MARWAY
POWER SOLUTIONS

RCM Technical Note — RESTful API

A Primer on RESTful API as it applies to Remote Monitoring and Control of RCM PDUs

Jan 2023 • r1

Marway Power Solutions
1721 S. Grand Ave., Santa Ana, CA 92705
800-462-7929 • marway@marway.com

Table of Contents

What is REST and RESTful API

This document provides a brief introduction to what REST is, what RESTful API is, and why you should consider using it for automation of Marway's Optima RCM power distribution units. The document repeats some of the introductory discussion from the Optima RCM User Guide. It does not repeat the technical reference details of the manual, but does provide a deeper look at the practical details of how to implement the API.

REST

REST is an acronym for **RE**presentational **S**tate **T**ransfer which at an academic level boils down to being an architectural style of how to represent information on a distributed hypermedia system. In other words, it's a particular style of how to design links and connections to information resources on a computer network (such as the internet).

It was originally described in 2000 (see the reference material sidebar), but has gained rapid popularity over the last decade. During that time, there's been significant chatter amongst software developers about the academic and pragmatic nuances. For the purposes the RCM software, the objectives and implications are far simpler than what has to be considered for more complex applications.

REST Reference Materials

The canonical academic paper which defined REST:

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Some additional internet discussions about REST:

http://en.wikipedia.org/wiki/Representational_state_transfer

https://developer.ibm.com/articles/ws-restful/

RESTful API

One pedantic point to cover is the difference between the terms REST and RESTful. REST is an architecture not a protocol (such as HTTP, FTP, etc.). Therefore, a system does not access information via REST. Rather, the data is accessed in a REST-like or RESTful manner. The latter term has the broader usage, and thus we end up with the term RESTful API—a programmer's interface to something using a RESTful design style. A lot of people will of course simplify this, and just call it REST. If you want to be technically accurate in your terminology, use RESTful API when describing your code. Otherwise, call it what you want to.

RESTful API in the Wild

RESTful APIs have become very widely used with internet-based services where one remote service exchanges information with another remote service. If you have any history with how the world wide web evolved in the 1990s and 2000s, you may remember the widespread adoption of XML, SOAP, and other standards to somewhat standardize data exchanges between legacy computing systems across the internet. RESTful APIs evolved as a bit of a reset in response to the complexity of those early systems.

At this point, RESTful APIs are the foundation of a lot of the "web APIs" and service "mashups" that integrate online data systems. Their commonality, popularity, and simplicity is starting to spill into the machine control world of industrial control. Where Modbus, SCPI, and other industrial buses have been common, the success of Ethernet as a factory bus is giving rise to the use of RESTful APIs as a communication protocol.

RESTful API for RCM

The RESTful API for RCM software is designed to support optimized machine-to-machine control of a PDU by a remote script such as in automated test and evaluation ("ATE"), in remotely managed LANs such as EGSE, and even in consolidating custom digital control panels to rack mounted equipment in forward operating service depots. The focus of the API is to provide access to power management features such as switching outlets and collecting power data where available. The API (as of this writing) does not support access to non-power-related resources such as Users, Logs, configuration settings, etc. It is possible to automate access to these resources using Telnet commands to create onboarding scripts. The User Guide provides the full dictionary of RESTful API resources.

RESTful API and HTTP

Since the REST architecture was originally conceived in the context of hypermedia (i.e. the world wide web), a fundamental tenet of REST is the use of HTTP as the communication protocol. We rely on the already defined elements of the HTTP protocol such as the URL, request form parameters, headers, and the response body.

You might want to review this page for an overview of HTTP message parts. Keep in mind that RCM uses HTTP 1.1. https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Programming for a RESTful API involves using an HTTP library for your programming language of choice in order to deliver an HTTP request to the PDU, and acquire an HTTP response which is usually already parsed into a data structure of header and body components for you. We'll look at these in more detail farther down.

RCM vs. a Web Page

Let's use a standard web browser as way to understand the RESTful API. When you load a specific web page, (Marway's RCM documentation web page for example), you'll notice in the web browser what's called the URL (the location or address of the page) such as `http://www.marway.com/rcm/docs`. Behind the scenes, an HTTP request was sent to the Marway web server request resource `/rcm/docs`. The server constructed an HTTP response to that request which is all the text and images which make up the page, and populated the `body` element of that HTTP response. There are additional elements to that response which you don't see in what are called HTTP headers—data fields with things like status codes, timestamps, and other meta data about the response.

