Redfish

Reference Guide

Revision 1.0

The information in this USER'S MANUAL has been carefully reviewed and is believed to be accurate. The vendor assumes no responsibility for any inaccuracies that may be contained in this document, makes no commitment to update or to keep current the information in this manual, or to notify any person organization of the updates. Please Note: For the most up-to-date version of this manual, please see our web site at www.supermicro.com.

Super Micro Computer, Inc. ("Supermicro") reserves the right to make changes to the product described in this manual at any time and without notice. This product, including software, if any, and documentation may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any medium or machine without prior written consent.

IN NO EVENT WILL SUPERMICRO BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, SPECULATIVE OR CONSEQUENTIAL DAMAGES ARISING FROM THE USE OR INABILITY TO USETHIS PRODUCT OR DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCHDAMAGES. IN PARTICULAR, SUPERMICRO SHALL NOT HAVE LIABILITY FOR ANY HARDWARE,SOFTWARE, OR DATA STORED OR USED WITH THE PRODUCT, INCLUDING THE COSTS OFREPAIRING, REPLACING, INTEGRATING, INSTALLING OR RECOVERING SUCH HARDWARE,SOFTWARE, OR DATA.

Any disputes arising between manufacturer and customer shall be governed by the laws of Santa Clara County in the State of California, USA. The State of California, County of Santa Clara shall be the exclusive venue for the resolution of any such disputes. Super Micro's total liability for all claims will not exceed the price paid for the hardware product.

Information in this document is subject to change without notice. Other products and companies referred to herein are trademarks or registered trademarks of their respective companies or mark holders.

Printed in the United States of America

Manual Revision 1.0

Release Date: August 14, 2015

1 Introduction

The Redfish Scalable Platforms Management API ("Redfish") is a new interface that uses RESTful interface semantics to access data defined in a model format to perform out-of-band systems management. It is suitable for a wide range of servers, from stand-alones to rack mount and blade environments, but scales equally well for large scale cloud environments.

Redfish is a management standard which uses data model representation inside of a hypermedia RESTful interface. it is based on REST, that's how Redfish is easier to use and implement than many other solutions. Since it's model oriented, it is capable of expressing the relationships between components in modern systems as well as the semantics of the services and components within them. It is also easily extensible. By using a hypermedia approach to REST, Redfish can express a large variety of systems from multiple vendors. Utilizing JSON (JavaScript Object Notation) data format which is in plain text, allows many types of parameters to be available such that it enables scalability, human readability, and flexibility for most programming environments by easily interpreting payload.

The model is exposed in terms of an interoperable OData Schema with the payload of the messages being expressed in JSON following OData JSON conventions. The schema (available in both XML and JSON formats) includes annotations to facilitate the automatic translation of the schema to JSON Schema. The ability to externally host the schema definition of the resources in a machine-readable format allows the meta data to be associated with the data without encumbering Redfish services with the meta data, thus enabling more advanced client scenarios as found in many data center and cloud environments.

This document will provide you with an overview of Restful API services and describe how to receive Redfish API responses directly from a Supermicro BMC (Baseboard Management Controller).

2 HTTP Request Methods

The following HTTP methods are used to implement different actions, as described below.

- Read Requests (GET):

The GET method is used to request a representation of a specified resource. The representation can be either a single resource or a collection.

- Update (PATCH):

The PATCH method is used to apply partial modifications to a resource.

- Replace (PUT):

The PUT method is used to completely replace a resource. Any properties omitted from the body of the request are reset to their default value.

- Create (POST):

The POST method is used to create a new resource. This request is submitted to the resource collection in which the new resource is meant to belong.

- Actions (POST):

The POST method may also be used to initiate operations on the object (Actions). The POST operation may not be idempotent.

- Delete (DELETE):

The DELETE method is used to remove a resource.

2.1 Reponses

Four types of responses are supported, as defined below.

- Metadata Responses:

These describe the resources and types exposed by the service to generic clients.

- Resource Responses:

JSON representation of an individual resource.

- Resource Collection Responses:

JSON representation of a collections of resources.

