RESTful Server Configuration with iDRAC RESTful API

Dell EMC Customer Solutions Center

November 2017

Authors

Texas Roemer, Sr. Test Engineer (Dell EMC Server Solutions)
Paul Rubin, Sr. Product Manager (Dell EMC Server Solutions)
Jonas Werner, Sr. Solutions Architect (Dell EMC Customer Solutions Center)

A Dell EMC Technical White Paper

Revisions

Date	Description
June 2016	Initial release
June 2017	Updated for 14th generation PowerEdge servers
October 2017	Updated for iDRAC7/8 2.50.50.50 firmware
November 2017	Corrected typos in section 2.7

Contents

Executive summary

The growing scale of cloud- and web-based data center infrastructure is reshaping the needs of IT administration worldwide. New approaches to systems management are needed to keep up with a growing and changing market.

To accommodate the needs for efficient systems management automation, Dell has developed the integrated Dell Remote Access Controller (iDRAC) with Lifecycle Controller RESTful API with support for the Distributed Management Task Force (DMTF) Redfish standard. Together, the iDRAC RESTful API and Redfish enable the automation of scalable and secure management. One of the latest enhancements to the iDRAC RESTful API is support for RESTful server configuration with Server Configuration Profiles (SCP). Using the iDRAC SCP RESTful API, administrators can obtain the configuration details of 12th, and 13th, and 14th generation Dell PowerEdge servers, preview the application of a configuration file to a server, and apply configuration files to establish BIOS, iDRAC, PERC RAID controller, NIC, and HBA settings.

This document provides an overview of the iDRAC RESTful API, the Redfish Scalable Platforms Management API standard, and details the use of the iDRAC SCP RESTful API for RESTful configuration management of PowerEdge servers.

1 Introduction

As the scale of deployment has grown for x86 servers, IT administrators have seen their scope expand from managing a handful of servers to hundreds or even thousands of servers. The deployment scale and the IT models have changed – from physical to virtual, from on-premises to cloud to hybrid cloud – leading to wholesale changes in the tools and processes of IT management.

In response to these changes, Dell and the industry have developed new systems management automation methods and standards that utilize web and cloud computing technologies. Among these technologies, APIs using the Representational State Transfer (REST) architecture, such as the Distributed Management Task Force (DMTF) Redfish standard, are becoming key enablers for efficient automation of server deployment and update.

The heart of embedded management automation in every Dell PowerEdge server—the iDRAC with Lifecycle Controller— provides the ability to generate a human-readable snapshot of server configuration via a Server Configuration Profile (SCP). This single file contains all BIOS, iDRAC, Lifecycle Controller, Network, and Storage settings. After capture, this file can be modified as needed, and applied to other servers, including different server models. The iDRAC has supported export, preview, and import operations for SCP using the WS-Man API and RACADM command line interface since the 12th generation of PowerEdge servers. With the version 2.40.40.40 firmware update or later, these operations are also supported using iDRAC RESTful API extensions, enabling RESTful configuration of all settings for 12th, 13th and 14th generation PowerEdge servers.

This whitepaper provides an overview of the iDRAC RESTful API and Redfish standard and illustrates the practical use of the RESTful Server Configuration Profile features:

- Showing how to clone or replace settings from a designated source or “golden” server
- Preview applying these settings
- Importing the settings to a target server.

1.1 DMTF Redfish Standard

There are various out-of-band systems management standards available in the industry today. However, there is no single standard that can be easily used within emerging programming standards, readily implemented within embedded systems, and meet the demands of today’s evolving IT solution models.

Emerging IT solutions models have placed new demands on systems management solutions to support expanded scale, higher security, and multi-vendor openness while also aligning with modern DevOps tools and processes.

Recognizing these needs, Dell and other IT solutions leaders within the DMTF undertook the creation of a new management interface standard. After a multi-year effort, the new standard, Redfish v1.0, was announced in July, 2015.

Its key benefits include:

- Increased simplicity and usability
- Encrypted connections and heightened security

- A programmatic interface that can be easily controlled through scripts
- Ability to meet the Open Compute Project's Remote Machine Management requirements
- Based on widely used standards for web APIs and data formats

Redfish has been designed to support the full range of server architectures from monolithic servers to converged infrastructure and hyper scale architecture. The Redfish data model, which defines the structure and format of data representing server status, inventory, and available operational functions, is vendor-neutral. Administrators can then create management automation scripts that can manage any Redfish compliant server. This is crucial for the efficient operation of a heterogeneous server fleet.

Using Redfish also has significant security benefits—unlike legacy management protocols, Redfish utilizes HTTPS encryption for secure and reliable communication. All Redfish network traffic, including event notifications, can be sent encrypted across the network.

Redfish provides a highly organized and easily accessible method to interact with a server using scripting tools. The web interface employed by Redfish is supported by many programming languages and its tree-like structure makes information easier to locate. Data returned from a Redfish query can be turned into a searchable dictionary consisting of key-value-pairs. By looking at the values in the dictionary, it is easy to locate settings and current status of a system managed by Redfish. These settings can then be updated and actions can be issued to one or multiple systems.

Since its July, 2015 introduction, Redfish has continued to grow and evolve with specification updates released in 2016 covering key operations such as BIOS configuration, server firmware update, and detailed server inventory.

1.2 iDRAC with Lifecycle Controller RESTful API

To support the DMTF Redfish standard, the iDRAC with Lifecycle Controller has been enhanced to support a RESTful API in addition to the current support for the IPMI, SNMP, and WS-Man standard APIs. The iDRAC RESTful API builds upon the Redfish standard to provide a RESTful interface for Dell value-add operations including

- Information on all iDRAC with Lifecycle Controller out-of-band services—web server, SNMP, virtual media, SSH, Telnet, IPMI, and KVM
- Expanded storage subsystem reporting covering controllers, enclosures, and drives
- For the PowerEdge FX2 modular, detailed chassis information covering power supplies, temperatures, and fans
- With the iDRAC Service Module installed under the server OS, the API provides detailed inventory and status reporting for host network interfaces including such details as IP address, subnet mask, and gateway for the Host OS

The following sections provide details concerning the iDRAC RESTful API interface calls that enable RESTful configuration of PowerEdge servers with Server Configuration Profiles (SCP).

1.3 Configuring servers with Server Configuration Profiles

Beginning with 12th generation PowerEdge servers, iDRAC with Lifecycle Controller has featured the use of Server Configuration Profiles (SCP) to configure BIOS, iDRAC, PERC RAID controller, and NIC/HBA settings in a single file. This greatly simplifies bare metal deployments and scale out operations by removing complexity from server configuration. Rather than manually interacting with BIOS F2/F10 screens or writing complex scripts, administrators can set up an initial “gold” configuration server, capture the settings into an SCP file, modify the profile as needed, and apply the profile across a pool of target servers. Beginning with iDRAC7/8 firmware version 2.50.50.50 and iDRAC9 firmware version 3.00.00.00, SCP files can be expressed in either XML or JSON format. Figure 1 illustrates an SCP XML format file:

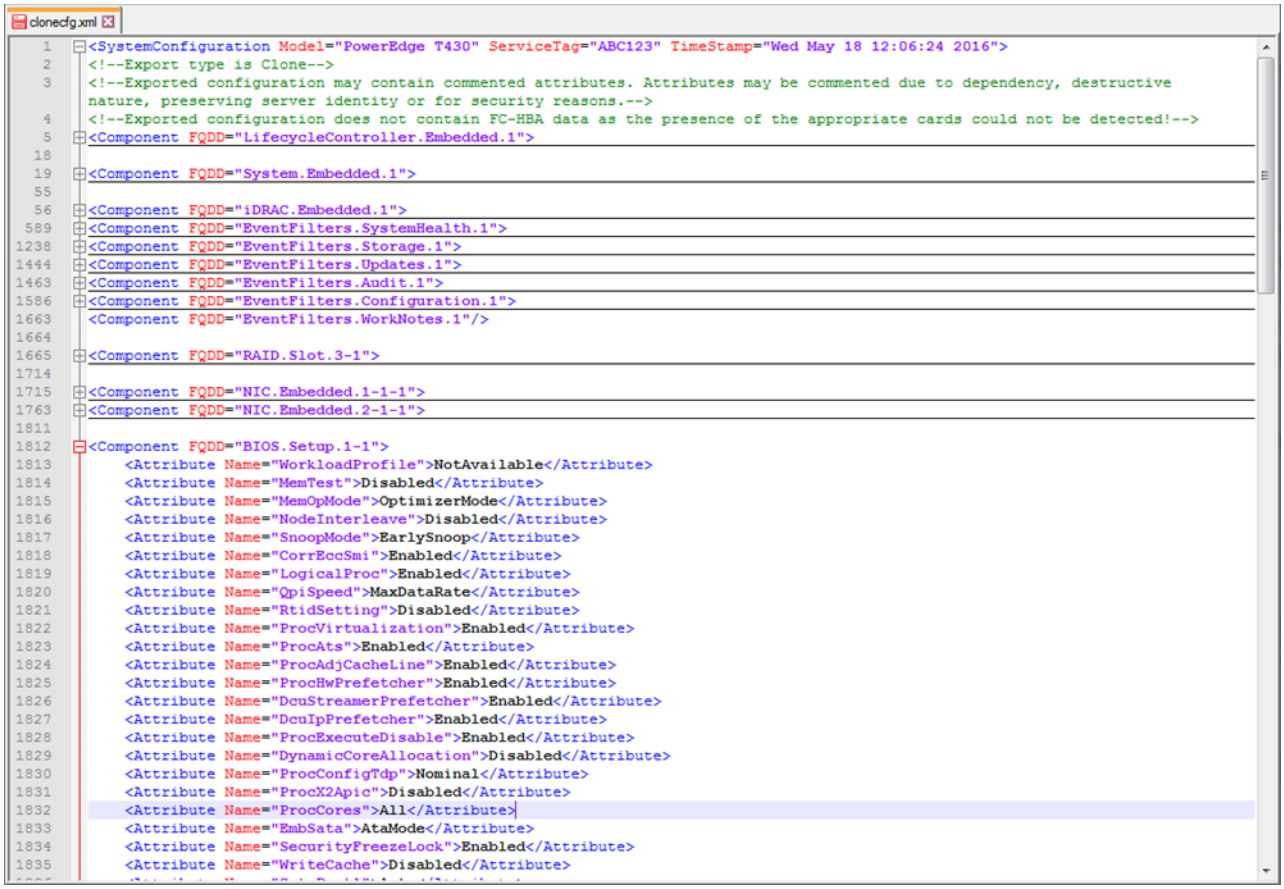


Figure 1 Viewing an SCP XML file

iDRAC with Lifecycle Controller supports a range of mechanisms for SCP file operations. These mechanisms include:

- iDRAC USB Direct: Imports an SCP file from a USB memory key attached to the iDRAC USB port
- Zero Touch Auto Config: Imports an SCP file from a network share when the server is attached to the network
- iDRAC RESTful and WS-Man APIs: Provide application calls for exporting, previewing import and importing SCP files from a network share

- PowerEdge PowerShell cmdlets: Support WS-Man SCP operations via PowerShell scripting
- RACADM CLI: Provides SCP export, preview, and import operations via local SCP files in addition to network share-based SCP files

For details on these features, see the **Additional Information** section.