We can correlate that request-response exchange to a remote control request-response with a PDU. Let's say we want to read the amps value of phase A of a three-phase power inlet on the PDU. We need an HTTP URL (a location or address) to this information such as `http://192.168.5.10/phases/1/amps`. With this URL we've asked for the amps of the first phase (which maps to phase A). For RCM, the response to this request is a simple number in string form, such as "`18.65`" which makes up the entire HTTP `body`. Again, there are some behind the scenes HTTP headers to pay attention to, but that's the simplistic gist of how RCM uses REST over HTTP to automate requests and responses.

Lists vs. Scalars

RESTful APIs as used for web service data exchanges have rich data structures. It's possible to requests lists such as items on purchase order, or products in a catalog, where each item in is list is itself a multi-field data structure. These data structures come in XML or JSON formats which get parsed into data structures native to the programming language being used.

For RCM, the API in its current form is intentionally very simple with an emphasis on scalars (single values) as responses which other than converting them to native data types of the programming language, do not need

parsing. RCM doesn't have "discovery" requests (i.e. how many phases?, how many outlets?). The programmer (and thus the code) must know the power structure. The PDU model number can be requested, and from that the code can branch or populate an internal power architecture model.

As you read about the various capabilities of classic RESTful APIs, keep in mind the RCM is much simpler.

Key Headers

HTTP headers are essentially meta data for the request and response payloads. One of the headers is named `Content-Type` which uses file MIME types to identify the format of the content in the HTTP `body`. The typical `Content-Type` for an RCM response is `text/plain`. Your code will test the `Content-Type` header to make sure it is equal to the "text/plain" string.

Another `Content-Type` value is "text/html". RCM uses this to flag that the content is an error code and not data. This is not standard RESTful API behavior, but within the limited scope of what the HTTP server in the embedded OS allows (which was not designed for RESTful API implementation), we had to use this as a flag instead of the preferred method.

Why Use the RESTful API

The RESTful API is provided as an alternative to the common practice of scripting to a Telnet interface or even SNMP. Telnet is intended to be a human interface. From the machine perspective, there's a lot of wordy noise to parse out to acquire just the data values needed. The RCM RESTful API eliminates all that noise. While SNMP is designed as a machine-to-machine interface, it can be complex to develop scripts with, and difficult for people to read. It's not a human friendly protocol. Since a RESTful API is basic HTTP, and is more human readable than SNMP, it can be both quicker to develop custom automation scripts with, and more efficient for the machine-to-machine dialog.

So far we keep referring to HTTP (as the protocol), but keep in mind that the RESTful API also works over HTTPS. With RCM specifically, the HTTPS implementation offers better encryption than the included SNMP with MD5/DES encryption.

Programming with RCM RESTful API

RCM Request Syntax

Each request will be made of up elements in the URL as well as additional HTTP form parameters. We refer to the key URL elements as the verb, noun, instance ID, and attribute. The form of a request URL like this (where the words starting with : are variables to be replaced):

```
POST https://192.168.0.10/outlets/1/switch?session_id=x
```

can be represented generically as this:

```
:verb https://192.168.0.10/:noun/:id/:attribute?:form_parameters
```

The noun is what REST refers to as the resource.

API Verbs

The RCM Software RESTful API makes use of GET, POST, PATCH, and DELETE behaviors. There is no need for PUT, and HEAD is not utilized in any meaningful way. However, the embedded web server, like most web servers, supports only GET, POST, and HEAD request methods. Since HEAD is not used, there's no conflict there. However, to provide support for PATCH and DELETE, the API uses a common technique of creating a pseudo-verb through a form parameter.

Whenever a URL needs GET or POST, the standard HTTP request method can be specified. However, when PATCH or DELETE are needed, the URL must be submitted with POST, and the request must include a form parameter named rest_method along with a string value of "PATCH" or "DELETE" as needed. For example, while the following would be ideal:

```
PATCH https://192.168.0.10/outlets/1/switch?new_value=on&session_id=abcxyz
```

it is necessary to accomplish that message this way:

```
POST https://192.168.0.10/outlets/1/switch?
new_value=on&session_id=abcxyz&rest_method=PATCH
```

This form-parameter technique is quite common in RESTful APIs since many web servers (and browsers) do not support all HTTP verbs. The world wide web got started with GET, POST, and HEAD, and thus we are left to create a work around for those verbs not widely supported yet.

