Template:Networking rutxxx configuration example JSON RPC windows

□

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 (Windows)

This section describes how to use JSON-RPC with a Windows operating system. If you're using a Linux OS, jump to this section of the guide: **[[{{{link}}}|JSON-RPC with Linux]]**

HTTP POST

To login to the router via JSON-RPC you will need software capable of sending **HTTP POST** requests to the router. The simplest solution is to install an extension similar to Chrome "**Postman**" (download link **here**).
If you're using Firefox you can use **"RESTClient"** (download link **here**).Once you've installed the add-on, Click it to launch it:

Obtaining a session ID

First, you must obtain a **Session ID**. In order to do so, you must send a HTTP POST request to the router asking for it.

- Enter the router's IP address into the URL field **http://192.168.1.1/ubus** (*use LAN IP for local access, WAN IP for remote access*).
- Change the orange code fields with router's username and password (*Or change it into the **Postman User Auth.** field.*)
- Then paste the following command into the **Body or Content to send** field:

```
{
    "jsonrpc":"2.0", "id":1, "method":"call", "params":
    [
        "00000000000000000000000000000000", "session", "login",
        {
            "username":"root", "password":"admin01"
        }
    ]
}
```

The section highlighted in orange is the router's admin password which by default is admin01. Replace this part with your own router's password.

- Once you have everything in order, click **Send**:

Here's how the response should look like (The Session ID is shown in the sixth red rectangle):

- Copy the Session ID since you'll be needing it when issuing other commands to the router.

NOTE: if later on your commands stop working and you get a Response like this:

```
{
   "jsonrpc": "2.0",
   "id": 1,
   "error": {
       "code": -32002,
       "message": "Access denied"
   }
}
```

It probably means that your Session ID has expired so you'll need to ask for a new one. **A Session ID expires after 300 seconds (5 minutes).**

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

```
{
    "jsonrpc": "2.0", "id": 1, "method": "call", "params":
    [
        "bde01a2da4a6f4a515bb9466f90bc58a", "file", "exec",
        {
            "command":"gsmctl",
            "params":
            [
                "-q"
            ]
        }
    ]
}
```

The test highlighted in red is your Session ID, and highlighted in orange are the command and the parameter. In this example we're using a **gsmctl -q** command that returns the router's RSSI (signal strength) value.

Look for **stdout** in the post response: **"stdout":"-69\n**". This tells us that the router's current signal

strength is **-69 dBm**.

Getting Network Config

You can issue many SSH commands in a similar manner. For example, if you wish to check the *network* the command to do so would be:

```
{
    "jsonrpc": "2.0", "id": 1, "method": "call", "params":
    [
        "a74c8e07646f0da2bfddce35bf3de1f3", "file", "exec",
        {
            "command":"cat",
            "params":
            [
                "/etc/config/network"
            ]
        }
    ]
}
```

Again the command and the parameter are highlighted in orange. In this case the **cat** command is used to view the contents of the **/etc/config/network** file. The Response is:

Setting router parameters

To set parameters, is necessary to use three commands, they are **set**, **commit** and **luci-reload**

- The First one is used to set router parameters
- The second one is used to commit the changes from RAM to flash memory
- The third one is used to the changes take effect

We'll not go into detail on UCI commands in this article, but you can check out our **UCI command usage** guide for detailed examples.

UCI SET

The **uci set** command is used to set router parameters.

As an example, lets try to change the router's WiFi SSID. The command to do so looks like this:

```
{
    "jsonrpc":"2.0", "id":1, "method":"call", "params":
    [
        "9704f676709d9dedc98d7718c4e3e7d2", "uci", "set",
        {
            "config":"wireless",
            "type":"wifi-iface",
            "match":
            {
                "ssid": "Teltonika_Router"
            },
            "values":
            {
                "ssid":"9999"
            }
        }
    ]
}
```

- The sections highlighted in orange describes the config file's name and section. ***(In this case, wireless config and wifi-iface section).***
- Highlighted in red is the option of the config section that you wish to change. ***(In this case, the router's SSID.)***
- Finally, highlighted in green is the value that will replace the old value. ***(In this case, it change the router's SSID to 9999.)***
- Note: You should be sure that all the fields are correctly written

If the issued command was a success, you should see a Response like this:

UCI COMMIT

When you apply changes using *uci set*, you're only changing a copy of the file that is located in the router's RAM memory. In order for the changes to take place you'll need to issue a **uci commit** command that will commit the changes from RAM to flash memory. Continuing from the example above, lets commit the Wireless SSID changes. The JSON-RPC command to do so looks like this:

```
{
    "jsonrpc":"2.0", "id":1, "method":"call", "params":
    [
        "9704f676709d9dedc98d7718c4e3e7d2", "uci", "commit",
        {
            "config":"wireless"
        }
    ]
}
```

1. When committing changes, you will need to specify the name of the file where the changes took place ***(In this case, wireless config, which is highlighted in orange).***

   If the commit was successful, you should see the same message as before: 

   LUCI-RELOAD

   The last step to take in order for the changes to take effect is the **luci-reload** command which restarts all of the router's services. The *luci-reload* command looks like this:

```
{
    "jsonrpc": "2.0", "id": 1, "method": "call", "params":
    [
        "428a9fa57f1a391db0bd1b865fa16bb5", "file", "exec",
        {
            "command": "luci-reload"
        }
    ]
}
```

   The command itself is highlighted in orange.
   

   Setting Multiple Parameters

   This next example describes how to set multiple parameters in a single config file, in this case, changes the default DHCP server values with custom ones:

```
{
    "jsonrpc":"2.0", "id":1, "method":"call", "params":
    [
        "558a9b03c940e52f373f8c02498952e3", "uci", "set",
        {
            "config":"dhcp", "type":"dhcp", "match":
            {
                "start":"100",
                "limit":"150",
                "leasetime":"12h"
            },
            "values":
            {
                "start":"75",
                "limit":"100",
                "leasetime":"6h"
            }
        }
    ]
}
```

The command above will change the router's DHCP Server's current Start address, Address limit and Lease time values ***(highlighted in orange)*** to custom values provided in the **"values"** section of the command ***(highlighted in green)***.

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