1.3.1 New SCP features for iDRAC7 or iDRAC8, and iDRAC9

The introduction of 14th generation PowerEdge servers with iDRAC9 firmware 3.00.00.00 includes enhancements to SCP operations including:

- SCP operations via HTTP, HTTPS, and via local file streaming in addition to CIFS and NFS
- Server firmware update from repository as part of SCP import
- SCP export and import in JSON format in addition to XML
- iDRAC9 GUI SCP page for interactive SCP export and import
- Auto Config support via HTTP and HTTPS in addition to CIFS and NFS and support for JSON format in addition to XML

Note: when performing HTTPS-based SCP operations with the iDRAC RESTful API with Redfish, certificate warnings will be ignored; there is not currently an option to force certificate checking.

iDRAC7/iDRAC8 with firmware version 2.50.50.50 or later includes these SCP feature enhancements:

- SCP operations by using local file streaming in addition to CIFS and NFS
- Server firmware update from repository as part of the SCP import process
- SCP export and import in JSON format in addition to the XML file format

Details are included below on these enhancements for RESTful server configuration.

2 Using iDRAC RESTful API with Server Configuration Profiles

To create a Server Configuration Profile file for import to a new or to-be-reprovisioned server, create a "golden configuration" on a PowerEdge server. After the server is configured as needed, it can act as a template for other servers that need to use the same or similar server settings. To use these settings, an SCP file must be created by exporting the current server settings to a file. The exported SCP file can be edited, as needed, and used to either preview an import to a target server or be actually imported to configure a target server.

Note: Ensure that the RESTful API is enabled for the iDRAC with Lifecycle Controller before you perform any of the actions provided in the below scripts. For help in verifying and enabling the RESTful API, see Appendix 1.

Let us now see how the iDRAC RESTful API SCP operations can be automated with the use of the Python scripting language. Before explaining the SCP RESTful APIs, review the RACADM Command Line Interface SCP operations. Following are examples of RACADM commands to export an SCP file from a server and place the results into a CIFS or NFS network share:

```
# racadm get –f serverscp –t xml –u <CIFSuser> -p <CIFSpassword> -l
<CIFS Share path>

# racadm get –f serverscp –t xml –l <NFS Share path>
```

For 14th generation PowerEdge servers, additional RACADM options include:

```
# racadm –get –f serverscp –t JSON –u <HTTP/S user> -p <HTTP/S password> -l
<HTTP/S Share path>
```

In addition to the required parameters—SCP file name, network share pathname, and for CIFS share access, a valid CIFS username and password—you can specify a few more optional parameters. These parameters include:

- Selectors to include specific server component configuration information such as BIOS only
- Option to hash encode exported passwords
- Option to produce an SCP file in a form that clones the source server by not duplicating settings that uniquely identify the server such as server service tag, or to produce an SCP file that can replace the source server by duplicating all settings, including identifying settings.

Appendix 2 illustrates the available SCP export, preview, and import options available using the iDRAC SCP RESTful API. This JSON-formatted output is produced using any iDRAC-supported web browser to access the following URL:

https://<iDRAC IP> /redfish/v1/Managers/iDRAC.Embedded.1

In the Appendix 2 JSON output, under the "OEM" section, are the supported methods for SCP operations including **ExportSystemConfiguration**, **ImportSystemConfiguration** and **ImportSystemConfigurationPreview**. The required and optional parameters are detailed within each method.

2.1 Preparing to use SCP ExportSystemConfiguration method

This section provides details about the creation of Python scripts for exporting a system configuration to a CIFS share by using a RESTful POST command. Before creating the Python script, two specific Python modules are needed: **requests** and **json**. If your version of Python does not have these modules installed, use “make-install” to install them.

To begin the script, compile the URL for the POST command. From Appendix 2, view the JSON output for URL “https://<iDRAC IP>/redfish/v1/Managers/iDRAC.Embedded.1”, look within the OEM **ExportSystemConfiguration** method and find “Target”. This URL is assigned to the variable “url” for the POST command. For example:

```
url =
https://<iDRACIP>/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.Export
SystemConfiguration
```

Create a header that acts as a dictionary that specifies that the content type of the POST operation will be JSON:

```
headers = {'content-type': 'application/json'}
```

Next, compile a payload dictionary for the parameters that will be provided for the **ExportSystemConfiguration** method. **ShareParameters** must be a nested dictionary within a dictionary; this nesting is visible within the JSON output in Appendix 2:

```
payload =
{"ExportFormat":"XML","ShareParameters":{"Target":"ALL","IPAddress":"192.168.0.1
30","ShareName":"cifs_share","ShareType":"CIFS","UserName":"<cifs
username>","Password":"<cifs password>","FileName":"R740xd_SCP.xml"}}
```

This payload indicates that:

- The SCP will be exported in XML format
- All possible server configuration components—BIOS, iDRAC, PERC, NIC, and HBA—will be exported
- Specifies the address of the source server iDRAC
- Provides the CIFS share pathname, file name, and credentials to access the CIFS share

Compile the POST command by passing in the URL, payload, and header. Assign this command to a variable which will be used to parse the data, check the status code, and get the job ID for the SCP export job. This script uses Basic Authentication and passes the required iDRAC administrator credentials:

```
response = requests.post(url, data=json.dumps(payload), headers=headers,
verify=False, auth=('username','password'))
```

Using the above, here is an example Python script to export the SCP file to a CIFS share:

Script – redfish_SCP_export_cifs.v1.py: Version 1

```
#
# Version 1 - Python iDRAC RESTful API script to export
#             server configuration profile
#
import requests, json
url = 'https://192.168.0.120/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemConfiguration'
payload = {"ExportFormat":"XML","ShareParameters":{"Target":"ALL","IPAddress":"192.168.0.130","ShareName":"cifs_share","ShareType":"CIFS","UserName":"user","Password":"password","FileName":"R740xd_SCP.xml"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False,   auth=('username','password'))
print '- Response status code is: %s' % response.status_code
response_output=response.__dict__
print response_output
```

2.2 Executing iDRAC RESTful API SCP export script

We will now execute the SCP export Python script and parse the data returned. The key information needed from the output are the “status code” returned by the export request and the “Job ID” – when an SCP export is requested, the iDRAC will create an asynchronous job to gather all of the requested settings from the server components, assemble the results into a file, and write the file to the target network share. The job will be monitored to determine its successful completion.

```
$ python ./redfish_SCP_export_cifs.v1.py
- Response status code is: 202
{'cookies': <<class 'requests.cookies.RequestsCookieJar'>[]>, '_content': '',
'headers': {'Content-Length': '0', 'Keep-Alive': 'timeout=60, max=100',
'Server': 'Apache/2.4', 'Connection': 'Keep-Alive', 'Location':
'/redfish/v1/TaskService/Tasks/JID_967978014801', 'Cache-Control': 'no-cache',
'Date': 'Wed, 07 Jun 2017 01:10:01 GMT', 'OData-Version': '4.0', 'Access-
Control-Allow-Origin': '*', 'Content-Type':
'application/json;odata.metadata=minimal;charset=utf-8', 'X-Frame-Options':
'DENY'}, 'url':
u'https://192.168.0.120/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674
_Manager.ExportSystemConfiguration', 'status_code': 202, '_content_consumed':
True, 'encoding': 'utf-8', 'request': <PreparedRequest [POST]>, 'connection':
<requests.adapters.HTTPAdapter object at 0x7ffa89d5f290>, 'elapsed':
datetime.timedelta(0, 1, 102737), 'raw':
<requests.packages.urllib3.response.HTTPResponse object at 0x7ffa89d8f290>,
'reason': 'Accepted', 'history': []}
$
```

A status code return of “202” indicates the SCP export request was successful. For more information on possible returned status code values, consult the Dell Redfish API Reference Guide.

