify_cert=True*)

Implements basic connection handling for Redfish APIs.

get_api_version()

Return api version.

Returns string – version

Raises AttributeError

10.1.4 mapping class

class redfish.mapping.**RedfishVersionMapping**(*version*, *rootname*)

Implements basic url path mapping beetween Redfish versions.

10.1.5 types class

class redfish.types.**Base**(*url*, *connection_parameters*)

Abstract class to manage types (Chassis, Servers etc...).

get_link_url(*link_type*, *data_subset=None*)

Need to be explained.

Parameters **parameter_name** – name of the parameter

Returns string – parameter value

get_name()

Get root name

Returns string – root name or "Not available"

get_parameter(*parameter_name*)

Generic function to get a specific parameter

Parameters **parameter_name** – name of the parameter

Returns string – parameter value

get_parameters()

Generic function to get all parameters

Returns string – parameter value

set_parameter(*parameter_name*, *value*)

Generic function to set a specific parameter

Parameters

- **parameter_name** – name of the parameter
- **value** – value to set

Returns string – http response of PATCH request

class redfish.types.**BaseCollection**(*url*, *connection_parameters*)

Abstract class to manage collection (Chassis, Servers etc...).

class redfish.types.**Device**(*url*, *connection_parameters*)

Abstract class to add common methods between devices (Chassis, Servers, System).

get_asset_tag()
Get asset tag of the device.

Returns asset tag or "Not available"

Return type string

get_fw_version()
Get firmware version of the device.

Returns firmware version or "Not available"

Return type string

get_manufacturer()
Get device manufacturer

Returns device manufacturer or "Not available"

Return type string

get_model()
Get device model

Returns device model or "Not available"

Return type string

get_name()
Get name of the device.

Returns name or "Not available"

Return type string

get_part_number()
Get part number of the device.

Returns part number or "Not available"

Return type string

get_serial_number()
Get serial number of the device.

Returns serial number or "Not available"

Return type string

get_sku()
Get sku number of the device.

Returns sku number or "Not available"

Return type string

get_status()
Get device status

Returns device status or "Not available"

Return type dict

get_uuid()
Get device uuid

Returns device uuid or "Not available"

Return type string

10.2 redfish-client classes

redfish-client This is a client using the python-redfish library to retrieve and perform action on redfish compatible systems.

class rfclient.**InventoryFile**(*inventory_file*)
redfisht-client inventory file management

add_manager(*manager_name*, *url*, *login*, *password*)
Add a manager to the configuration file

Parameters

- **manager_name** (*str*) – Name of the manager
- **url** (*str*) – Url of the manager
- **login** (*str*) – Login of the manager
- **password** (*str*) – Password of the manager

check_manager(*manager_name*)
Check if the manager exists in configuration file

Parameters **manager_name** (*str*) – Name of the manager

delete_manager(*manager_name*)
Delete manager

Parameters **manager_name** (*str*) – Name of the manager

Returns Nothing

get_manager_info(*manager*)
Show manager info (url, login, password)

Parameters **manager** (*str*) – Name of the manager

Returns info containing url, login, password

get_managers()
Get manager configured

Returns Managers

manager_incorect(*exception*)
Log and exit if manager name is incorect

modify_manager(*manager_name*, *parameter*, *parameter_value*)
Modify the manager settings

Parameters

- **manager_name** (*str*) – Name of the manager
- **parameter** – url | login | password
- **parameter_value** (*str*) – Value of the parameter

Returns Nothing

save()
Save the configuration file data

exception rfclient.**RedfishClientException**(*message=None*, ***kwargs*)
Base class for redfish client exceptions

CHAPTER

ELEVEN

CONTRIBUTING

If you would like to contribute to the development of this project, please consider reading https://wiki.openstack.org/wiki/How_To_Contribute.

Submit your issues to https://bugs.launchpad.net/python-redfish/+filebug.

In order to submit a patch, git clone the project at https://github.com/openstack/python-redfish.git Make your modifications and then make a git review for it which will be followed on https://review.openstack.org/#/q/project:openstack/python-redfish

You can also share and discuss on the mailing list as well at http://mondorescue.org/sympa/arc/python-redfish.

CHAPTER

TWELVE

FAQ

- Q1 : error in setup command: Invalid environment marker: (python_version < ’3’)

  This error is caused by an old setuptools version that does not understand "python_version < ’3’". Upgrade setuptools using:

```
pip install --upgrade setuptools
```

CHAPTER

THIRTEEN

HELP REQUIRED

We need help on the following topic:

- debian/ubuntu dependencies packaging.
- installation on distributions which are not Fedora or Mageia.
- documentation.

Any contribution will be welcomed.

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX