User Scripts examples

□

Contents

Introduction

User Scripts is a service in RUT routers that provides the user with the possibility to create custom **shell scripts** that run at the end of router's boot process.

This article will provide some examples on the usage of User Scripts. Although it must be noted that one should be at least be partially familiar with the syntax and basic principles of shell scripts beforehand as this guide will not provide a tutorial on shell scripts themselves.

Example 1: 1-Wire Data to Server for TRB141

Since there is no Web interface support for getting 1-wire data, it has to be accessed via SSH or Console modes. As of now, the Data to Server function for 1-wire sensor data can only be done by writing a custom script since there are no direct methods of storing this data.

This article will provide a step-by-step configuration on how to configure data to server using a custom script that allows 1-wire data to be sent on an HTTP server.

For the instruction on how to setup 1-Wire sensor with the TRB141 kindly navigate to this article that we have.

Disclaimer

---

1) This configuration together with the custom script is tested and written on the firmware version TRB1_R_00.07.04.2 of TRB141.

2) The provided script is written for example purposes only as our devices, specifically TRB141, doesn't have a built-in functionality to send 1-wire data to a specific server.

Writing the Script

Connect to the TRB141 CLI via SSH or WebUI.

Create a new directory for you to save the script that is to be created later. (Optional)

Navigate to it, then create a new sh file using the “vi” command followed by the filename of your choice. (Don’t forget to add the .sh extension at the end).

```
vi 1wire_script.sh
```

A text editor window will open and press the letter ‘i’ to enter insert mode.

Copy the provided code and paste it into your script file.

```
#!/bin/sh

# Capture output of ubus call ioman.gpio.onewire status command

status=$(ubus call ioman.gpio.onewire status)

# Parse value of "value" parameters
value=$(echo "$status" | grep -o '"value":[^,}]*' | sed 's/.*:
*"\?\([^,"}]*\)"\?/\1/')

if [ "$value" -eq 0 ]; then

  ubus call ioman.gpio.onewire update '{"value":"1"}'

fi

BASE_DIR="/sys/bus/w1/devices/"

do
# Iterate through each sensor directory
for sensor_dir in $BASE_DIR/28-*; do

  sensor_id=$(basename $sensor_dir)

  # Read temperature data from device file and filter it to display up to 2
decimal points

  temp_raw=$(cat $sensor_dir/w1_slave | grep "t=" | awk -F"=" '{print $2}')
```

```
  temp=$(echo "$temp_raw" | awk '{printf "%.2f", $1/1000}')

  # Define HTTP server URL and payload data

  SERVER_URL="192.168.2.227"

  PAYLOAD="{\"SensorID\": \"$sensor_id\", \"temperature\": $temp}"

  # Send temperature data to server using cURL

  curl -X POST -H "Content-Type: application/json" -d "$PAYLOAD" $SERVER_URL

done
```

To use this script, replace the 'DEVICE_FILE', 'SERVER_URL', and 'PAYLOAD' variables with the appropriate values for your system and HTTP server. Press the 'esc' on your keyboard to exit insert mode, then type :x, and press enter.

Run the command: **'chmod 777 <filename>'** to make the script executable on your end.

Code explanation

This line uses **'ubus'** command to call the **'ioman.gpio.onewire'** service and execute the **'status'** command. The output of this command is saved to the *status* variable.

This block of code extracts the value of the **'value'** parameter from the 'status' output using **'grep'**and **'sed'** and stores it in the 'value' variable.

After that, it checks if the value stored in the **'value'** variable is equal to 0. If it is, the **'ubus'** command is used to call the **'ioman.gpio.onewire'** service and execute the 'update' command, which updates the 'value' parameter to 1.

This line sets the 'BASE_DIR' variable to the base directory for 1-wire devices.

This line sets up a 'for' loop to iterate through all directories in '$BASE_DIR' that start with '28-', which is the prefix for 1-wire temperature sensors.

After that, it extracts the sensor ID from the sensor directory path using the 'basename' command and stores it in the 'sensor_id' variable.

This block reads the temperature data from the device file **'$sensor_dir/w1_slave'** and extracts the raw temperature value using **'grep'** and **'awk'**. The temperature value is encoded as an integer.

Which then converts the raw temperature value to degrees Celsius by dividing it by 1000 and then formatting it with two decimal places using **'printf'**. The resulting temperature value is stored in the **'temp'** variable.

These lines define the URL of the server to which the temperature data will be sent and the payload data that will be sent with each request. The payload includes the **'sensor_id'** and **'temp'** variables, formatted as a JSON object.

This line uses the **'curl'** command to send an HTTP POST request to the server specified by **'$SERVER_URL'**. The **'-X POST'** option specifies that the request should be a POST request, and the **'-H "Content-Type: application/json"'** option specifies that the payload data is in JSON format. The **'-d "$PAYLOAD"'** option specifies the payload data to be sent with the request.

Output

]

Run the script by entering the file path of the file or **./<filename>.sh** if you are in the same directory.

If configured correctly, you should be able to see the temperature data on your HTTP server.

Example 2: Mobile Parameters to HTTP Server

Mobile parameters are needed to measure signal features that can be acquired using **gsmctl** commands. However, there is no available command to check/display all of the desired details.

With that said, a custom script is needed to show them using one command. This article provides step-by-step instructions on how to write the script, programs that are required, and how to run it.

If you're having trouble finding any page or some of the parameters described here on your device's WebUI, you should turn on **"Advanced WebUI"** mode. You can do that by **clicking** the **"Basic"** button **under "Mode"**, which is located at the top-right corner of the WebUI.

Disclaimer

---

1) This page describes a custom script that is tested on the latest firmware version at that time (RUTXXX_R_00_07_04_2).

2) While we strive to keep the information accurate and up-to-date, these are exclusively meant for illustrative purposes only. It's important to clarify that the functionalities being discussed are not inherent features of our routers. Any reliance you place on the information within this document/website/article is strictly at your own discretion.

Prerequisites

For this particular configuration you will need:

- A Gateway or a Router with GSM (RUTX14 is being used in the example)
- 1 SIM card
- 1 PC
- HTTP Server (Hercules)

Preparation

- Prepare RUTX14, power up the device, insert the sim card, check that mobile interface is active and working. SIM1, PWR and signal strength indicators should light up.

Network -> Interfaces should look similar to this:

Making a Directory and Script

---

1. Connect to RUTX14 CLI using PUTTY or WEBUI.

2. Create a new directory for the custom script with the following command: `mkdir /etc/config/script`

3. Then create a new .sh file: `touch /etc/config/script/<filename>.sh`

4. To edit the filename, use `vi <filename>.sh`. Press **I**. To save and exit, **Press Esc** then **:x** or to exit without saving, **Press Esc** then **:q!**.

5. After editing, save then run command `chmod +x <filename>` to make the script executable.

Testing

---

6. Run the script: `/etc/config/script/<filename>` or if you are inside script directory just run `./<filename>`

7. If all configurations are correct, all parameters should be displayed in the HTTP Server

Script

To get a copy of the whole script, please refer to this file.