Notice that Job ID is a value nested within a dictionary of a dictionary. We will now modify the script to access the dictionary and then parse the value using regular expressions to obtain the Job ID. The added code for this purpose is highlighted in the following script:

Script – redfish_SCP_export_cifs.v2.py version 2:

```
#
# Version 2 - Python iDRAC RESTful API script to export
#             server configuration profile
#
import requests, json, re
url = 'https://192.168.0.120/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemConfiguration'
payload = {"ExportFormat":"XML","ShareParameters":{"Target":"ALL","IPAddress":"192.168.0.130",
"ShareName":"cifs_share","ShareType":"CIFS","UserName":"user","Password":"password","FileName": "R730_SCP.xml"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False,     auth=('username','password'))
print '- Response status code is: %s' % response.status_code
response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
print "- Job ID is: %s" % job_id
```

Now, execute version 2 and view the returned status code and Job ID:

```
$ python ./redfish_SCP_export_cifs.v2.py
- Response status code is: 202
- Job ID is: JID_967983367454
$
```

After a successful export request operation, wait till the export job is complete. This is performed by querying the RESTful API Task Service until the job completes successfully or indicates an error. We will create a script to check the job status and then run the script with the Job ID as an input parameter:

Script: rest_SCP_get_job_status.py

```
import requests, sys
job_id=sys.argv[1]
req = requests.get('https://192.168.0.120/redfish/v1/TaskService/Tasks/%s' % (job_id),   auth=("username", "password"), verify=False)
statusCode = req.status_code
print "- Status code is: %s" % statusCode
data = req.json()
message_string=data[u"Messages"]
print "- Job ID = "+data[u"Id"]
print "- Name = "+data[u"Name"]
print "- Message = "+message_string[0][u"Message"]
print "- JobStatus = "+data[u"TaskState"]
```

```
$ python ./redfish_SCP_get_job_status.py JID_967983367454
- Status code is: 200
- Job ID = JID_967983367454
- Name = Export: Server Configuration Profile
- Message = Successfully exported Server Configuration Profile
- JobStatus = Completed
$
```

Now we combine and enhance the scripts to perform the SCP export and await completion of the export job:

Script – redfish_SCP_export_cifs.v3.py version 3:

```
# Python script using Redfish API to perform iDRAC feature
# Server Configuration Profile (SCP) for export only

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemCon
figuration' % idrac_ip

# For payload dictionary supported parameters, refer to schema
# "https://'iDRAC IP'/redfish/v1/Managers/iDRAC.Embedded.1/"

payload = {"ExportFormat":"XML","ExportUse":"Default","ShareParameters":{"Target":"LifecycleContro
ller","IPAddress":"192.168.0.130","ShareName":"smb_share","ShareType":"CIFS","FileName":file,"User
Name":"username","Password":"password"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n##### Command Failed, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
```

```
    print "\n- %s successfully created for ExportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in data[u"Messages"] or "completed with errors" in data[u"Messages"]:
        print "Job failed, current message is: %s" % data[u"Messages"]
        sys.exit()
    elif data[u"TaskState"] == "Completed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif data[u"TaskState"] == "Completed with Errors" or data[u"TaskState"] == "Failed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue



data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

```
$ python ./redfish_SCP_export_cifs.v3.py 192.168.0.120 root calvin
jwr_rf_exp.v3_02.xml

- JID_967992694673 successfully created for ExportSystemConfiguration method

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Exporting Server Configuration Profile.

- Query job ID command passed

Job ID = JID_967992694673
Name = Export: Server Configuration Profile
Message = Successfully exported Server Configuration Profile
JobStatus = Completed

JID_967992694673 completed in: 0:00:13
$
```

2.3 Exporting JSON SCP to an HTTP(S) share with iDRAC9

Serving SCP files from a web server can be a useful way to provide easy-access to configuration files for a fleet of servers. The following script will export the SCP to a web server and takes the following as input variables:

- iDRAC IP address,
- Method or action (Import or Export)
- A file name for the JSON or XML file to be imported / exported

Hardcoded values are present for:

- The iDRAC username and password
- The Web server IP address
- The Web server folder name
- Data format (Currently JSON but can be set to XML)

Note that it is exporting in Default mode which will result in a non-destructive SCP file when imported into another server. Alternatively Clone or Replace can be used to enable settings which may destroy data – for example by reconfiguring RAID volumes.

Either HTTP or HTTPS can be used; for this example HTTP is hardcoded in the script.

Script – redfish_SCP_export_http.py

```
# Python script using Redfish API to perform iDRAC feature
# Server Configuration Profile (SCP) for export to HTTP share only

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemCon
figuration' % idrac_ip

# For payload dictionary supported parameters, refer to schema
# "https://'iDRAC IP'/redfish/v1/Managers/iDRAC.Embedded.1/"

payload = {"ExportFormat":"JSON","ExportUse":"Default","ShareParameters":{"Target":"All","IPAddres
s":"192.168.0.130","ShareName":"webshare","ShareType":"HTTP","FileName":file}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n##### Command Failed, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- %s successfully created for ExportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
```

```
    data = req.json()
    message_string=data[u"Messages"]
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in data[u"Messages"] or "completed with errors" in data[u"Messages"]:
        print "Job failed, current message is: %s" % data[u"Messages"]
        sys.exit()
    elif data[u"TaskState"] == "Completed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif data[u"TaskState"] == "Completed with Errors" or data[u"TaskState"] == "Failed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue



data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

```
$ python ./redfish_SCP_export_http.py 192.168.0.120 root calvin
jwr_rf_exp_http_01.xml

- JID_968007336828 successfully created for ExportSystemConfiguration method

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Exporting Server Configuration Profile.
```

```
- Query job ID command passed

Job ID = JID_968007336828
Name = Export: Server Configuration Profile
Message = Successfully exported Server Configuration Profile
JobStatus = Completed

JID_968007336828 completed in: 0:00:13
$
```

2.4 Exporting SCP to a streamed local file

Exporting SCP directly to a local file is supported with iDRAC7/iDRAC8 with firmware version 2.50.50.50 or later, and with iDRAC9 firmware version 3.00.00.00 or later. In the following example, the SCP values are streamed to a local file on the client executing the script. The required parameters are the iDRAC IP address, and iDRAC admin username and password. The export format will be XML.

NOTE: The “Target” setting picks the subsection of interest. In this case it is set to “BIOS” but it could be changed to export settings for “RAID”, “NIC”, “iDRAC”, etc. This can be very useful to select specific sections to export and subsequently import on another system. There is no need to clone the entire server if a targeted approach will meet the need.

Script – redfish_SCP_export_local.py

```
# Python script using Redfish API to export SCP attributes to a local file

import requests, json, sys, re, time
from datetime import datetime

try:
    idrac_ip       = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
except:
    print "-
FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC password"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemCon
figuration' % idrac_ip
payload = {"ExportFormat":"XML","ShareParameters":{"Target":"BIOS"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

if response.status_code != 202:
    print "- FAIL, status code not 202\n, code is: %s" % response.status_code
    print "- Error details: %s" % response.__dict__
    sys.exit()
else:
    print "\n- Job ID successfully created for ExportSystemConfiguration method\n"
```

```
response_output=response.__dict__
job_id=response_output["headers"]["Location"]

try:
    job_id=re.search("JID_.+",job_id).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

start_time=datetime.now()

while True:
    current_time=(datetime.now()-start_time)
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    d=req.__dict__
    if "<SystemConfiguration Model" in str(d):
        print "\n- Export locally successfully passed. Attributes exported:\n"
        zz=re.search("<SystemConfiguration.+</SystemConfiguration>",str(d)).group()

        #Below code is needed to parse the string to set up in pretty XML format
        q=zz.replace("\\n"," ")
        q=q.replace("<!--  ","<!--")
        q=q.replace(" -->","-->")
        del_attribute='<Attribute Name="SerialRedirection.1#QuitKey">^\\\\</Attribute>'
        q=q.replace(del_attribute,"")
        l=q.split("> ")
        export_xml=[]
        for i in l:
            x=i+">"
            export_xml.append(x)
        #export_xml=re.sub(">> \n",">",export_xml)
        export_xml[-1]="</SystemConfiguration>"
        d=datetime.now()
        filename="%s-%s-%s_%s%s%s_export.xml"% (d.year,d.month,d.day,d.hour,d.minute,d.second)
        f=open(filename,"w")
        for i in export_xml:
            f.writelines("%s \n" % i)
        f.close()
        for i in export_xml:
            print i

        print "\n"
        req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth
=(idrac_username, idrac_password), verify=False)
        data = req.json()
        message_string=data[u"Messages"]
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        print "Message = "+message_string[0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        print "\nExported attributes also saved in file: %s" % filename
        sys.exit()
    else:
        pass
```

```
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    current_time=(datetime.now()-start_time)

    if statusCode == 202 or statusCode == 200:
        print "\n- Execute job ID command passed, checking job status...\n"
        time.sleep(1)
    else:
        print "Execute job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if str(current_time)[0:7] >= "0:10:00":
        print "\n-
FAIL, Timeout of 10 minutes has been reached before marking the job completed."
        sys.exit()

    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue

data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
```

Executing the script results in the attributes being printed to standard output and also saved to a local XML file named with the current date and time.

```
python ./redfish_SCP_export_local.py 192.168.0.120 root calvin
- Job ID successfully created for ExportSystemConfiguration method
- Export locally successfully passed. Attributes exported:
```

```
<SystemConfiguration Model="PowerEdge C6420" ServiceTag="BLDXXXX" TimeStamp="Tue Jun  6 14:12:21 2
017">
<!--Export type is Normal,XML,Selective-->
<!--
Exported configuration may contain commented attributes. Attributes may be commented due to depend
ency, destructive nature, preserving server identity or for security reasons.-->
<Component FQDD="BIOS.Setup.1-1">
    <Attribute Name="WorkloadProfile">NotAvailable</Attribute>
    <Attribute Name="MemTest">Disabled</Attribute>
    <Attribute Name="MemOpMode">OptimizerMode</Attribute>
    <Attribute Name="NodeInterleave">Disabled</Attribute>
    <Attribute Name="CorrEccSmi">Enabled</Attribute>
    <Attribute Name="OppSrefEn">Disabled</Attribute>
    <Attribute Name="LogicalProc">Enabled</Attribute>
    <Attribute Name="CpuInterconnectBusSpeed">MaxDataRate</Attribute>
    <Attribute Name="ProcVirtualization">Enabled</Attribute>
    <Attribute Name="ProcAdjCacheLine">Disabled</Attribute>
```

```
<Output shortened for brevity>
```

```
</Component>
</SystemConfiguration>

Job ID = JID_963619479070

Name = Export: Server Configuration Profile
Message = Successfully exported Server Configuration Profile
JobStatus = Completed
JID_963619479070 completed in: 0:00:04

Exported attributes are also saved in file: 2017-6-2_11555_export.xml
```

2.5 Previewing SCP imports with iDRAC RESTful API

Before importing an SCP to apply configuration settings to a server, it is recommended to preview the SCP file using the **ImportSystemConfigurationPreview** method. The following Python script will preview the import of an SCP file stored on a CIFS share with the iDRAC IP address, iDRAC admin credentials and the SCP file name passed as arguments.