- Error Responses:

Top-level JSON response providing additional information in the case of an HTTP error.

2.2 HTTP Status Code Description

Status Code	Description
200	OK
201	Created
202	Accepted
204	No Content
301	Moved permanently
302	Found
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
415	Unsupported Media Type
500	Internal Server Error

501	Not Implemented
503	Service Unavailable

2.3 List of Available APIs

/redfish/v1/
/redfish/v1/Chassis
/redfish/v1/Chassis/1
/redfish/v1/Chassis/1/Thermal
/redfish/v1/Chassis/1/Power
/redfish/v1/Managers
/redfish/v1/Managers/1
/redfish/v1/Managers/1/NetworkProtocol
/redfish/v1/Managers/1/EthernetInterfaces
/redfish/v1/Managers/1/EthernetInterfaces/1
/redfish/v1/Managers/1/SerialInterfaces
/redfish/v1/Managers/1/SerialInterfaces/1
/redfish/v1/Managers/1/LogServices/
/redfish/v1/Managers/1/LogServices/Log1/
/redfish/v1/Managers/1/LogServices/Log1/Entries
/redfish/v1/Managers/1/LogServices/Log1/Entries/[num]
/redfish/v1/Managers/1/LogServices/Log1/Actions/LogService.Reset – (POST)
/redfish/v1/Managers/1/Actions/Manager.Reset – (POST)
/redfish/v1/SessionService/
/redfish/v1/SessionService/Sessions/
/redfish/v1/SessionService/Sessions/[num]
/redfish/v1/Systems
/redfish/v1/Systems/1
/redfish/v1/Systems/1/Processors/[num]
/redfish/v1/Systems/1/SimpleStorage (Support NVME and LSI. LSI is restricted to LSI 3108 series)
/redfish/v1/Systems/1/EthernetInterfaces/[num] (MUST install TAS and run it)
/redfish/v1/Systems/1/Actions/ComputerSystem.Reset (POST)
/redfish/v1/AccountService
/redfish/v1/AccountService/Accounts/
/redfish/v1/AccountService/Accounts/[num]
/redfish/v1/AccountService/Roles
/redfish/v1/AccountService/Roles/Admin
/redfish/v1/AccountService/Roles/Operator
/redfish/v1/AccountService/Roles/ReadOnlyUser
/redfish/v1/EventService

/redfish/v1/EventService/Subscriptions
/redfish/v1/EventService/Actions/EventService.SendTestEvent (POST)

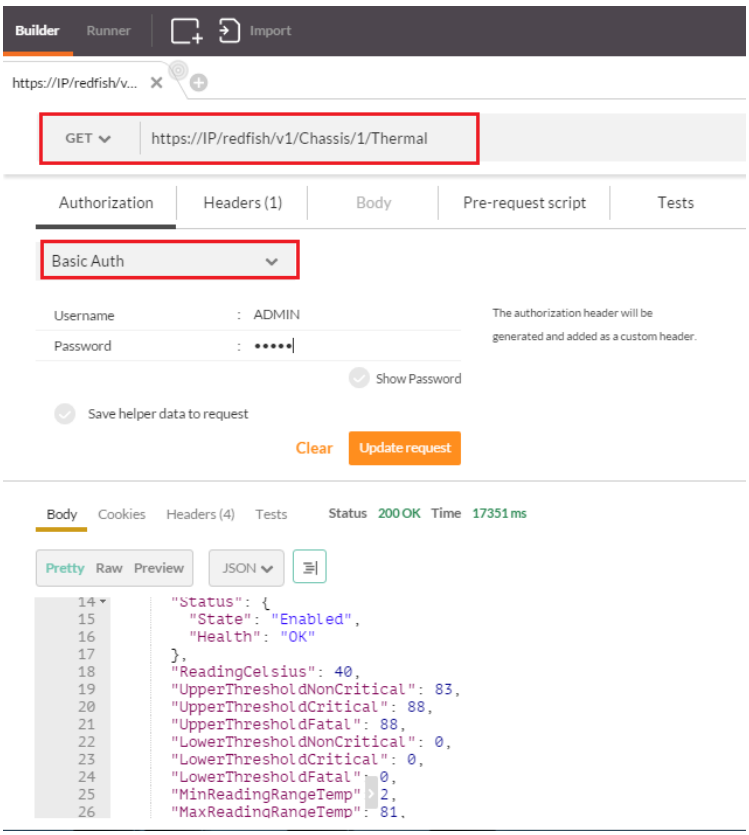
3 Using RESTful APIs

Begin by installing PostMan or Advanced REST client plug-ins in your browser (either Chrome or Firefox).

3.1 Authentication

Redfish supports both "Basic Authentication" and "Redfish Session Login Authentication" (as described below under Session Management). Service does not require a client to create a session when Basic Authentication is used.

3.1.1 Basic Authentication: HTTP BASIC authentication uses compliant TLS connections to transport the data between any third party authentication service and clients.
Example: The following example uses Postman (a web API application) to display a chassis response using Basic Authentication.



Note: Always check the status code once you get the response from the Redfish URL. You can refer to the status code table mentioned above. (All URLs/commands are case sensitive.)

3.1.2 Session Management:

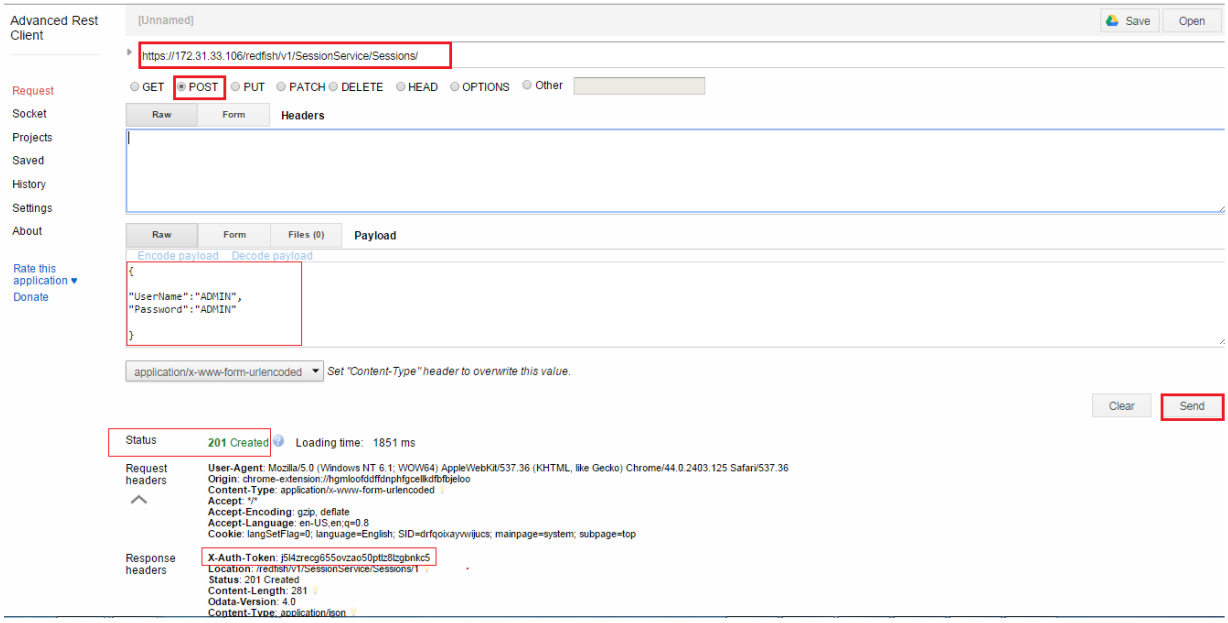
Redfish Service uses session management to implement authentication. This includes orphaned session timeouts and a number of simultaneous open sessions.

Step1. User can Post the following username/password information to the Post payload field, which will create a new session.

```
{
"UserName": "<username>",
"Password": "<password>"
}
```

Example of applying for Authentication using a Chrome-based app (Advanced Rest Client):

1. Enter this URL into your browser: https://BMC ip/redfish/v1/SessionService/Sessions
2. Apply the POST method with username/password info inside the payload and send it.
3. The user will receive 201 messages created with X-AUTH token code.



- Users can create maximum of 16 sessions.
- **Session Lifetime**: For Redfish sessions, as long as a client sends requests for the session within the session timeout period, the session will remain open and the session authentication token will remain valid. If the sessions times-out, the session will be automatically terminated. The default session timeout is 30 minutes. If a user is not active for 30 minutes, the token will be rendered invalid.

  Users can always modify the "SessionTimeout" by using IPMI web or a patching operation from Redfish.

  Example: [PATCH] https://IP/redfish/v1/SessionService

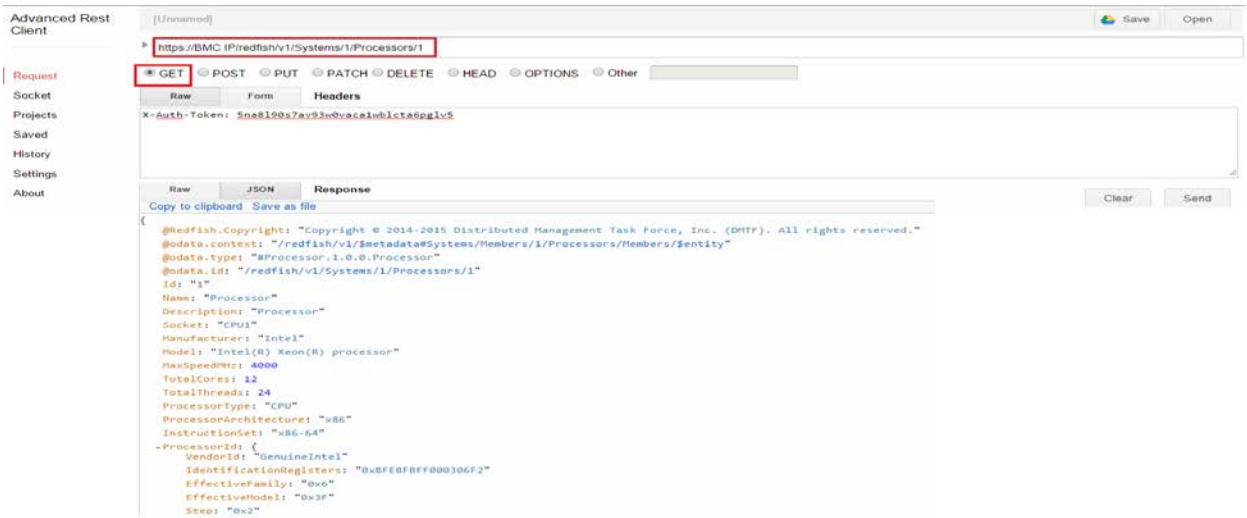
Payload: {"SessionTimeout": 20}->Send->Status Code: 200 OK

- **Session Termination or Logout**: A Redfish session is terminated when the client logs-out. This is accomplished by performing a DELETE to the session resource identified by the link returned in the location header either when the session was created or if the SessionId is returned in the response data. The ability to DELETE a session by specifying the session resource ID allows an administrator with sufficient privilege to terminate other users sessions from a different session.
  Example: [DELETE] https://IP/redfish/v1/SessionService/Sessions/2(num)->Send->StatusCode:200 OK

Log in	Log Out
Operation : POST	Operation: DELETE
URI: redfish/v1/SessionService/Sessions/	URI: redfish/v1/SessionService/Sessions/(num)
Request headers: Content-Type: application/json	Request headers: Content-Type: application/json
Request body: {"UserName":"UserName" , "Password":"Password"}	Requestbody: NONE
Response: 201 created	Response: 200 OK
X-Auth Token header displays Location and session ID ex: Location: /redfish/v1/SessionService/Sessions/5	

Step2. The response will include an X-Auth-token header with a session token and a location header. Copy "X-Auth-Token: <Value>" and parse it to the HTTP header in order to GET API response.

- Parse X-Auth token value to get API response:



3.2 Examples

Users can integrate current APIs into their software and applications in order to receive all services provided by Redfish APIs.

