



## UCI (统一配置接口) – 技术参考资料


- UCI(统一配置接口)应用程序和库文件的项目管理页面 [http://nbd.name/gitweb.cgi?p=uci.git;a=summary]
- UCI适用于OpenWrtR10367 (trunk) [https://dev.openwrt.org/changeset/10367]之后的版本
- 在本地GNU/Linux发行版上使用git命令可获取源代码:

```
git clone git://nbd.name/uci.git
```

- 本文为UCI技术参考，UCI详细语法请参看 [UCI\(统一配置接口\) – 使用手册](#)

### UCI是什么？

UCI 是一个用C [https://en.wikipedia.org/wiki/C\_(programming\_language)]语言编写的小应用程序(也可用作一个shell 脚本 [https://en.wikipedia.org/wiki/shell 脚本]封装器), 其开发目的是集中管理一个OpenWrt设备内的所有配置选项. UCI是对 OpenWrt历史版本White Russian上 NVRAM基础配置管理系统 的一个继承, 是一个对众多应用程序自带的标准配置文件的包装, 比如像: `/etc/network/interfaces`, `/etc/exports`, `/etc/dnsmasq.conf`, `/etc/samba/samba.conf` 等.



UCI配置文件位于目录 `/etc/config`内, 其在线访问文档在 [OpenWrt-Wiki 之 UCI 配置文档](#).

你可以用任意文本编辑器或命令行应用程序uci来修改它们,也可通过各种编程 API（如 Shell, Lua和C）来修改它们。比如本WUILuCI – 技术参考就是使用Lua来管理它们。

### UCI的依赖项

- libuci 一个用C [https://en.wikipedia.org/wiki/C\_(programming\_language)]语言写的UCI的小程序库
- libuci-lua 一个Lua [https://en.wikipedia.org/wiki/Lua\_(programming\_language)]语言的插件, 可被像LuCI的程序使用.

Both are maintained in the same git as UCI.

### 程序包

该功能主要由两个程序包uci和libuci提供. 程序包libuci-lua也能用到.

名称	大小(Byte)	说明
uci	7196	统一配置接口 (UCI)应用程序
libuci	18765	统一配置接口 (UCI)的C语言库
libuci-lua	~6000	libuci的 Lua [https://en.wikipedia.org/wiki/Lua_(programming_language)]语言插件, 像LuCI用到它

### 安装文件

#### uci

安装目录	文件类型	说明
<code>/sbin/uci</code>	binary	uci的可执行文件
<code>/lib/config/uci.sh</code>	shell script	<code>/sbin/uci</code> 的兼容Shell脚本封装

#### libuci

安装目录	文件类型	说明
<code>/lib/libuci.so</code>	symlink	libuci.so.xxx的符号链接
<code>/lib/libuci.so.2011-01-19</code>	binary	库文件

#### libuci-lua

安装目录	文件类型	说明
<code>/usr/lib/lua/uci.so</code>	binary	库文件

## Lua绑定UCI

对于那些喜欢lua的人,可以在代码中通过软件libuci-lua访问操控UCI. 只要安装了这个软件,然后在lua代码中运行

```
require("uci")
```

## API

该 api 相当简单

顶层入口

uci.cursor() (实体化一个 uci 上下文实例) 例如

```
x = uci.cursor()
```

或

```
x = uci.cursor(nil, "/var/state")
```

如果你想包括状态值

这里你可以调用常用操作

```
x:get("config", "sectionname", "option")
```

返回字符串 或 nil(没找到数据时)

```
x:set("config", "sectionname", "option", "value")
```

设置简单的字符串变量

```
x:set("config", "sectionname", "option", { "foo", "bar" })
```

设置列表变量

```
x:delete("config", "section", "option")
```

删除选项

```
x:delete("config", "section")
```

删除段

```
x:add("config", "type")
```

adds new anon section "type" and returns its name

```
x:set("config", "name", "type")
```

adds new section "name" with type "type"

```
x:foreach("config", "type", function(s) ... end)
```

遍历所有类型为的"type"段,并以每个"s"为参数调用回调函数. s 是一个包含所有选型和两个特有属性的列表

- s['.type'] → 段类型
- s['.name'] → 段名称

如果回调函数返回 **false** [NB: not nil!], foreach() 在这个点会终止,不再继续遍历任何剩余的段. 如果至少存在一个段且回调函数没有产生错误,foreach() 会返回 true; 否则返回 false.

```
x:reorder("config", "sectionname", position)
```

Move a section to another position. Position starts at 0. This is for example handy to change the wireless config order (changing priority).

```
x:revert("config")
```

discards any changes made to the configuration, that have not yet been committed

```
x:commit("config")
```

commits (saves) the changed configuration to the corresponding file in /etc/config

That's basically all you need

## UCI构架相关

花一点时间来理解 "section" 和 "type" 之间的差异是值得的. 让我们从一个例程开始:

```
#uci show system
system.@system[0]=system
system.@system[0].hostname=OpenWrt
system.@system[0].timezone=UTC
system.@rdate[0]=rdate
system.@rdate[0].server=ac-ntp0.net.cmu.edu ptbtime1.ptb.de ac-ntp1.net.cmu.edu ntp.xs4all.nl ptbtime2.ptb.de cudns.cit.cornell.edu ptbtime3.ptb.de
```

这里, x:get("system","@rdate[0]","server") 将不正常工作. 因为 rdate 是个 type, 不是 section.