API Nouns / Resources

The target or object of a RESTful API request is a resource. This is the noun of the request. In the RCM software, supported resources include: session, system, inlets, phases, circuits, outlets, and others. Not all resources are supported on every PDU.

API Attributes

Attributes are the details about an object (noun/resource). For an RCM system object, that includes the label, location, asset_id, contact, and more. For an outlet object, attributes include label, switch, rated_volts, and others.

For the most part, any object available in the RESTful API will support all of the attributes supported by the RCM command line for that same object. Sometimes, the CLI includes attribute groups such as power, rating, or switch_info, but these are not individual data point attributes, and would not be supported in the RESTful API.

API Form Parameters

These are request parameters passed as HTTP form parameters. In a POST, these will usually be part of the header, and in a GET they would normally be part of the URL, but either style works.

Common form parameters will be session_id and rest_method. Though some requests may have others.

Responses

As of this release, the API does not support discovery of the PDU power architecture (how many power inlets, phases, outlets, etc.). That is, it does not support requesting collections which could be parsed to navigate or "walk" available resources. It is assumed the power architecture of the PDU is a known entity which will remain stable over the life of a given script of automated tasks.

While these limitations might been seen as restrictive from the traditional RESTful API perspective, it helps to focus the limited resources of an embedded controller on the most value-added capabilities for the majority of applications.

Given the API's focus on working with specific individual attributes, responses are single-value text strings with the Content-Type of text/plain, and there should be no need to parse the HTTP response body.

If a request fails, or was malformed to start with, a value in the form of a negative number is returned. Additionally, the HTTP Content-Type will be set to text/html to indicate the body is an error (see side bar on page 164 of the version 2.5 User Guide "RCM REST Error Codes"). Therefore, while the body should not need parsed, the HTTP Content-Type header needs to be tested to determine if the response is an error or valid data.

Authentication API

RCM resources are not public. A conversation over the RESTful API must be initialized with an authentication which results in the creation of a session. Each subsequent request must be authenticated with a session ID.

A session ID is obtained with a POST to /session with parameters account=&password=. A 32-character string is returned. That string has to be submitted as part of all other requests in a parameter named session_id=.

The user can be any RCM user allowed access via the web. It is beneficial to define a user specific to each script using the RESTful API. Two sources logging into the PDU using the same name and password will interrupt each others sessions.

Reminder: for enhanced security on any network, the PDU should have HTTPS enabled to encrypt the authentication dialog which occurs between the automation script and the PDU web server.

RCM RESTful API Examples

RCM Resources in the User Guide

All RCM RESTful API resources (nouns) are described in the User Guide. In the examples below, refer to the User Guide for the details of the specific resources.

A GET Request/Response in HTTP Notation

For our first example, let's look at a request for the amps on phase 1 of the PDU. All requests for power objects are done using their IDs. If this were a single phase PDU, there is only one phase. If it were a three phase PDU the phases A, B, C are mapped to IDs 1, 2, 3. The IDs for these are always shown in the web UI and in the command line responses. These are the same IDs used for RESTful requests.

Let's map out what we are trying to do:

- The request is to fetch or GET a resource.
- The request is for power phase data, so that is the /phases resource (User Guide page 166)
- The request is for ID 1 of the phases.
- The request is for amps.

The resource format for this request is:

```
GET /phases/:id/amps
```

The request URL for this request will be (let's ignore the session_id for now):

```
GET https://IP_ADDRESS/phases/1/amps?session_id=SESSION_ID
```

If you had a session_id, and typed that URL (without the GET word) in a browser, the PDU would return a response in your browser window with only the text of something like "18.65" (no quotes). However, there is more to the response than just that text.

The full HTTP response to this request is below. Blue = header field name, red = the header field value, green = the HTTP body (the content you would see in a browser).

```
HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: no-cache,no-store
Expires: Thu, 26 Oct 1995 00:00:00 GMT
Content-Length: 5
Server: MarwayRCM/2.5.1
Set-Cookie: C9=1BIppfCyGz3op6CHS8B2R8j8GYhX3sh2; path=/
Connection: close

18.65
```

Let's look at each of these in more detail.