Script: redfish_SCP_import_preview.py

```
import requests, json, re, sys, time

from datetime import datetime


try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()


url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemCon
figurationPreview' % idrac_ip

payload = {"ShareParameters":{"Target":"ALL","IPAddress":"192.168.0.130","ShareName":"cifs_share",
"ShareType":"CIFS","FileName":file,"UserName":"cifs_user","Password":"cifs_password"}}

headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
```

```
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n- FAIL, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- PASS, %s successfully created for ImportSystemConfigurationPreview method\n" % (job
_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    final_message_string=str(message_string)
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- PASS, Query job ID command passed"
    else:
        print "\n- FAIL, Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in final_message_string or "completed with errors" in final_message_string or "Not
 one" in final_message_string or "Unable" in final_message_string:
        print "\n- FAIL, detailed job message is: %s" % data[u"Messages"]
        sys.exit()

    if data[u"TaskState"] == "Completed":
        print "\n- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        print "- JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        print "\n- Preview Details: %s" % message_string
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue



data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

Next, execute the script to obtain the status and Job ID; as with the Export and Import methods, the Preview method creates a job that must be queried until completed to determine success or failure of the import preview:

```
python ./redfish_SCP_import_preview.py 192.168.0.120 root calvin
jwr_rf_exp.v3_02.xml

- PASS, JID_968020063229 successfully created for
ImportSystemConfigurationPreview method

- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.

- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.

- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.

- PASS, Query job ID command passed

- Job ID = JID_968020063229
- Name = Preview Configuration
- JobStatus = Completed

JID_968020063229 completed in: 0:00:07

- Preview Details: [{u'Message': u'Estimated time for applying configuration
changes is 10 seconds.', u'MessageID': u'SYS088', u'MessageArgs': [],
u'MessageArgs@odata.count': 0}, {u'Message': u'Successfully previewed Server
Configuration Profile import operation.', u'MessageId': u'SYS081',
u'MessageArgs': [], u'MessageArgs@odata.count': 0}]
```

When the import preview operation is complete, you can view a report that indicates whether a server reboot is required after the SCP import and also provides the estimated time needed for the configuration operation. To view the report, use the following RACADM command:

```
$ racadm -r 192.168.0.120 -u root -p calvin lclog viewconfigresult -j
JID_968020063229
SeqNumber       = 34729
Job Name        = Preview Configuration
Message ID      = SYS088
Message         = Estimated time for applying configuration changes is 10
seconds.
SeqNumber       = 34726
FQDD            = LifecycleController.Embedded.1
Job Name        = Preview Configuration
```

Note: **viewconfigresults** not yet available via the RESTful API.

The following example shows a preview involving an SCP file that contains an error. The preview script that generates an error will be run and then we will use RACADM to obtain details on the error:

```
python ./redfish_SCP_import_preview.py 100.65.99.196 root calvin
jwr_rf_exp.v3_02.xml

- PASS, JID_968022511916 successfully created for
ImportSystemConfigurationPreview method


- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.


- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.


- PASS, Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Previewing Server Configuration Profile.


- PASS, Query job ID command passed

- Job ID = JID_968022511916
- Name = Preview Configuration
- JobStatus = Completed

JID_968022511916 completed in: 0:00:07

- Preview Details: [{u'Message': u'Estimated time for applying configuration
changes is 0 seconds.', u'MessageID': u'SYS088', u'MessageArgs': [],
u'MessageArgs@odata.count': 0}, {u'Message': u'Not one of the Possible Values
for Attribute', u'Severity': u'Critical', u'Oem': {u'Dell': {u'@odata.type':
u'#DellManager.v1_0_0.ServerConfigurationProfileResults', u'Name':
u'LCAttributes.1#CollectSystemInventoryOnRestart', u'ErrCode': u'9240',
u'OldValue': u'Enabled', u'DisplayValue': u'Collect System Inventory on
Restart', u'NewValue': u'this_is_an_invalid_value'}}, u'MessageID': u'RAC015'},
{u'Message': u'Preview of Server Configuration Profile import operation
indicated that no configuration changes will be successful.', u'MessageId':
u'SYS080', u'MessageArgs': [], u'MessageArgs@odata.count': 0}]
```

Note that the return message accurately states that the value that was passed is "`Not one of the Possible Values for Attribute`"

If racadm is used to get more detail of the error, we get the following output

```
$ racadm -r 192.168.0.120 -u user -p password lclog viewconfigresult -j
JID_968022511916
SeqNumber       = 34743
Job Name        = Preview Configuration
Message ID      = SYS088
Message         = Estimated time for applying configuration changes is 0
seconds.
SeqNumber       = 34741
FQDD            = LifecycleController.Embedded.1
Job Name        = Preview Configuration
DisplayValue    = Collect System Inventory on Restart
Name            = LCAttributes.1#CollectSystemInventoryOnRestart
OldValue        = Enabled
NewValue        = this_is_an_invalid_value
MessageID       = RAC015
Status          = Failure
ErrCode         = 9240
$
```

2.6 Importing SCPs with iDRAC RESTful API

Next, we will create a script for importing SCPs with the **ImportSystemConfiguration** method. The **ImportSystemConfiguration** method provides additional optional parameters as shown in the JSON output in Appendix 2:

```
HostPowerState@ Redfish.AllowableValues: [
  "On",
  "Off"
],
ImportSystemConfiguration@ Redfish.AllowableValues: [
  "TimeToWait",
  "ImportBuffer"
],
ShutdownType@ Redfish.AllowableValues: [
  "Graceful",
  "Forced",
  "NoReboot"
]
```

These parameters control the following:

- Whether the server is powered on (default) or powered off after the SCP import is completed
- Specify a timeout period when awaiting an OS graceful shutdown (default is 1800 seconds)
- Indicate the type of server shutdown to be performed after the SCP is imported
  - Graceful OS shutdown (default)
  - Forced OS shutdown
  - No reboot is to be performed, postponing application of the SCP import until a server shutdown is later commanded

Note: Depending on the type of configuration object being modified by an SCP import, the new values could be applied immediately without a reboot or could require staging and a reboot of the server to apply the new values. Object groups that support immediate update include **LifecycleController**, **System, iDRAC**, and **EventFilters**. For more details on immediate and staged updates, see the RACADM CLI Guide available at **dell.com/idracmanuals**. The script parameters include iDRAC IP address, iDRAC admin username and password and file name of SCP file; the script will import from a CIFS share.

Script: redfish_SCP_import_cifs.py

```
# Python script using Redfish API to perform iDRAC feature
# Server Configuration Profile (SCP) for import only

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemCon
figuration' % idrac_ip

# For payload dictionary supported parameters, refer to schema
# "https://'iDRAC IP'/redfish/v1/Managers/iDRAC.Embedded.1/"

payload = {"ShutdownType":"Forced","ShareParameters":{"Target":"All","IPAddress":"192.168.0.130","
ShareName":"cifs_share","ShareType":"CIFS","FileName":file,"UserName":"cifs_user","Password":"cifs
_password"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
```

```
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n- FAIL, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- %s successfully created for ImportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    final_message_string=str(message_string)
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in final_message_string or "completed with errors" in final_message_string or "Not
 one" in final_message_string or "Unable" in final_message_string:
        print "\n- FAIL, detailed job message is: %s" % data[u"Messages"]
        sys.exit()
    elif "Successfully imported" in final_message_string or "completed with errors" in final_messa
ge_string or "Successfully imported" in final_message_string:
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif "No changes" in final_message_string:
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue
```

```
data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

Now we will execute the above import script. After the import job is marked complete, failed, or completed with errors, we will execute the RACADM command **lclog viewconfigresult** to get more information on the import job.

```
$ python ./redfish_SCP_import_cifs.py 192.168.0.120 root calvin
jwr_rf_exp.v3_02.xml

- JID_968067831986 successfully created for ImportSystemConfiguration method


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Importing Server Configuration Profile.


- Query job ID command passed
- Job ID = JID_968067831986
- Name = Import Configuration
- Message = Successfully imported and applied Server Configuration Profile.

- JID_968067831986 completed in: 0:00:13
```

```
$ racadm -r 192.168.0.120 -u user -p password lclog viewconfigresult -j
JID_743455085871
SeqNumber       = 34771
FQDD            = LifecycleController.Embedded.1
Job Name        = Import Configuration
DisplayValue    = Collect System Inventory on Restart
Name            = LCAttributes.1#CollectSystemInventoryOnRestart
OldValue        = Enabled
NewValue        = Disabled
Status          = Success
ErrCode         = 0
$
```

2.7 Importing SCP with Firmware Repository Update

Beginning with iDRAC9 firmware 3.00.00.00 and iDRAC7/8 firmware 2.50.50.50, SCP import files support a new attribute **RepostoryUpdate** which points to a PowerEdge firmware repository created by Dell Repository Manager. The repository file must be stored in the same directory or a sub-directory of the directory holding the SCP import file. RepositoryUpdate can specify a directory, in which case iDRAC will search there for the default repository named "Catalog.xml". Or RepositoryUpdate can explicitly specify the file name of the repository.

When an SCP file containing a **RepositoryUpdate** attribute is imported, the iDRAC will compare the content of the repository file with the currently installed server firmware. As needed, iDRAC will update the server firmware to match the versions stored in the repository; once the firmware update has been performed, any configuration settings in SCP file will be applied to the server. Note that **RepositoryUpdate** is supported via network shares only and is not supported by streamed local file SCP import.

