UCI command usage

The information on this page is updated in accordance with the **00.07.4** firmware version .

Unified Configuration Interface (**UCI**) is a small utility written in C (a shell script-wrapper is available as well) and is intended to centralize the whole configuration of a device running on OpenWrt.

□

Contents

Summary

UCI commands provide the user with the maximum degree of control since they can be issued via many different forms of router monitoring and administration (SSH, CLI, SMS, JSON-RPC) and can be used to set or get any router parameter. This chapter is a guide on how to use UCI commands with RUT devices.

How do I execute UCI commands?

UCI commands can executed via the following methods:

- **SSH** - you can use UCI commands via SSH either with Linux OS's Terminal app or the PuTTY app with Windows OS (a download link is provided at the bottom of this page)

- **CLI** - you can use UCI commands via the Command Line Interface found in the router's **WebUI**
- **SMS** - you can execute UCI commands via SMS messages with the SMS Utilities **uci api** rule
- **JSON-RPC** - you can execute UCI commands via JSON-RPC. Refer to this guide for more information: **Monitoring via JSON-RPC**

Available commands

This section provides a list of possible UCI commands and options.

UCI commands

Command	Target	Description
batch	-	Executes a multi-line UCI script which is typically wrapped into a here document syntax
export	[<config>]	Exports the configuration in a machine readable format. It is used internally to evaluate configuration files as shell scripts
import	[<config>]	Imports configuration files in UCI syntax
changes	[<config>]	Lists staged changes to the given configuration file or if none given, all configuration files
commit	[<config>]	Writes changes of the given configuration file, or if none is given, all configuration files, to the filesystem. All "uci set", "uci add", "uci rename" and "uci delete" commands are staged into a temporary location until they are written to flash with the "uci commit" command. This is used exclusively for UCI commands and is not needed after editing configuration files with a text editor
add	<config> <section-type>	Adds an anonymous section of type *section-type* to the given configuration
add_list	<config>.<section>.<option>=<string>	Adds the given *string* to an existing list option
del_list	<config>.<section>.<option>=<string>	Removes the given *string* from an existing list option
show	[<config>[.<section>[.<option>]]]	Shows the given option, section or configuration in compressed notation. If no option is given, shows all configuration files
get	<config>.<section>[.<option>]	Gets the value of the given option or the type of the given section
set	<config>.<section>[.<option>]=<value>	Sets the value of the given option, or add a new section with the type set to the given *value*
delete	<config>[.<section>[[.<option>][=<id>]]]	Deletes the given section or option
rename	<config>.<section>[.<option>]=<name>	Renames the given option or section to the given name
revert	<config>[.<section>[.<option>]]	Reverts the given option, section or configuration file. Used to undo any changes performed with UCI and not yet committed with *uci commit*
reorder	<config>.<section>=<position>	Moves the specified section to the given *position*. Used for easier management purposes

Options

```
-c  <path>  set the search path for config files (default: /etc/config)
-d  <str>   set the delimiter for list values in uci show
-f  <file>  use <file> as input instead of stdin
-m          when importing, merge data into an existing package
-n          name unnamed sections on export (default)
-N          don't name unnamed sections
-p  <path>  add a search path for config change files
-P  <path>  add a search path for config change files and use as default
-q          quiet mode (don't print error messages)
-s          force strict mode (stop on parser errors, default)
-S          disable strict mode
-X          do not use extended syntax on 'show'
```

Configuration hierarchy

UCI commands can be used to set and obtain parameters, but to do so, one has to first know the names of the **config** file, its **section** and the **option** that they are trying to interact with. Different configurations for different router functions and services are stored in config files. These config files have sections and section usually store multiple options

The elements in the UCI model are:

- config: main configuration groups like network, system, firewall. Each configuration group has it's own file in /etc/config
- sections: a config is divided into sections. A section can either be named or unnamed
- types: a section can have a type. E.g, in the network config we typically have sections of the type "interface"

- options: each section has options that hold configuration values
- values: value of an option

Sections

Sections deserve some extra explanation in regard to naming. A section can be **named** or **unnamed**. Unnamed sections will get an autogenerated ID/CFGID (like "cfg023579") and be presented with an anonymous-name (like "@wifi-iface[0]")

Example of **anonymous-name** (cmd: uci show wireless):

```
...
wireless.@wifi-iface[0]=wifi-iface
wireless.@wifi-iface[0].device=radio0
wireless.@wifi-iface[0].network=lan
wireless.@wifi-iface[0].mode=ap
...
```

Example of **autogenerated ID/CFGID** (cmd: uci show wireless.@wifi-iface[0]):

```
...
wireless.cfg023579=wifi-iface
wireless.cfg023579.device=radio0
wireless.cfg023579.network=lan
wireless.cfg023579.mode=ap
...
```

Configuration files

This section provides a list of all available configuration files of RUT routers. Note that these are all possible config files from any RUT router (RUT230, RUT240, RUT850, RUT950, RUT955) and that therefore some of them may not exist in your router.

File	Description
/etc/config/avl	Stores AVL (Automatic Vehicle Location) configuration settings
/etc/config/buttons	Defines the behavior of the reset button on the device
/etc/config/call_utils	Provides utilities for making and managing calls
/etc/config/cli	Defines command-line interface (CLI) settings
/etc/config/dhcp	Configures Dynamic Host Configuration Protocol (DHCP) settings for the network
/etc/config/dropbear	Configures settings for the Dropbear SSH server
/etc/config/email_to_sms	Configures settings for the email-to-SMS gateway
/etc/config/events_reporting	Configures settings for reporting system events

/etc/config/firewall	Configures firewall settings
/etc/config/fstab	Configures file system mount points
/etc/config/gps	Configures settings for the Global Positioning System (GPS)
/etc/config/hwinfo	Provides hardware information about the device
/etc/config/iojuggler	Provides utilities for managing input/output (IO) settings
/etc/config/ioman	Provides utilities for managing input/output (IO) settings
/etc/config/ip_blockd	Configures IP address blocking settings and stores blocked IP addresses
/etc/config/ipsec	Configures settings for the IPsec VPN
/etc/config/kmod_man	Manages kernel modules
/etc/config/mdcollectd	Collects and sends system metrics to a remote server
/etc/config/modbus	Configures settings for the Modbus protocol
/etc/config/modbus_data_sender	Sends Modbus data to a remote server
/etc/config/modbus_master	Configures the device as a Modbus master.
/etc/config/modbusgateway	Configures the device as a Modbus gateway.
/etc/config/multi_wifi	Configures settings for multiple Wi-Fi networks.
/etc/config/mwan3	Configures settings for Multi-WAN load balancing and failover.
/etc/config/network	Configures network settings.
/etc/config/ntpclient	Configures settings for the Network Time Protocol (NTP) client.
/etc/config/ntpserver	Configures settings for the NTP server.
/etc/config/openvpn	Configures settings for the OpenVPN VPN.
/etc/config/operctl	Provides utilities for managing system operations.
/etc/config/overview	Provides an overview of the system.
/etc/config/p910nd	Configures settings for the p910nd printer server.
/etc/config/package_restore	Restores installed packages after formware update.
/etc/config/periodic_reboot	Configures periodic system reboots.
/etc/config/ping_reboot	Configures system reboots triggered by ping responses.
/etc/config/post_get	Configures POST/GET service.
/etc/config/pptpd	Configures settings for the PPTP VPN server.
/etc/config/profiles	Configures profiles for the system.
/etc/config/quota_limit	Configures mobile Data Limits.
/etc/config/rms_mqtt	Configures settings for the RMS connect.
/etc/config/rpcd	Configures settings for the Remote Procedure Call (RPC) daemon.
/etc/config/rs_console	Configures settings for the serial console.
/etc/config/rs_modbus	Configures settings for the Modbus protocol over serial.

/etc/config/rs_modem	Configures settings for the modem.
/etc/config/rs_overip	Configures settings for the serial over IP protocol.
/etc/config/rut_fota	Configures settings for firmware over the air (FOTA) updates.
/etc/config/sim_switch	Configures settings for switching between SIM cards.
/etc/config/simcard	Configures settings for the SIM card.
/etc/config/sms_gateway	Configures settings for the SMS gateway.
/etc/config/sms_utils	Provides utilities for managing SMS messages.
/etc/config/snmpd	Configures the Simple Network Management Protocol (SNMP) daemon.
/etc/config/snmptrap	Configures settings for SNMP traps.
/etc/config/snmptrap-opkg	Installs and configures the SNMP trap package.
/etc/config/socat	Configures settings for the socat command-line utility.
/etc/config/system	Configures system settings.
/etc/config/telnetd	Configures settings for the Telnet daemon.
/etc/config/uhttpd	Configures settings for the HTTP server.
/etc/config/user_groups	Configures user groups.
/etc/config/vuci	Configures settings for the VuCI web interface.
/etc/config/widget	Configures widgets for the web interface.
/etc/config/wireless	Configures wireless network settings.
/etc/config/xl2tpd	Configures settings for the L2TP VPN server.

Obtaining parameters

This section will overview uci get and uci show commands used to obtain router parameters, option and section names and contents of entire configs or sections.

UCI get

The **uci get** command returns values for specific options. When using uci get, you have provide the correct path to the option that you are looking for. For example, in order to obtain the Wi-Fi Access Point's SSID you would have to use a command that looks like this:

```
# uci get wireless.@wifi-iface[0].ssid
```

Response:

The command above returns the Wi-Fi Access Point's SSID. As you can see the uci get command is used. What follows after the command is the path to the value that we're looking for (SSID, in this case). The SSID value can be found in the wireless config, the @wifi-iface[0] section, stored under an option called ssid. So the basic syntax for a uci get command is this:

```
# uci get <config>.<section>[.<option>]
```

UCI show

If you don't know what the exact option is called and in which section of what config file it is stored, you can use the **uci show** command. uci show can also be used to obtain values of specific options, but it is more commonly used to display the contents of entire sections or configs. Lets modify the example above by saying that want to find out the SSID value but don't know the exact section or option under which the value is stored. In this case we'll the uci show command to view the contents of the entire wireless config:

```
# uci show wireless
```

Response:

As you can see, the response shows the entire wireless config and its entities. Note that instead of just showing values (like in the case of uci get) you can see the config name, section name and option name before each one.

Most config file names are simple. Wireless config is called wireless, OpenVPN config is called openvpn, etc. But even so one doesn't necessarily have to know what a config file is called, especially before interacting with it. To see the names of all config files and what kind of settings they store you can refer to the **table above**. Or if you're CLI or SSH and want to check the names of config files on the spot, you can use the **ls** command. Since RUT configs ar stored in **/etc/config**, the full commands should look like this:

```
# ls /etc/config
```

The ls command is used to view the contents of a directory. Here is an example of the /etc/config directory of a RUT955 router:

So when you plan on obtaining specific parameters or setting parameter values, you should always start with finding out option and section names. To accomplish this, we recommend using the uci show <config> commands.

Setting parameters

UCI can also be used to set parameters, add lists of parameters and even add entire sections to config files.

UCI set

The **uci set** command is used to set the values of specific options. It can set only one option at a time. For example, this time lets try changing the Wi-Fi Access Point's SSID to *wifi_set_by_uci*:

```
# uci set wireless.@wifi-iface[0].ssid=wifi_set_by_uci
```

As you may have noticed, the command is very similar to uci get, except it has an equals to ('**=**') sign added at the end and after the sign is the value that we want to assign to the option.

The next step is to commit the changes by using the **uci commit** command and to restart all the services relevant to our configuration by using the **/etc/init.d/wireless restart** command:

```
# uci commit wireless
# /etc/init.d/wireless restart
```

After this, your changes will be applied and in use. Notice that when using uci commit you can specify the config file for which you want to commit changes (you can even specify the exact section and option). This is useful when making changes to multiple options in case you make any mistakes, because before committing any changes you can easily undo them with the **uci revert** command. The command by itself will undo all the changes made by uci up until the last commit. It can also be used on specific config files, sections and options in order to undo specific changes.

UCI add_list

Some variables hold more than one value, unlike options. These variables are called **lists**. For example, if you use MAC filter on your Wi-Fi Access point, the MAC addresses are saved not as options but as a list.

Example of maclist (cmd: uci show wireless):

```
...
wireless.@wifi-iface[0].macfilter=deny
wireless.@wifi-iface[0].maclist=15:15:12:64:66:14 15:15:12:64:66:15
15:15:12:64:66:16
...
```

As an add_list usage example, lets add these MAC addresses to the list: 11:11:11:11:11:11, 22:22:22:22:22:22, 33:33:33:33:33:33

```
# uci add_list wireless.@wifi-iface[0].maclist=11:11:11:11:11:11
# uci add_list wireless.@wifi-iface[0].maclist=22:22:22:22:22:22
# uci add_list wireless.@wifi-iface[0].maclist=33:33:33:33:33:33
# uci commit wireless
# /etc/init.d/wireless restart
```

Notice that you have to use a separate command for adding each value and as with uci set you have to use uci commit and luci-reload in order for the changes to take effect.