Response Headers

HTTP/1.1 — The first line returns the version of HTTP being used, and the HTTP status code. It should always be 200 OK (see side bar on page 164 of the User Guide).

Content-Type — Identifies the content type of the response body. For all successful requests this will be text/plain. For requests which result in an error this will be text/html. This is not standard for RESTful APIs, but due to limitations in the embedded HTTP server, this is what we have to use (see side bar on page 164 of the User Guide).

Cache-Control — A field to help browsers know whether to cache the request and response. For RCM, the value will always be no-cache,no-store — this field is essentially meaningless, and should be ignored.

Expires — A field to help browsers know when the data should be considered stale. For RCM, the value will always be a long past date — this field is essentially meaningless, and should be ignored. It will not reflect the time of the request or response.

Content-Length — Indicates the length of the response body (which is always a text string). For RCM, the value is accurate, and may or may not be useful to your application code.

Server — An identification string of the HTTP server. For RCM, it will report the version of the firmware.

Set-Cookie — One for each unique cookie. For RCM Restful API, these can generally be ignored. C9 has the session ID cookie—which is the same session ID your code would have submitted with the request. C4, C5, and C6 are not used during RESTful API sessions, and should be ignored.

Connection — An HTTP control field which controls whether the message session (not the RCM login session) should be closed or kept alive after the response is delivered. It will always be close for RCM.

Use POST to Obtain a Session ID

The verb POST is used to make a request for a new session. (The reason goes into the original academic naming of the HTTP verbs.)

The HTTP request URL (User Guide p165)

```
POST /session?account=NAME&password=PASSWORD
Example: POST http://192.168.5.10/session?account=restapi&password=pP8*word
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: no-cache,no-store
Expires: Thu, 26 Oct 1995 00:00:00 GMT
Content-Length: 32
Server: MarwayRCM/2.5.1
Set-Cookie: C9=1BIppfCyGz3op6CHS8B2R8j8GYhX3sh2; path=/
Connection: close

1BIppfCyGz3op6CHS8B2R8j8GYhX3sh2
```

Your script will save the value of the body which is the session_id to be passed to all subsequent requests.

Use PATCH to Change an Outlet On/Off State

The HTTP request URL (User Guide p167)

```
PATCH /outlets/:id/switch?session_id=:sessionID&rest_method=PATCH&new_value=:value
Example: POST http://192.168.5.10/outlets/4/switch?
              session_id=:1BIppfCyGz3op6CHS8B2R8j8GYhX3sh2
              &rest_method=PATCH
              &new_value=on
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Cache-Control: no-cache,no-store
Expires: Thu, 26 Oct 1995 00:00:00 GMT
Content-Length: 2
Server: MarwayRCM/2.5.1
Set-Cookie: C9=1BIppfCyGz3op6CHS8B2R8j8GYhX3sh2; path=/
Connection: close

on
```