Here is an example SCP file that is stored in an HTTPS share directory Profiles; the repository is stored in Profiles/Repositories/

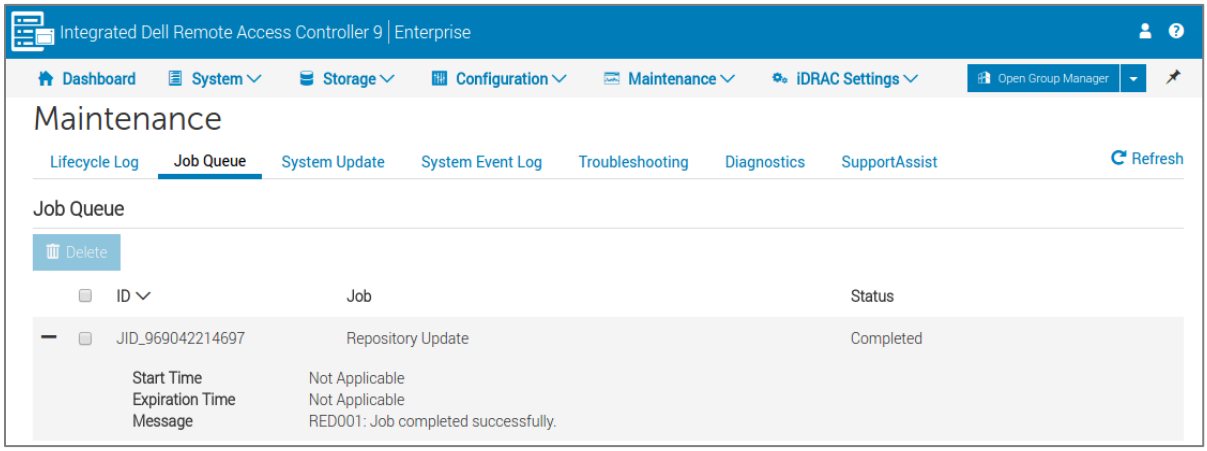
```
[root@localhost Profiles]# cat SCP_with_firmware_repo.xml
<SystemConfiguration>
<Component FQDD="System.Embedded.1">
<Attribute Name="LCD.1#Configuration">Service Tag</Attribute>
<!-- <Attribute Name="LCD.1#UserDefinedString"></Attribute> -->
<Attribute Name="LCD.1#vConsoleIndication">Enabled</Attribute>
<Attribute Name="LCD.1#QualifierWatt">Watts</Attribute>
<Attribute Name="LCD.1#QualifierTemp">C</Attribute>
<Attribute Name="LCD.1#ChassisIdentifyDuration">0</Attribute>
<Attribute Name="LCD.1#HideErrs">unhide</Attribute>
<Attribute Name="LCD.1#ErrorDisplayMode">SEL</Attribute>
<Attribute Name="LCD.1#FrontPanelLocking">Full-Access</Attribute>
<Attribute Name="LCD.1#LicenseMsgEnable">No-License-Msg</Attribute>
<Attribute Name="LCD.1#NMIResetOverride">Disabled</Attribute>
<Attribute Name="ThermalConfig.1#EventGenerationInterval">30</Attribute>
<Attribute Name="ThermalConfig.1#CriticalEventGenerationInterval">30</Attribute>
<Attribute Name="Storage.1#RemainingRatedWriteEnduranceAlertThreshold">10</Attribute>
<Attribute Name="Storage.1#AvailableSpareAlertThreshold">10</Attribute>
<Attribute Name="ThermalSettings.1#ThermalProfile">Default Thermal Profile Settings</Attribute>
<Attribute Name="ThermalSettings.1#AirExhaustTemp">70</Attribute>
<Attribute Name="ThermalSettings.1#FanSpeedOffset">Off</Attribute>
<Attribute Name="ThermalSettings.1#MinimumFanSpeed">255</Attribute>
<Attribute Name="ThermalSettings.1#PCIeSlotLFMSupport">Supported</Attribute>
<Attribute Name="ServerInfo.1#NodeID">ABCDEFG</Attribute>
<Attribute Name="Backplane.1#BackplaneSplitMode">0</Attribute>
<Attribute Name="ServerPwr.1#PowerCapSetting">Disabled</Attribute>
<Attribute Name="ServerPwr.1#PowerCapValue">32767</Attribute>
<Attribute Name="ServerPwr.1#PSRedPolicy">Not Redundant</Attribute>
<Attribute Name="ServerPwr.1#PSPFCEnabled">Disabled</Attribute>
<Attribute Name="ServerPwr.1#PSRapidOn">Enabled</Attribute>
<Attribute Name="ServerPwr.1#RapidOnPrimaryPSU">PSU1</Attribute>
<Attribute Name="PCIeSlotLFM.1#CustomLFM">0</Attribute>
<Attribute Name="PCIeSlotLFM.1#LFMMode">Automatic</Attribute>
<Attribute Name="PCIeSlotLFM.2#CustomLFM">0</Attribute>
<Attribute Name="PCIeSlotLFM.2#LFMMode">Automatic</Attribute>
<Attribute Name="ServerPwrMon.1#PowerConfigReset">None</Attribute>
```

```
<Attribute Name="ThermalHistorical.1#IntervalInSeconds">0</Attribute>
<Attribute Name="PowerHistorical.1#IntervalInSeconds">0</Attribute>
<Attribute Name="RepositoryUpdate">Profiles/Repositories/</Attribute>
</Component>
</SystemConfiguration>
```

If our <Share name> is called “http_root” we can use “find” to display the directories and files to get an idea of the folder structure.

```
[root@localhost http_root]# find .
.
./Profiles
./Profiles/Repositories
./Profiles/Repositories/BIOS_firmware_WN64_1.0.3.EXE
./Profiles/Repositories/Catalog.xml
./Profiles/SCP_with_firmware_repo.xml
```

Running an import of this SCP file will result in a Repository update job as per the below iDRAC9 screenshot:



2.8 Importing SCP from an HTTP share with iDRAC

Importing an SCP file from a HTTP(S) server is supported by iDRAC9. The below script demonstrates this with parameters input for the iDRAC IP and an SCP import filename.

Script: redfish_SCP_import_http.py

```
# Python script using Redfish API to perform iDRAC feature
# Server Configuration Profile (SCP) for import only

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
```

```
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemCon
figuration' % idrac_ip

# For payload dictionary supported parameters, refer to schema
# "https://'iDRAC IP'/redfish/v1/Managers/iDRAC.Embedded.1/"

payload = {"ShutdownType":"Forced","ShareParameters":{"Target":"ALL","IPAddress":"192.168.0.130","
ShareName":"WebServerFolder","ShareType":"HTTP","FileName":file}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n- FAIL, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- %s successfully created for ImportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    final_message_string=str(message_string)
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in final_message_string or "completed with errors" in final_message_string or "Not
 one" in final_message_string or "Unable" in final_message_string:
        print "\n- FAIL, detailed job message is: %s" % data[u"Messages"]
        sys.exit()
    elif "Successfully imported" in final_message_string or "completed with errors" in final_messa
ge_string or "Successfully imported" in final_message_string:
```

```
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif "No changes" in final_message_string:
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue



data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

When we run the script we can verify that the JSON SCP file is successfully imported from the web server.

```
python ./redfish_SCP_import_http.py 192.168.0.120 root calvin
jwr_rf_exp_http_04_all.json

- JID_968132722365 successfully created for ImportSystemConfiguration method


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Importing Server Configuration Profile.


- Query job ID command passed
- Job ID = JID_968132722365
- Name = Import Configuration
- Message = Successfully imported and applied Server Configuration Profile.

- JID_968132722365 completed in: 0:00:12
```

2.9 Importing SCP from a streamed local file

Configuration changes do not have to be located on a remote file share. It is fully possible to stream settings from a local file, or indeed specify specific settings directly in the code, from a local client machine. This feature requires iDRAC7/8 firmware 2.50.50.50 or later and iDRAC9 firmware 3.00.00.00 or later.

As input parameters the following script will take an iDRAC IP address, the IP admin credentials and a local filename of an SCP file to stream to the target.

Script: redfish_SCP_import_local_file.py

```
# Python script using Redfish API to perform iDRAC
# Server Configuration Profile (SCP) import from local file

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip=sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    filename=sys.argv[4]
except:
    print "\n- FAIL: You must pass in script name\iDRAC ip\iDRAC username\iDRAC password"
    sys.exit()

try:
    f=open(filename,"r")
except:
    print "\n-FAIL, \"%s\" file doesn't exist" % filename
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemCon
figuration' % idrac_ip

# Code needed to modify the XML to one string to pass in for POST command
z=f.read()
z=re.sub(" \n ","",z)
z=re.sub(" \n","",z)
xml_string=re.sub("    ","",z)
f.close()

payload = {"ImportBuffer":"","ShareParameters":{"Target":"ALL"}}
payload["ImportBuffer"]=xml_string
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=('root
','calvin'))

#print '\n- Response status code is: %s' % response.status_code

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
```

```
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n- FAIL, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- %s successfully created for ImportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    final_message_string=str(message_string)
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in final_message_string or "completed with errors" in final_message_string or "Not
 one" in final_message_string:
        print "\n- FAIL, detailed job message is: %s" % data[u"Messages"]
        sys.exit()
    elif "Successfully imported" in final_message_string or "completed with errors" in final_messa
ge_string or "Successfully imported" in final_message_string:
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif "No changes" in final_message_string:
        print "- Job ID = "+data[u"Id"]
        print "- Name = "+data[u"Name"]
        try:
            print "- Message = "+message_string[0][u"Message"]
        except:
            print "- Message = %s" % message_string[len(message_string)-1][u"Message"]
        print "\n- %s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue
```

```
data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

```
$ python ./redfish_SCP_import_local_file.py 192.168.0.120 root calvin
jwr_rf_clone.xml

- JID_968285068285 successfully created for ImportSystemConfiguration method

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Importing Server Configuration Profile.

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Waiting for the system to shut down.

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Applying configuration changes.

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Updating component configuration.

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Staged component configuration is complete.

- Query job ID command passed
- Job ID = JID_968285068285
- Name = Import Configuration
- Message = Successfully imported and applied Server Configuration Profile.

- JID_968285068285 completed in: 0:05:01
$
```

2.10 Cloning servers with iDRAC RESTful API

Cloning servers saves considerable time and effort whether you do it for installing or re-deploying servers, or to ensure that nodes in a cluster conform to certain specified settings. Cloning helps in minimizing configuration discrepancies among cluster nodes by protecting configuration from human errors.