Extensive example

With all that we have learned lets try a more complicated example: lets you want to create an OpenVPN server. The server will be called **MyServer**, will use a **TUN** type interface and **TLS** authentication. In order to create this server we will first have to create a section for the server in

the openvpn config:

```
# uci add openvpn server_MyServer
# uci set openvpn.server_MyServer=openvpn
```

The first line creates a section called *server_MyServer*, the second line specifies the section type, in this case - *openvpn*. Now lets add the rest of the configurations:

```
# uci set openvpn.server_MyServer.persist_key=1
# uci set openvpn.server_MyServer.port=1194
# uci set openvpn.server_MyServer.keepalive=10 120
# uci set openvpn.server_MyServer.persist_tun=1
# uci set openvpn.server_MyServer.status=/tmp/openvpn-
status_server_MyServer.log
# uci set openvpn.server_MyServer.verb=5
# uci set openvpn.server_MyServer.proto=udp
# uci set openvpn.server_MyServer.dev=tun_s_MyServer
# uci set openvpn.server_MyServer.enable=1
# uci set openvpn.server_MyServer.comp_lzo=yes
# uci set openvpn.server_MyServer.cipher=BF-CBC
# uci set openvpn.server_MyServer._auth=tls
# uci set openvpn.server_MyServer._tls_cipher=all
# uci set openvpn.server_MyServer.server=10.0.0.0 255.255.255.0
# uci set
openvpn.server_MyServer.ca=/lib/uci/upload/cbid.openvpn.server_MyServer.ca
# uci set
openvpn.server_MyServer.cert=/lib/uci/upload/cbid.openvpn.server_MyServer.cer
t
# uci set
openvpn.server_MyServer.key=/lib/uci/upload/cbid.openvpn.server_MyServer.key
# uci set
openvpn.server_MyServer.dh=/lib/uci/upload/cbid.openvpn.server_MyServer.dh
# uci set openvpn.server_MyServer.client_config_dir=/etc/openvpn/ccd
# uci add_list openvpn.server_MyServer.push="route 192.168.1.0 255.255.255.0"
# uci add_list openvpn.server_MyServer.push="route 192.168.56.0
255.255.255.0'
```

And don't forget to *uci commit* and *restart the daemon*:

```
# uci commit openvpn
# /etc/init.d/openvpn restart
```

A few notes about the configuration:

1. The options that go into an OpenVPN server are standard OpenWRT OpenVPN server options. If you do not posses all the required information needed to create an OpenVPN server, visit this OpenWRT guide: OpenVPN Setup Guide for Beginners.
2. Note that I added two values to the list named **push**. As mentioned before, when adding values to list-type parameters use separate commands for separate values. If the value has a space in it (as in the example above) use quotation marks around the value ("<*value*>").
3. Depending on your chosen authentication, the OpenVPN server instance might use certificate files for authentication with clients. A TLS server, as in our case, uses **Certificate authority**

(*.crt*), **Server certificate** (*.crt*), **Server key** (*.key*) and **Diffie Hellman Parameters** (*.pem*) files for authentication. A Static Key server uses a **Static Key** (*.key*) file for authentication. In the example above I had all the files upload beforehand to */lib/uci/upload*, so the commands that I used only provided the server's config with the paths to the files. When creating your own OpenVPN server you will have to generate your own certificates and upload the to */lib/uci/upload* (the default directory for certificates) or somewhere else, but make sure to specify the correct path. To upload files to the router use the **scp** command if you're working with a Linux type OS or use software called **WinSCP** if you are using Windows OS. Or use Easy-RSA to create certificates within the router. The newly created certificates will appear in **/etc/easy-rsa/keys**. You can create certificates with these commands:

```
build-ca
build-dh
build-key-server my-server
build-key-pkcs12 my-client
```

Additional examples

If the examples and explanations provided above did not suffice, we are providing this section of some additional ones in hopes to give you a better grasp of the syntax of UCI command usage.

Site Blocking

This example will provide instructions on how to enable RUT routers' Site Blocking feature and how to add hostnames to the Blacklist or Whitelist using only UCI commands. For the sake of our example lets say that you want to create a Blacklist that excludes access to all sites contained within the list. The sites in question are www.facebook.com, www.youtube.com and 9gag.com.