Example Code in C#

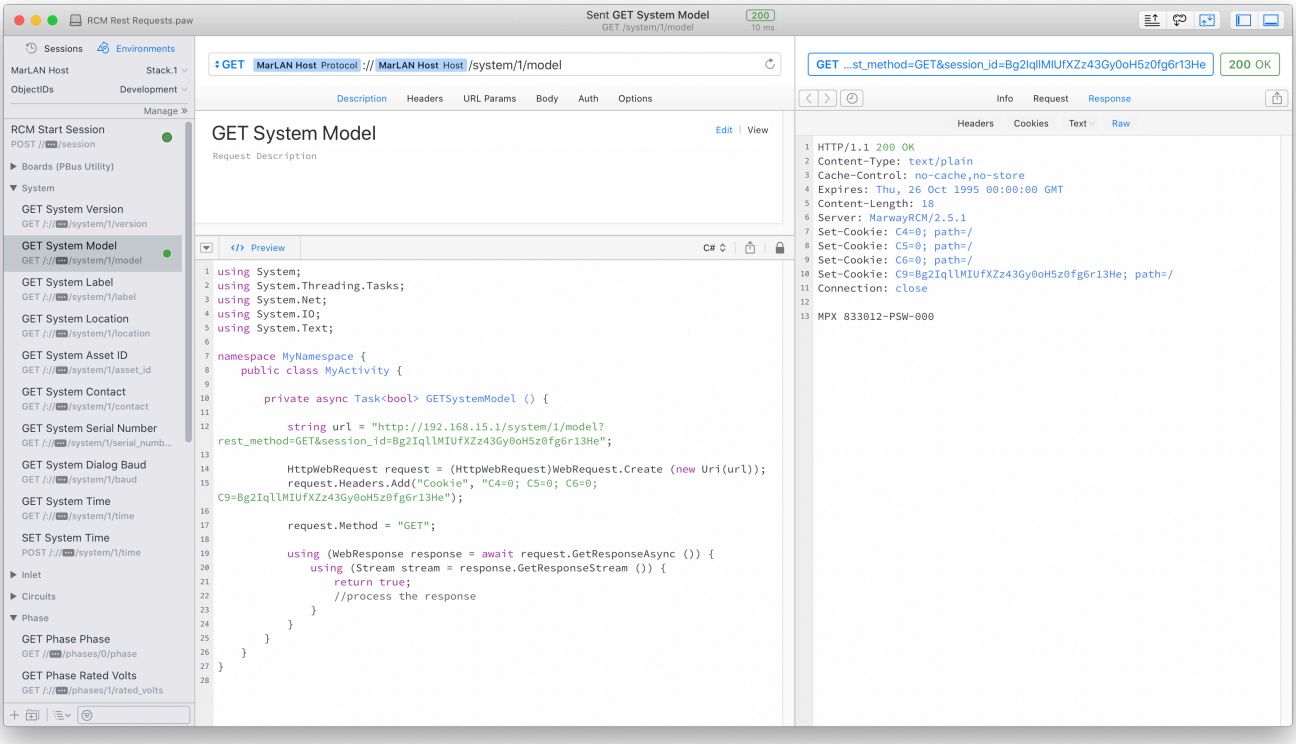
Using an API application, the following simplistic code was auto-generated to show how sending a request might be coded in the C# language. Obviously there are literals that need to be variables. There's also no error management, and it doesn't show how to extract the details from the response. However, what this shows is that the language's networking libraries, and in particular the HttpWebRequest, WebResponse, and related classes are used to manage the HTTP "conversation" with the PDU. There will be similar patterns in other languages.

```
using System;
using System.Threading.Tasks;
using System.Net;
using System.IO;
using System.Text;

namespace MyNamespace {
    public class MyActivity {

        private async Task<bool> GETPhaseAmps () {

            string url = "http://192.168.15.1/phases/1/amps?session_id=Bg2IqllMIUfXZz43Gy0oH5z0fg6r13He";

            HttpWebRequest request = (HttpWebRequest)WebRequest.Create (new Uri(url));
            request.Headers.Add("Cookie", "C4=0; C5=0; C6=0; C9=Bg2IqllMIUfXZz43Gy0oH5z0fg6r13He");

            request.Method = "GET";

            using (WebResponse response = await request.GetResponseAsync ()) {
                using (Stream stream = response.GetResponseStream ()) {
                    return true;
                    //process the response
                }
            }
        }
    }
}
```

Experiment Using an API Application

Relative to the rich API that a web service would have, the API for RCM is very simple. Still, you may find it useful to use a request generation application which can build and send requests to an RCM unit, or the RCM Simulator, much faster than writing code while you experiment. It's easier to quickly iterate trials while exploring and and building an understanding of the API.

For Windows, Mac, Linux there is Postman. For Mac there's Paw (now known as RapidAPI). This screenshot is Paw, just to give an idea what these tools are like. Saved requests on the left. Request details in the middle. Response showing HTTP headers on the right. Paw in particular also generates simplistic source code in a number of languages to give you a head start. Of course, the code will need modifications to use variables where there's magic literals, and to include proper error handling.