3.2.1 Redfish API Response for Chassis/Thermal


```
{
@Redfish.Copyright: "Copyright © 2014-2015 Distributed Management Task
Force, Inc. (DMTF). All rights reserved."
@odata.context: "/redfish/v1/$metadata#Chassis/Members/1/Thermal/$entity"
@odata.type: "#Thermal.1.0.0.Thermal"
@odata.id: "/redfish/v1/Chassis/1/Thermal"
Id: "Thermal"
Name: "Thermal"
Temperatures: [30]
Fans: [4]
0:
{
@odata.id: "/redfish/v1/Chassis/1/Thermal#/Fans/0"
MemberID: "0"
Status:
{
State: "Enabled"
Health: "OK"
}
UpperThresholdNonCritical: 25300
UpperThresholdCritical: 25400
UpperThresholdFatal: 25500
LowerThresholdNonCritical: 700
LowerThresholdCritical: 500
LowerThresholdFatal: 300
PhysicalContext: "Backplane"
RelatedItem: [2]
0:
{@odata.id: "/redfish/v1/Systems/1"}
1:
{@odata.id: "/redfish/v1/Chassis/1"}
FanName: "FAN1"
ReadingRPM: 7500
MinReadingRange: 702
MaxReadingRange: 25298
```

3.2.2 Redfish API Response for Systems/1/Processor/1


```
{
@Redfish.Copyright: "Copyright © 2014-2015 Distributed Management
Task Force, Inc. (DMTF). All rights reserved."
@odata.context: "/redfish/v1/$metadata#Systems/Members/1/Processors/
Members/$entity"
@odata.type: "#Processor.1.0.0.Processor"
@odata.id: "/redfish/v1/Systems/1/Processors/1"
Id: "1"
Name: "Processor"
Description: "Processor"
Socket: "CPU1"
Manufacturer: "Intel"
Model: "Intel(R) Xeon(R) processor"
MaxSpeedMHz: 4000
TotalCores: 12
TotalThreads: 24
ProcessorType: "CPU"
ProcessorArchitecture: "x86"
InstructionSet: "x86-64"
ProcessorId:
{
VendorId: "GenuineIntel"
IdentificationRegisters: "0xBFEBFBFF000306F2"
EffectiveFamily: "0x6"
EffectiveModel: "0x3F"
Step: "0x2"
}
-
Status:
{
State: "Enabled"
Health: "OK"
```

3.2.3 Python Code for Redfish API Repsonse

```
base_url = 'https://"IP"/redfish/v1/Managers/1/SerialInterfaces/1'
dict_host = requests.get(base_url).json()
print (json.dumps(dict_host, indent=2))
```

Output:

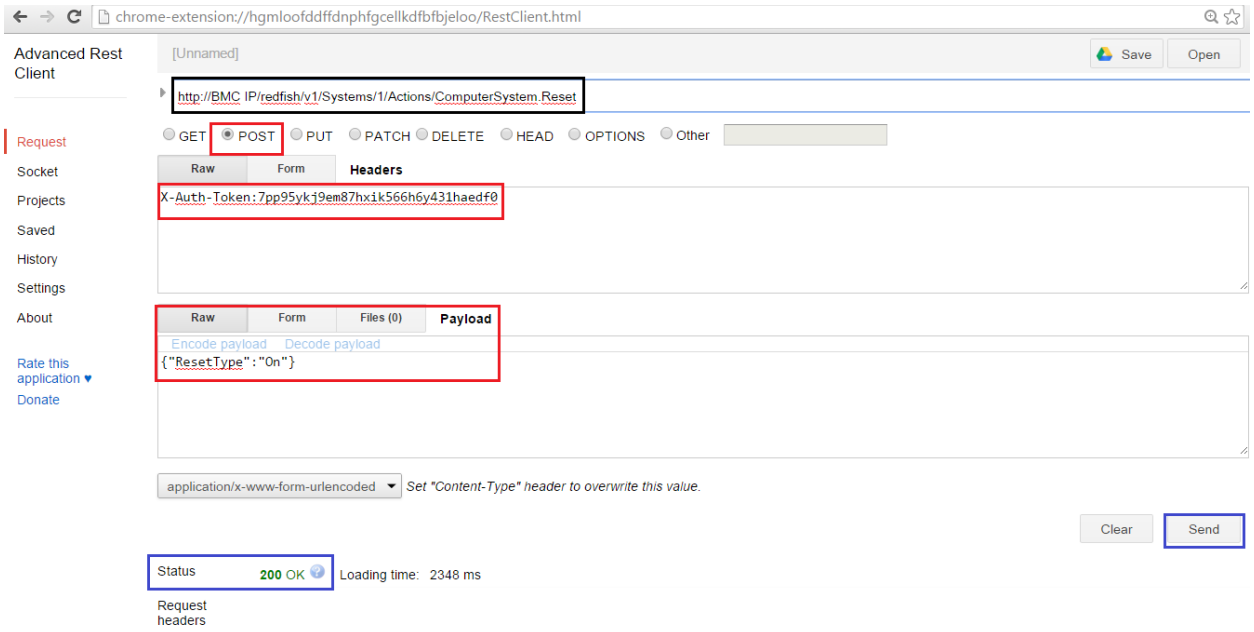

```
{
  "@odata.type": "#SerialInterface.1.0.0.SerialInterface",
  "Parity": "None",
  "Name": "SerialInterfaces",
  "DataBits": "8",
  "@odata.id": "/redfish/v1/Managers/1/SerialInterfaces/1",
  "@odata.context":
"/redfish/v1/Managers/1/SerialInterfaces/1/$metadata#Managers/Links/Members/1/Links/SerialInterfaces
/$entity",
  "FlowControl": "None",
  "SignalType": "Rs232",
  "StopBits": "8",
```


3.3 Posting an Action Item

The following Action items can POST with their respective URLs, which are supported by Redfish at this time.

- /redfish/v1/Managers/1/LogServices/Log1/Actions/LogService.Reset
- /redfish/v1/Managers/1/Actions/Manager.Reset
- /redfish/v1/Systems/1/Actions/ComputerSystem.Reset

Refer to the picture below. You should receive "status code 200"after posting any actions.



3.4 Fan Mode (OEM Feature)

This is a Supermicro OEM feature that is implemented under /redfish/v1/Chassis. Users can Patch (modify) the Fan Mode using following values.

The fan modes are defined as enum property as below: (User can use any of following names)

```
<EnumType Name="FanMode">
        <Member Name="Standard"/>
        <Member Name="FullSpeed"/>
        <Member Name="Optimal"/>
        <Member Name="HeavyIO"/>
        <Member Name="PUE"/> </EnumType>
```

Step: Use the Patch operation and parse the following payload for your system.

```
{
"Oem": {
     "OemFan": {
       "FanMode": "PUE"
     }
  }
}
```

3.5 Event Service

The event service is a new alert mechanism for Redfish. This alert will be sent out through http to the web server that is subscribed to by the users.

First, user needs to add a subscription to inform Redfish who will receive this event.

After user adds subscriptions, he can use the action “SendTestEvent” to send a testing event.

Alternatively, user can generate an event in the BMC and Redfish will automatically send an event alert to the destination(s) in the subscriptions. For this reason, you need to implement the event listener, which is like a web server that can receive https POST data that describes the Redfish event format.

For the current stage, user can launch Wireshark on the destination to sniff the packet to learn user receive the Redfish event.

****

To add a subscription:

[Post]https://IP/redfish/v1/EventService/Subscriptions/

{"Destination":"http://www.dnsname.com/Destination1","Context":"user1_test","EventTypes":["Alert","StatusChange"],"Protocol":"Redfish"}

- User can subscribe to a max. of 16 events.

To see all subscriptions:

[GET]https://IP/redfish/v1/EventService/Subscriptions/

To send a testing event:

[Post]https://IP/redfish/v1/EventService/Actions/EventService.SendTestEvent

{"EventType":["Alert"]}

User can delete events using the Delete service.

[DELETE]https://IP/redfish/v1/EventService/Subscriptions/1 (num)