执行 x:get\_all("system")的返回结果如下:

```
{
  cfg02f02f={[".name"]="cfg02f02f",[".type"]="system",hostname="OpenWrt",[".index"]=0,[".anonymous"]=true,timezone="UTC"},
  cfg04e10c={[".name"]="cfg04e10c",[".type"]="rdate",[".index"]=1,[".anonymous"]=true,
    server={"ac-ntp0.net.cmu.edu","ptbtime1.ptb.de","ac-ntp1.net.cmu.edu","ntp.xs4all.nl","ptbtime2.ptb.de","cudns.cit.cornell.edu","ptbtime3.ptb.de"}}
}
```

[".type"] 列出了段的类型 [".name"] 列出了段实际名称。这里可以看到，这些名称是系统生成的。[".index"] 是列表的索引(+1)

据我所知，好像还没有办法直接访问"@rdate[0]"。你必须用x:foreach 迭代列出所给定类型的所有元素。我一般用下面的函数：

```
uci=require("uci")
function getConfigType(conf,type)
  local curs=uci.cursor()
  local ifce={}
  curs:foreach(conf,type,function(s) ifce[s[".index"]]=s end)
  return ifce
end
```

getConfigType("system","rdate") 返回：

```
{{[".name"]="cfg04e10c",[".type"]="rdate",[".index"]=1,[".anonymous"]=true,
server={"ac-ntp0.net.cmu.edu","ptbtime1.ptb.de","ac-ntp1.net.cmu.edu","ntp.xs4all.nl","ptbtime2.ptb.de","cudns.cit.cornell.edu","ptbtime3.ptb.de"}}}
```

所以，如果你想修改system.@rdate[0].server,你需要迭代所需类型的段名(!".name")),然后调用x:set("system","cfg04e10c","server","zzz.com") 希望这对你有帮助  
Sophana (不过, Luci有一个Cursor:get\_first [[http://luci.subsignal.org/api/luci/modules/luci.model.uci.html#Cursor.get\\_first](http://luci.subsignal.org/api/luci/modules/luci.model.uci.html#Cursor.get_first)] 函数实现了类似的功能, 不同的是他类型而不是段作为第二个参数.)

## 附加信息

另请参考 [LuCI UCI model functions \[http://luci.subsignal.org/api/luci/modules/luci.model.uci.html\]](http://luci.subsignal.org/api/luci/modules/luci.model.uci.html). 那是LuCI对CUI的用法。它用一些更简便的函数扩展了UCI指针类。

---

## OpenWrt之外的用法

如果你想在OpenWrt之外使用libuci(例如:您正在计算机上用C语言开发一个应用程序)，然后准备如下：

获取源代码：

```
git clone git://nbd.name/uci.git
```

转至源码目录（CMakeLists.txt所在的目录），并配置 build 为无Lua绑定：

```
cd uci/; cmake -D BUILD_LUA=BOOL=OFF .
```

编译并安装uci ALS根（这将吧uci安装到/usr/local/，请参阅如下主题,怎样在你的home目录下安装无root权限使用UCI: <https://forum.openwrt.org/viewtopic.php?id=40547> [<https://forum.openwrt.org/viewtopic.php?id=40547>]）：

```
make install
```

打开 /etc/ld.so.conf, 并把你安装的uci库添加到这里：

```
vi /etc/ld.so.conf
```

在 /etc/ld.so.conf 某处添加这一行

```
/usr/local/lib
```

以root执行 ldconfig 来执行 /etc/ld.so.conf 的变化

```
ldconfig
```

To compile your application you have to link it against the uci library. Append -luci in your Makefile: 编译您的应用程序，你必须在您的Makefile文件中附加-luci以链接uci库：

```
$(CC) test.o -o test -luci
```

并且可以在这个页面找到如何在C中使用UCI的例程: <https://forum.openwrt.org/viewtopic.php?pid=183335#p183335> [<https://forum.openwrt.org/viewtopic.php?pid=183335#p183335>] 更多的例程查你从git上克隆的开源的UCI源码目录，打开cli.c 或 ucimap-example.c文件

## 运行

所有的 uci set, uci add, uci rename 和 uci delete 命令 都是在/tmp中实现, 并用uci commit命令立即写入flash. 这明显不允许用户使用文本编辑器,而是使用应用uci的脚本,图形界面和其他应用程序.

wip