A server master image can be created from a known-good configuration and then imported to multiple servers. For full clones, where all settings are imported rather than a subset, the source and destination servers must have matching hardware components and preferably, matching firmware revisions to ensure compatibility.

Use clones to:

- Quickly deploy multiple identical servers
- Ensure uniform settings across a cluster
- Aid troubleshooting by applying settings from a known-good server to a non-working server

2.11 Creating a master image of an already configured server

When exporting the server configuration profile, there are three options available:

1. Default
2. Clone
3. Replace

These are listed under the heading “ExportUse@Redfish.AllowableValues” in Appendix 2 as part of the JSON displayed when accessing the following URL in a browser:

https://<iDRAC IP>/redfish/v1/Managers/iDRAC.Embedded.1

For this example, we use the “Clone” option as it ensures that any existing settings on the target device are overwritten to match the settings in the master image. This process can be destructive as it replaces settings like RAID volumes on the target server. Verify if an exported SCP will replace RAID volumes by checking if the **RAIDaction** value is set to **CreateAuto** as shown below:

```
<Attribute Name="RAIDreconstructRate">30</Attribute>
<Components FQDD="Disk.Virtual.0:RAID.Integrated.1-1">
  <Attribute Name="RAIDaction">CreateAuto</Attribute>
  <Attribute Name="LockStatus">Unlocked</Attribute>
  <Attribute Name="RAIDinitOperation">None</Attribute>
  <!-- <Attribute Name="T10PIStatus">Disabled</Attribute>
  <Attribute Name="DiskCachePolicy">Enabled</Attribute>
  <Attribute Name="RAIDdefaultWritePolicy">WriteBack</Attribute>
  <Attribute Name="RAIDdefaultReadPolicy">AdaptiveReadAhead</Attribute>
```

To enable cloning, update the configuration export script to include the **ExportUse** option with **Clone** specified. This option is added to the JSON payload part of the export script in the same order as the options

were displayed when verifying them using the browser session. Using the wrong order may cause the export to fail or the option to be ignored.

Script: redfish_SCP_export_clone.py

```
# Python script using Redfish API to perform iDRAC feature
# Server Configuration Profile (SCP) for export only with clone enabled

import requests, json, sys, re, time

from datetime import datetime

try:
    idrac_ip = sys.argv[1]
    idrac_username = sys.argv[2]
    idrac_password = sys.argv[3]
    file = sys.argv[4]
except:
    print "\n- FAIL, you must pass in script name along with iDRAC IP/iDRAC username/iDRAC paasswo
rd/file name"
    sys.exit()

url = 'https://%s/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemCon
figuration' % idrac_ip

# For payload dictionary supported parameters, refer to schema
# "https://'iDRAC IP'/redfish/v1/Managers/iDRAC.Embedded.1/"

payload = {"ExportFormat":"XML","ExportUse":"Clone","ShareParameters":{"Target":"All","IPAddress":
"192.168.0.130","ShareName":"cifs_share","ShareType":"CIFS","FileName":file,"UserName":"cifs_user"
,"Password":"cifs_password"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, verify=False, auth=(idrac
_username,idrac_password))

d=str(response.__dict__)

try:
    z=re.search("JID_.+?,",d).group()
except:
    print "\n- FAIL: detailed error message: {0}".format(response.__dict__['_content'])
    sys.exit()

job_id=re.sub("[,']","",z)
if response.status_code != 202:
    print "\n##### Command Failed, status code not 202\n, code is: %s" % response.status_code
    sys.exit()
else:
    print "\n- %s successfully created for ExportSystemConfiguration method\n" % (job_id)

response_output=response.__dict__
job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
start_time=datetime.now()

while True:
```

```
    req = requests.get('https://%s/redfish/v1/TaskService/Tasks/%s' % (idrac_ip, job_id), auth=(id
rac_username, idrac_password), verify=False)
    statusCode = req.status_code
    data = req.json()
    message_string=data[u"Messages"]
    current_time=(datetime.now()-start_time)
    if statusCode == 202 or statusCode == 200:
        print "\n- Query job ID command passed"
        time.sleep(10)
    else:
        print "Query job ID command failed, error code is: %s" % statusCode
        sys.exit()
    if "failed" in data[u"Messages"] or "completed with errors" in data[u"Messages"]:
        print "Job failed, current message is: %s" % data[u"Messages"]
        sys.exit()
    elif data[u"TaskState"] == "Completed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    elif data[u"TaskState"] == "Completed with Errors" or data[u"TaskState"] == "Failed":
        print "\nJob ID = "+data[u"Id"]
        print "Name = "+data[u"Name"]
        try:
            print "Message = "+message_string[0][u"Message"]
        except:
            print data[u"Messages"][0][u"Message"]
        print "JobStatus = "+data[u"TaskState"]
        print "\n%s completed in: %s" % (job_id, str(current_time)[0:7])
        sys.exit()
    else:
        print "- Job not marked completed, current status is: %s" % data[u"TaskState"]
        print "- Message: %s\n" % message_string[0][u"Message"]
        time.sleep(1)
        continue

data = req.json()
print "Job ID = "+data[u"Id"]
print "Name = "+data[u"Name"]
print "Message = "+data[u"Messages"]
print "JobStatus = "+data[u"TaskState"]
```

2.12 Applying a master configuration image to a target server

The configuration files resulting from exporting settings using the **Clone** options can be imported just as easily and in the same fashion as any other configuration file using the import scripts previously shown.

Utilizing the script which was modified to include the **ExportUse** option the master image configuration is first exported.

```
$ python ./redfish_SCP_export_clone.py 192.168.0.120 root calvin
jwr_rf_clone.xml


- JID_968276760233 successfully created for ExportSystemConfiguration method


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Exporting Server Configuration Profile.


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Exporting Server Configuration Profile.


- Query job ID command passed

Job ID = JID_968276760233
Name = Export: Server Configuration Profile
Message = Successfully exported Server Configuration Profile
JobStatus = Completed

JID_968276760233 completed in: 0:00:24
$
```

2.12.1 Modifying the iDRAC IP address to match the clone target

Since a full clone is desired, all settings are kept as-is in the export file with the exception of the iDRAC address. Unless the iDRAC IP address is updated, the target iDRAC IP address will become identical to the master and will not function on the network. We will now modify the iDRAC IP address from 192.168.0.120 to 192.168.0.121 to avoid this address conflict:

```
<Attribute Name="NICStatic.1#DNSDomaininFromDHCP">Disabled</Attribute>
<Attribute Name="IPv4Static.1#Address">192.168.0.121</Attribute>
<Attribute Name="IPv4Static.1#Netmask">255.255.255.0</Attribute>
<Attribute Name="IPv4Static.1#Gateway">192.168.0.1</Attribute>
<Attribute Name="IPv4Static.1#DNS1">8.8.8.8</Attribute>
```

2.12.2 Importing the cloned SCP to the target server

The SCP import script require no modification for the cloning process and is used below. Since all settings are applied due to the clone setting it takes about 5 minutes to complete. The output has been shortened for brevity.

```
$ python ./redfish_SCP_import_local_file.py 192.168.0.121 root calvin
jwr_rf_clone.xml

- JID_968285068285 successfully created for ImportSystemConfiguration method


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Importing Server Configuration Profile.


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Waiting for the system to shut down.

- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Applying configuration changes.


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Updating component configuration.


- Query job ID command passed
- Job not marked completed, current status is: Running
- Message: Staged component configuration is complete.


- Query job ID command passed
- Job ID = JID_968285068285
- Name = Import Configuration
- Message = Successfully imported and applied Server Configuration Profile.

- JID_968285068285 completed in: 0:05:01
$
```

2.13 Using partial SCP imports

Backing up, restoring and cloning entire server configurations is useful but in some cases only small changes are required. For example, you might need to clone only a few BIOS settings. In such cases, XML imports are useful because it is possible to modify multiple settings in one go. This is in contrast to using RACADM single object set commands where each setting must be modified individually.

It can be valuable to have a set of preconfigured XML snippets containing settings for just a subset of a server's configuration available when needed. Like a master clone image, they are known to be good and can be imported without having to modify more settings than required.

2.14 Creating SCP files for partial imports

Exported server settings within an SCP are saved in XML format that can be easily edited with a text editor. XML uses open and closing tags and hierarchy to group content.

In the following example the VNC server settings have been singled out. In this example, the start and end tags for **SystemConfiguration** and **Component** are retained while all other information is removed. The same applies to the **Attributes** for the VNC server. When exported, the **iDRAC.Embedded.1** component section contains many more attributes, but for this example only the VNC settings have been kept.

```
<SystemConfiguration Model="PowerEdge FC630" ServiceTag="000000" TimeStamp="Mon
Sep 26 18:57:00 2016">

<Component FQDD="iDRAC.Embedded.1">

 <Attribute Name="VNCServer.1#Enable">Enabled</Attribute>
 <Attribute Name="VNCServer.1#Password">password</Attribute>
 <Attribute Name="VNCServer.1#Port">5901</Attribute>
 <Attribute Name="VNCServer.1#LowerEncryptionBitLength">Disabled</Attribute>
 <Attribute Name="VNCServer.1#Timeout">300</Attribute>
 <Attribute Name="VNCServer.1#SSLEncryptionBitLength">Disabled</Attribute>

</Component>

</SystemConfiguration>
```

This SCP can be saved as a known-good configuration file for enabling VNC. The import process is the same as shown in previous scripts throughout the document. Generally, the settings are not unique to their server types. The PowerEdge FC630 tag at the start can be ignored and the XML can be used for any server with VNC capabilities and of similar iDRAC with Lifecycle Controller firmware version.

2.15 Keeping order among server configuration files

Using SCP files to manage a fleet of servers can save time and effort but the number and variation of different configuration files can multiply. One way to keep SCP files separate and in order is to use a versioning system, such as Github or Subversion. Detailing the use of such tools is out of scope for this white paper but a recommended best practice.

