Template:Networking rutxxx configuration example JSON RPC linux

□

Contents

Introduction

The information in this page is updated in accordance with the **[[Media:{{{fw_version}}}_WEBUI.bin|{{{fw_version}}}]]** firmware version.

JSON-RPC is a remote procedure call protocol encoded in JSON. It is a very simple protocol (and very similar to XML-RPC), defining only a few data types and commands. JSON-RPC allows for notifications (data sent to the server that does not require a response) and for multiple calls to be sent to the server which may be answered out of order.

This article provides a guide on how to use JSON-RPC on RUTxxx routers.

Configuration overview and prerequisites

Before we begin, let's overview the configuration that we are attempting to achieve and the prerequisites that make it possible.
Prerequisites:

- A PC for with a HTTP request software.
- An Internet connection. (*This example is based in a local configuration, but also can be used via wired WAN or a remote installation with Public IP*)
- One RUTxxx series router.

Configuration scheme: [[File:{{{file_scheme}}}|border|class=tlt-border]]

Enabling JSON-RPC

Before anything else, you'll need to make sure JSON-RPC is enabled on your router. JSON-RPC is enabled by default, so if you haven't made any changes to the router's access settings, everything should be in order. Otherwise you can check JSON-RPC status by logging into your router's WebUI and navigating to **System → Administration → Access Control**. Look for the *WebUI* section; there will be an **Enable JSON RPC** field. Make sure it is checked:

Using JSON-RPC (Linux)

This section describes how to use JSON-RPC commands with a Linux OS system. To find the guide for Windows users, jump to this section: **[[{{{link}}}|JSON-RPC on Widnows]]**

Obtaining a session ID

To log in to the router via JSON-RPC you must first obtain a Session ID. To do so, you must send an **HTTP POST** request to the router. Open the Linux **Terminal** app and execute this command:

```
curl -d "{ \"jsonrpc\": \"2.0\", \"id\": 1, \"method\": \"call\", \"params\":
[ \"00000000000000000000000000000000\", \"session\", \"login\", {
\"username\": \"root\", \"password\": \"admin01\"  } ] }"
http://192.168.1.1/ubus
```

The section highlighted in orange is the router's admin password. admin01 is the default value, replace it with your router's password. The address highlighted in green is the router's IP address. Replace this value with your router's IP. If you're trying to reach the router from **[[{{{name}}}_LAN|LAN]]**, use the local IP address (default: 192.168.1.1), if you're trying to reach the router from **[[{{{name}}} WAN|WAN]]**, use the router's WAN IP address.

The picture above depicts the process of obtaining a Session ID. The ID itself is encapsulated in a blue rectangle. Copy this ID as you will need it to authenticate yourself when using other commands.

Getting router parameters

Now that you have obtained a Session ID, you can issue commands to the router. Lets start with commands that return information about the router.

Getting RSSI

This is a command that returns the router's **RSSI**(signal strength) value:

```
curl -d "{ \"jsonrpc\": \"2.0\", \"id\": 1, \"method\": \"call\", \"params\":
[\"a74c8e07646f0da2bfddce35bf3de1f3\", \"file\", \"exec\", {
\"command\":\"gsmctl\", \"params\": [\"-q\"] } ] }" http://192.168.1.1/ubus
```

Highlighted in red is the Session ID. Replace it with the Session ID that was provided to you. Highlighted in orange is the command that we used for our query and highlighted in green is the parameter for the command: **gsmctl -q**.

The picture bellow is a visual representation of this example. Encapsulated in a blue rectangle is the answer to the *gsmctl -q* query: **-73 dBm**.

You can issue almost any Linux command in a similar manner. For example, if you wish to get a list of file names contained in the config folder, the Linux command to do so would be **ls /etc/config**, which, translated to JSON-RPC, would be:

```
curl -d "{\"jsonrpc\": \"2.0\", \"id\": 1, \"method\": \"call\", \"params\":
[\"a74c8e07646f0da2bfddce35bf3de1f3\", \"file\", \"exec\",
{\"command\":\"ls\", \"params\": [\"/etc/config\"] } ] }"
http://192.168.1.1/ubus
```

The command is encapsulated in an orange rectangle and the answer - in a blue one.

Setting router parameters

This section will describe how to use **uci set** commands in order to set or change various router parameters via JSON-RPC. For more general information about the usage and syntax of UCI commands, check out our **UCI command usage** guide.

UCI SET

The *uci set* command is used to set router parameters. As an example, lets try to change the router's **LAN IP address**. The command to do so looks like this:

```
curl -d "{\"jsonrpc\":\"2.0\", \"id\":1, \"method\":\"call\",
\"params\":[\"590bde71578da2fabfe77ba86c00e4e5\", \"uci\", \"set\", {
\"config\":\"network\", \"type\":\"interface\", \"match\":
{\"ipaddr\":\"192.168.1.1\"}, \"values\": {\"ipaddr\":\"192.168.56.1\"} } ]
}" http://192.168.1.1/ubus
```

The sections highlighted in orange describe the config file's name and section (in this case, network config and interface section). Highlighted in red is the option in the config file that you wish to change (in this case, the router's LAN IP address, **ipaddr**). Finally, highlighted in green is the value that will to replace the old value (in this case, change the router's LAN IP address to 192.168.56.1). If the command was issued successfully, you should see a Response like this:

UCI COMMIT

In order to commit the changes from RAM to flash memory, you must execute a **uci commit** command. Continuing from the example above, lets commit the changes made to the *network* config. The command to do so looks like this:

```
curl -d "{\"jsonrpc\":\"2.0\", \"id\":1, \"method\":\"call\",
\"params\":[\"9704f676709d9dedc98d7718c4e3e7d2\", \"uci\", \"commit\",
{\"config\":\"network\"} ] }" http://192.168.1.1/ubus
```

When committing changes, you will need to specify the name of the file where the changes took place (in this case, network, which is highlighted in orange). If the commit was successful, you should see the same message as before:

{"jsonrpc":"2.0","id":1,"result":[0]}

LUCI-RELOAD

In order for the changes to take effect, use the **luci-reload** command which restarts all of the router's services. The luci-reload command looks like this:

```
curl -d "{\"jsonrpc\":\"2.0\",\"id\":1, \"method\":\"call\",
\"params\":[\"428a9fa57f1a391db0bd1b865fa16bb5\", \"file\", \"exec\",
{\"command\": \"luci-reload\"} ] }" http://192.168.56.1/ubus
```

The command itself is highlighted in orange.

Setting Multiple Parameters

This next example describes how to set multiple parameters in a single config file with one command. Lets change the default configuration of the Ping Reboot function (ping_reboot config file):

```
curl -d "{\"jsonrpc\":\"2.0\", \"id\":1, \"method\":\"call\",
\"params\":[\"558a9b03c940e52f373f8c02498952e3\", \"uci\", \"set\",
{\"config\":\"ping_reboot\", \"match\":{\"enable\":\"0\",
\"host\":\"8.8.8.8\", \"packet_size\":\"56\"}, \"values\":{\"enable\":\"1\",
\"host\":\"8.8.4.4\", \"packet_size\":\"64\"} } ] }" http://192.168.1.1/ubus
```

The command above will **enable** the Ping Reboot function, set the host to ping to **8.8.4.4** and ping packet size to **64**. The default values are highlighted in orange and the new ones are highlighted in green.

Note: Remember always to use the commands in the order (set, commit, luci-reload)