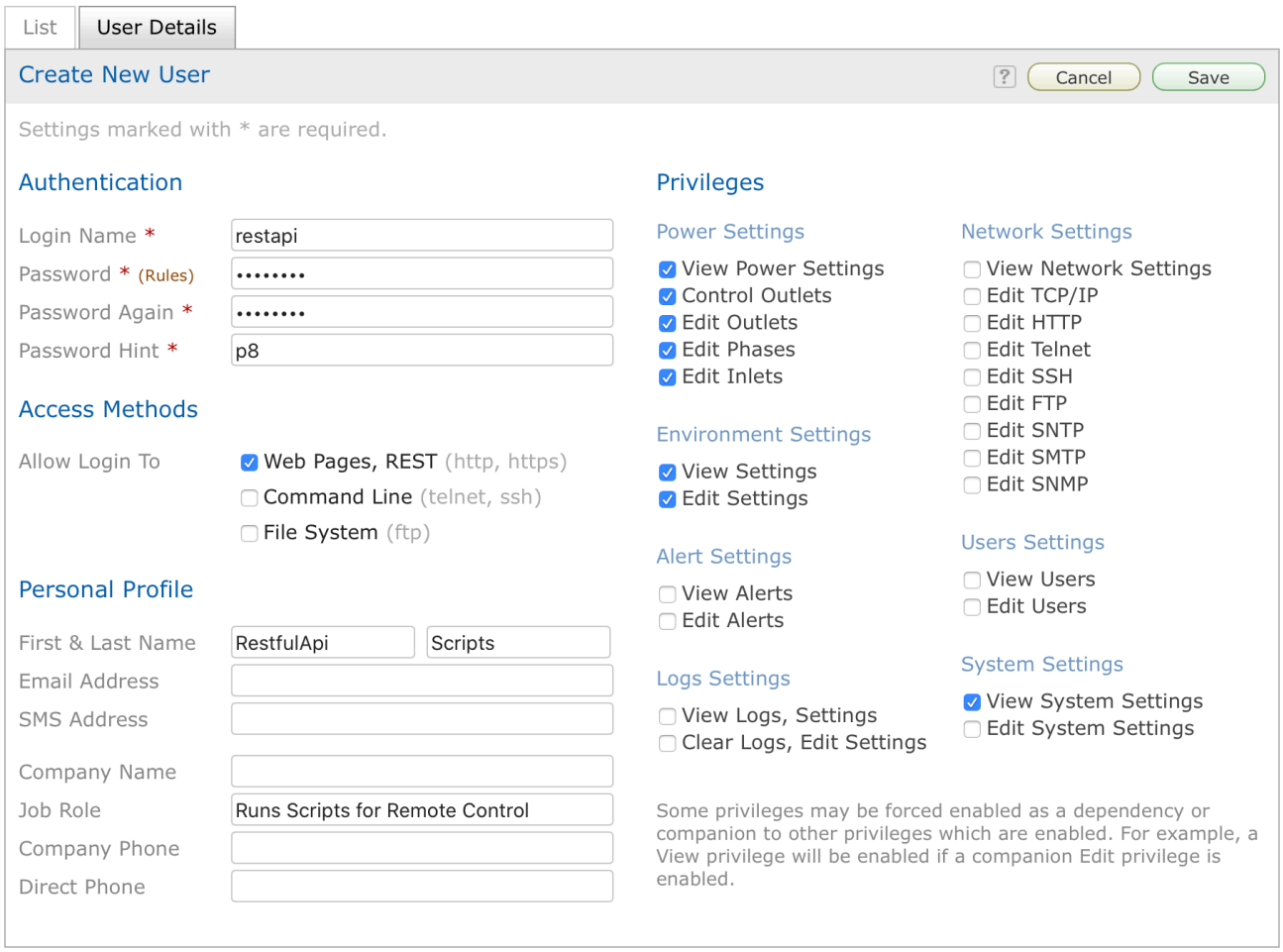
Configure a RESTful API User in RCM Web UI

It's best practice to create an RCM user account specific to each script or tool used to remotely connect to the RCM PDU. If two sources use the same account, the second login will disconnect the first login.

RESTful API is conducted over HTTP, so that access method must be allowed. The other access methods should be disabled for that account.

It's likely you'll want all power privileges granted, and environment privileges if you have T/H sensors connected. Note that the API allows for making changes to the power and environment setpoint values.

The View System privilege is necessary to acquire the model number, serial number, etc.



Change Log

R1b — Jan 23, 2023 — gw

- Initial Release (typos fixed)