To achieve such a task, the first relevant piece of required information is the config name, **hostblock**, where all the necessary configuration settings are stored. The next important thing to know is that each different website must be stored in a separate section of the type **block**. So we'll need to create a new section and enable each added element. Lets start:

First element:

```
# uci add hostblock block
# uci set hostblock.@block[0].host=www.facebook.com
# uci set hostblock.@block[0].enabled=1
```

Second element:

```
# uci add hostblock block
# uci set hostblock.@block[1].host=www.youtube.com
# uci set hostblock.@block[1].enabled=1
```

Third element:

```
# uci add hostblock block
# uci set hostblock.@block[2].host=9gag.com
# uci set hostblock.@block[2].enabled=1
```

Enabling Site Blocking:

```
# uci set hostblock.config.enabled=1
```

Final steps:

```
# uci commit hostblock
# /etc/init.d/hostblock restart
```

The first-third steps add hostnames of the websites to be blocked, which are saved under the option *host*. Each of the first three elements also need to be enabled, therefore, the option *enabled* is set to *1* next to each host. The fourth step is for enabling the Site Blocking service (by setting the option *enabled* in section *config* to *1*).

DHCP Server

This example will provide instructions on how configure RUT routers' DHCP Server using only UCI commands. For the sake of the example lets say that you want to change the dhcp range to *192.168.1.2 - 192.168.1.200* and the lease time to *30 minutes*

To achieve such a task, the first relevant piece of required information is the config name, **dhcp**, where all the necessary configuration settings are stored. *Lets start:*

Setting start address and limit:

```
# uci set dhcp.lan.start=2
# uci set dhcp.lan.limit=199
```

Setting lease time

```
# uci set dhcp.lan.leasetime=30m
```

Final steps:

```
# uci commit dhcp
# /etc/init.d/dhcp restart
```

The first step sets the start address to 2 and the limit of addresses to 199. The value of the *start* option is associated with the last section of an IP address (if start value is **2** then the starting IP address is 192.168.1.**2**(provided that the router's LAN IP is in the 192.168.1.0/24 network)), the value of the *limit* option denotes how many IP addresses can be leased out starting from and including the the start address. Then the second step is used to set the lease time. The *letter* option specifies the unit of time measurement (either *m* for minutes or *h* for hours). The *time* option specifies number of minutes (or hours in other cases) and the *leasetime* option is just the representation (nonetheless, it's still mandatory) of the previous two values, i.e., 30m - thirty minutes.

Mobile Data Limit

This example will provide instructions on how configure Mobile Data Limit and SMS Warning on RUT routers' using only UCI commands. For the sake of the example lets say that you want to set up a data limit of 1 GB with the limit counter restarting everyday at 10 a.m. and an SMS Warning that sends out a message when the 800 MB threshold is reached that also restarts everyday at 10 a.m.

To achieve such a task, the first relevant piece of required information is the config name, **quota_limit**, where all the necessary configuration settings are stored:

Enabling Mobile Data Limit and SMS Warning:

```
# uci set quota_limit.mob1s1a1=interface
# uci set quota_limit.mob1s1a1.enabled='1'
# uci set quota_limit.mob1s1a1.ifname='mob1s1a1'
# uci set quota_limit.mob1s1a1.reset_hour='10'
# uci set quota_limit.mob1s1a1.sim='1'
# uci set quota_limit.mob1s1a1.data_limit='10000'
# uci set quota_limit.mob1s1a1.enable_warning='1'
# uci set quota_limit.mob1s1a1.period='1'
# uci set quota_limit.mob1s1a1.warning_limit='8000'
# uci set quota_limit.mob1s1a1.warning_num='+37012345678'
```

Commit changes and restart the daemon

```
# uci commit quota_limit
# /etc/init.d/quota_limit restart
```

Disabling / Deleting configuration

Let us take the above example and disable the data limit using **uci**. For this, we simply need to change the "enabled" value to '0':

```
# uci set quota_limit.mob1s1a1.enabled='0'
# uci commit quota_limit
# /etc/init.d/quota_limit restart
```

If you wish to delete the whole configuration, **uci delete** can be used. Let us delete the entire block of configuration of mob1s1a1 interface:

```
# uci delete quota_limit.mob1s1a1
# uci commit quota_limit
# /etc/init.d/quota_limit restart
```

External links

- https://wiki.openwrt.org/doc/uci?do= - OpenWRT wiki page on the UCI system
- https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html - PuTTY downloads page