3 Tips, tricks, and suggestions

1. Depending on the execution environment, Python scripts may generate warnings that do not affect execution. The following example shows warning messages that may occur while running a Python script although the script returns the correct status code and job ID:

```
$ rest_SCP_export_script.py
C:\Python26\lib\site-packages\requests-2.10.0-
py2.6.egg\requests\packages\urllib3\util\ssl_.py:318: SNIMissingWarning: An
HTTPS request has been made, but the SNI (Subject Name Indication) extension to
TLS is not available on this platform. This may cause the server to present an
incorrect TLS certificate, which can cause validation failures. You can upgrade
to a newer version of Python to solve this. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#snimissingwarning.
SNIMissingWarning
C:\Python26\lib\site-packages\requests-2.10.0-
py2.6.egg\requests\packages\urllib3\util\ssl_.py:122: InsecurePlatformWarning: A
true SSLContext object is not available. This prevents urllib3 from configuring
SSL appropriately and may cause certain SSL connections to fail. You can upgrade
to a newer version of Python to solve this. For more information, see
https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarning.
InsecurePlatformWarning
C:\Python26\lib\site-packages\requests-2.10.0-
py2.6.egg\requests\packages\urllib3\connectionpool.py:821:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding
certificate verification is strongly advised. See:
https://urllib3.readthedocs.org/en/latest/security.html InsecureRequestWarning)
202
JID_744059638886
$
```

2. When the RESTful API calls return a failure or do not return a Job ID, extra details are available from a built-in message dictionary located with the “@Message.ExtendedInfo” key. The following example illustrates the Python script and shows execution of the script to return an error message string. To generate an error, we pass in an unsupported ShareType of “FTP”.

   Example Python script which looks for a specific status code and then parses the data, and prints the error message in pretty format.

```
import requests, json, re, sys
url = 'https://192.168.0.120/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem
/EID_674_Manager.ExportSystemConfiguration'
payload = {"ExportFormat":"XML","ShareParameters":{"Target":"ALL","IPAddress"
:"192.168.0.130","ShareName":"cifs_share","ShareType":"FTP","FileName":"R730_
SCP.xml","UserName":"user","Password":"password"}}
headers = {'content-type': 'application/json'}
response = requests.post(url, data=json.dumps(payload), headers=headers, veri
fy=False, auth=('username','password'))
print '\n- Response status code is: %s' % response.status_code
```

```
response_output=response.__dict__
if response.status_code == 400:
  print "- ExportSystemConfiguration method failed to return job ID"
  get_error_string=response_output["_content"]

message=re.search("@Message.ExtendedInfo.+?,",get_error_string).group().strip
("@Message.ExtendedInfo\":[{")
  print "- Error message is: \"%s\"" % message.strip(".\",")
  sys.exit()
else:
  pass

job_id=response_output["headers"]["Location"]
job_id=re.search("JID_.+",job_id).group()
print "- Job ID is: %s" % job_id
```

Example of running the script:

```
$ redfish_SCP_script.py
- ExportSystemConfiguration method failed to return job ID
- Response status code is: 400
- Error message is: "The value FTP for the property ShareType is not in the
list of acceptable values"
$
```

3. Detailed information about completed SCP operations are recorded in the Lifecycle Controller (LC) Log. The LC Log can be viewed using the iDRAC GUI, appropriate WS-Man API calls, or RACADM CLI commands.

4. When a job fails, the Redfish Task Service provides an error key shown as MessageId below. Details about the MessageId can be found in the Event and Error Message Reference Guide for 13th Generation Dell PowerEdge Servers . The following example shows an SCP export Job ID failure purposely caused by passing an invalid network share IP address. The error message ID is highlighted

```
$ rest_get_job_id.py JID_744092584176

- Command passed, code 200 returned

{
  u '@odata.type': u '#Task.v1_0_2.Task',
  u 'Description': u 'Server Configuration and other Tasks running on iDRAC are listed
here',
  u 'TaskState': u 'Completed',
  u 'Messages': [{
    u 'Message': u 'Unable to copy the system configuration XML file to the network
share.',
    u 'MessageId': u 'SYS045',
    u 'MessageArgs': [],
    u 'MessageArgs@odata.count': 0
```

```
  }, {
    u 'Message': u 'For more information run command - racadm lclog
viewconfigresult -j JID_xxxxxxxxxxx'
  }],
  u '@odata.id': u '/redfish/v1/TaskService/Tasks/JID_744092584176',
  u '@odata.context': u '/redfish/v1/$metadata#Task.Task',
  u 'TaskStatus': u 'Critical',
  u 'Messages@odata.count': 2,
  u 'StartTime': u '2016-09-20T17:07:38-05:00',
  u 'EndTime': u '2016-09-20T17:21:05-05:00',
  u 'Id': u 'JID_744092584176',
  u 'Name': u 'Export Configuration'
}

$
```

4 Summary

Using the iDRAC RESTful API, administrators can obtain the configuration details of 12th, 13th and 14th generation Dell PowerEdge servers, preview the application of a configuration file to a server, and apply configuration files to establish BIOS, iDRAC, PERC RAID controller, NIC, and HBA settings.

Dell is a committed leader in the development and implementation of open, industry standards. Supporting Redfish within the iDRAC with Lifecycle Controller further enhances the manageability of PowerEdge servers and provides another powerful tool to help IT administrators reduce complexity and help save time and money.

5 Additional Information

- For more information on iDRAC with Lifecycle Controller, visit the Dell TechCenter.
- Overview of the iDRAC Redfish API
  http://en.community.dell.com/techcenter/extras/m/white_papers/20442330
- DMTF white papers, Redfish Schemas, specifications, webinars, and work-in-progress documents
  https://www.dmtf.org/standards/redfish
- The Redfish standard specification is available from the DMTF website
- http://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.0.1.pdf

Details on using Server Configuration Profiles can be found in these white papers:

- Understanding the structure of SCP XML file
  http://en.community.dell.com/techcenter/extras/m/white_papers/20269601.aspx
- Cloning servers configuration with SCPs
  http://en.community.dell.com/techcenter/extras/m/white_papers/20439335.aspx
- Zero Touch Auto Configuration with SCPs
  http://en.community.dell.com/techcenter/extras/m/white_papers/20441340
- PowerShell Cmdlets for the iDRAC WS-Man API including SCP exports and import operations
  http://en.community.dell.com/techcenter/systems-management/w/wiki/7727.powershell-cmdlets-for-poweredge-servers

A.1 Verifying iDRAC RESTful API with Redfish service is enabled

Before running any iDRAC RESTful API with Redfish workflows, verify that Redfish support is enabled - default setting is **Enabled**. Redfish service enablement can be verified with WS-MAN, the RACADM CLI, or the iDRAC GUI.

Verifying whether Redfish is enabled using WS-MAN

```
C:\>winrm g http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/DCIM_iDRACCard
Enumeration?InstanceID=iDRAC.Embedded.1#Redfish.1#Enable -u:username -p:password
-r:https:/
/192.168.0.120/wsman -SkipCNcheck -SkipCAcheck -encoding:utf-8 -a:basic

DCIM_iDRACCardEnumeration
  AttributeDisplayName = Enable
  AttributeName = Enable
  CurrentValue = Enabled
  DefaultValue = Enabled
  Dependency = null
  DisplayOrder = 2250
  FQDD = iDRAC.Embedded.1
  GroupDisplayName = Redfish
  GroupID = Redfish.1
  InstanceID = iDRAC.Embedded.1#Redfish.1#Enable
  IsReadOnly = false
  PendingValue = null
  PossibleValues = Disabled, Enabled
C:\>
```

Verifying whether Redfish is enabled using RACADM

```
C:\>racadm -r 192.168.0.120 -u username -p password get idrac.redfish.enable
[Key=idrac.Embedded.1#Redfish.1]
Enable=Enabled
C:\>
```

Verifying whether Redfish is enabled using the iDRAC GUI

iDRAC7/8: Start the iDRAC GUI with a web browser and traverse to the Network Service page - Server -> iDRAC Settings -> Network -> Services -> Redfish. Verify that Enabled is selected.

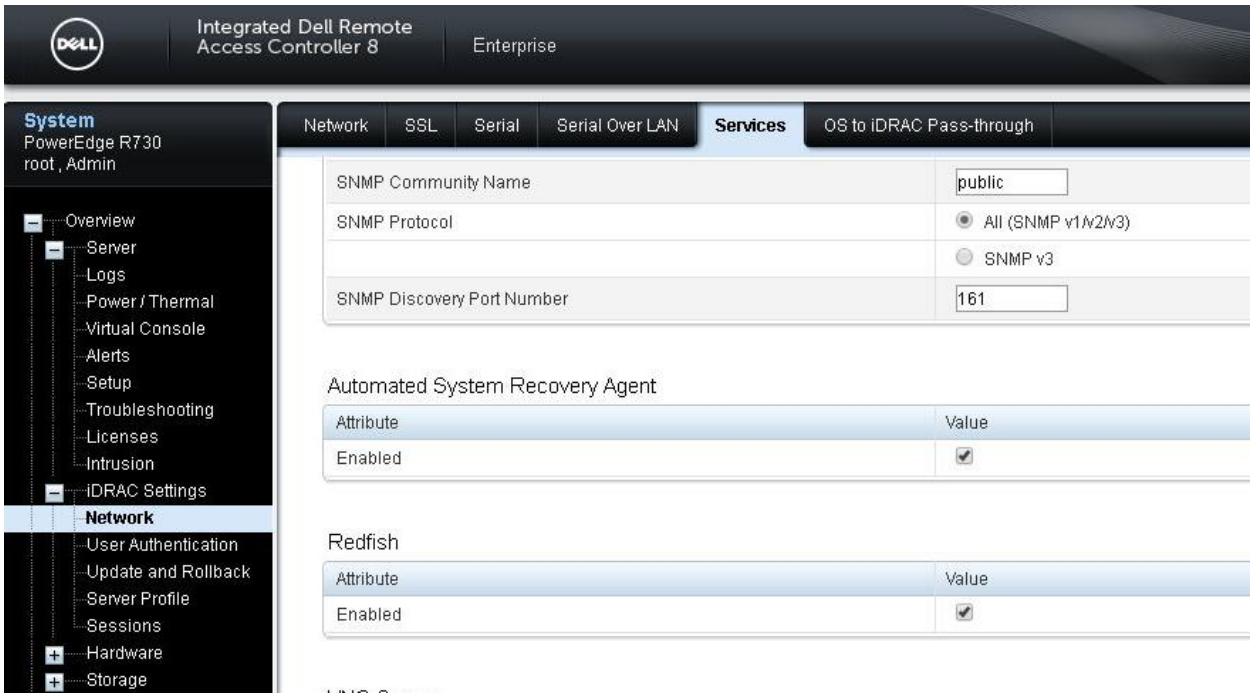


Figure 2 Enabling iDRAC RESTful API with Redfish service via iDRAC7/8 GUI

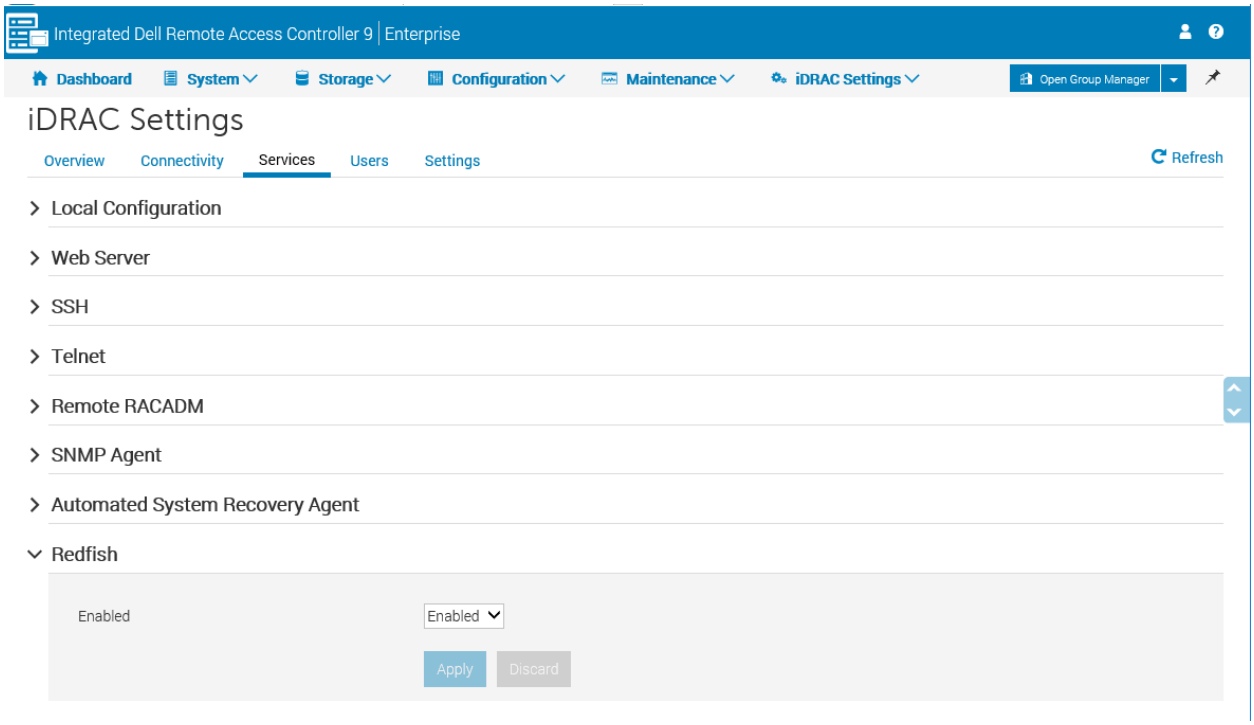


Figure 3 Enabling iDRAC RESTful API with Redfish service via iDRAC9 GUI

A.2 iDRAC RESTful API – SCP Export, Preview, and Import APIs

Using any iDRAC-supported web browser, the various parameters used for RESTful SCP export, preview and import operations can be viewed. Start an iDRAC-supported web browser and enter this URL:

https://<iDRACIP>/redfish/v1/Managers/iDRAC.Embedded.1

After entering the URL, input the iDRAC administrator user name and password when prompted. After you enter the credentials, JSON output similar to the one shown below is displayed.

In the JSON output, under the “OEM” section are the supported methods for the SCP feature including **ExportSystemConfiguration**, **ImportSystemConfiguration** and **ImportSystemConfigurationPreview**. Each method section details the supported parameters; specification of required or optional parameters for each method can be found in the iDRAC Redfish API Guide available at **dell.com/idracmanuals**.

```
{
    "@odata.context": "/redfish/v1/$metadata#Manager.Manager",
    "@odata.id": "/redfish/v1/Managers/iDRAC.Embedded.1",
    "@odata.type": "#Manager.v1_0_2.Manager",
    "Actions": {
        "#Manager.Reset": {
            "ResetType@Redfish.AllowableValues": [
                "GracefulRestart"
            ],
            "target":
"/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Manager.Reset"
        },
        "Oem": {
            "DellManager.v1_0_0#DellManager.ResetToDefaults": {
                "ResetType@Redfish.AllowableValues": [
                    "All",
                    "ResetAllWithRootDefaults",
                    "Default"
                ],
                "target":
"/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/DellManager.ResetToDefaults"
            },
            "OemManager.v1_0_0#OemManager.ExportSystemConfiguration": {
                "ExportFormat@Redfish.AllowableValues": [
                    "XML",
                    "JSON"
                ],
                "ExportUse@Redfish.AllowableValues": [
                    "Default",
                    "Clone",
                    "Replace"
                ],
```

```
"IncludeInExport@Redfish.AllowableValues": [
    "Default",
    "IncludeReadOnly",
    "IncludePasswordHashValues",
    "IncludeReadOnly,IncludePasswordHashValues"
],
"ShareParameters": {
    "IgnoreCertificateWarning@Redfish.AllowableValues": [
        "Disabled",
        "Enabled"
    ],
    "ProxySupport@Redfish.AllowableValues": [
        "Disabled",
        "EnabledProxyDefault",
        "Enabled"
    ],
    "ProxyType@Redfish.AllowableValues": [
        "HTTP",
        "SOCKS4"
    ],
    "ShareParameters@Redfish.AllowableValues": [
        "IPAddress",
        "ShareName",
        "FileName",
        "UserName",
        "Password",
        "Workgroup",
        "ProxyServer",
        "ProxyUserName",
        "ProxyPassword",
        "ProxyPort"
    ],
    "ShareType@Redfish.AllowableValues": [
        "NFS",
        "CIFS",
        "HTTP",
        "HTTPS"
    ],
    "Target@Redfish.AllowableValues": [
        "ALL",
        "IDRAC",
        "BIOS",
        "NIC",
        "RAID"
    ]
},
```

```
                "target":
"/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ExportSystemC
onfiguration"
            },
            "OemManager.v1_0_0#OemManager.ImportSystemConfiguration": {
                "HostPowerState@Redfish.AllowableValues": [
                    "On",
                    "Off"
                ],
                "ImportSystemConfiguration@Redfish.AllowableValues": [
                    "TimeToWait",
                    "ImportBuffer"
                ],
                "ShareParameters": {
                    "IgnoreCertificateWarning@Redfish.AllowableValues": [
                        "Disabled",
                        "Enabled"
                    ],
                    "ProxySupport@Redfish.AllowableValues": [
                        "Disabled",
                        "EnabledProxyDefault",
                        "Enabled"
                    ],
                    "ProxyType@Redfish.AllowableValues": [
                        "HTTP",
                        "SOCKS4"
                    ],
                    "ShareParameters@Redfish.AllowableValues": [
                        "IPAddress",
                        "ShareName",
                        "FileName",
                        "UserName",
                        "Password",
                        "Workgroup",
                        "ProxyServer",
                        "ProxyUserName",
                        "ProxyPassword",
                        "ProxyPort"
                    ],
                    "ShareType@Redfish.AllowableValues": [
                        "NFS",
                        "CIFS",
                        "HTTP",
                        "HTTPS"
                    ],
                    "Target@Redfish.AllowableValues": [
                        "ALL",
```

```
                        "IDRAC",
                        "BIOS",
                        "NIC",
                        "RAID"
                    ]
                },
                "ShutdownType@Redfish.AllowableValues": [
                    "Graceful",
                    "Forced",
                    "NoReboot"
                ],
                "target":
"/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemC
onfiguration"
            },
            "OemManager.v1_0_0#OemManager.ImportSystemConfigurationPreview": {
                "ImportSystemConfigurationPreview@Redfish.AllowableValues": [
                    "ImportBuffer"
                ],
                "ShareParameters": {
                    "IgnoreCertificateWarning@Redfish.AllowableValues": [
                        "Disabled",
                        "Enabled"
                    ],
                    "ProxySupport@Redfish.AllowableValues": [
                        "Disabled",
                        "EnabledProxyDefault",
                        "Enabled"
                    ],
                    "ProxyType@Redfish.AllowableValues": [
                        "HTTP",
                        "SOCKS4"
                    ],
                    "ShareParameters@Redfish.AllowableValues": [
                        "IPAddress",
                        "ShareName",
                        "FileName",
                        "UserName",
                        "Password",
                        "Workgroup",
                        "ProxyServer",
                        "ProxyUserName",
                        "ProxyPassword",
                        "ProxyPort"
                    ],
                    "ShareType@Redfish.AllowableValues": [
                        "NFS",
```

```
                        "CIFS",
                        "HTTP",
                        "HTTPS"
                    ],
                    "Target@Redfish.AllowableValues": [
                        "ALL"
                    ]
                },
                "target":
"/redfish/v1/Managers/iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.ImportSystemConfigurat
ionPreview"
            }
        }
    }
```