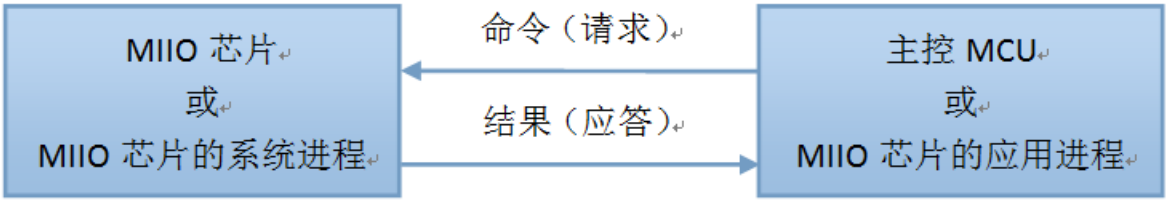
概述

MIIO 芯片提供一套可读的串口文本命令，供外部芯片调用，一般适用于 MIIO 模块不是主控芯片的情况，即 MIIO 芯片只负责网络通讯，而不关心业务逻辑。

文本命令采用“一问一答”的方式，每条命令，无论执行成功与否，都会返回结果或错误提示。 所有文本命令，如无特殊说明，都使用小写字母。 下文中，示例里的“↑”和“↓”分别代表命令和结果。有时，也称二者为“上行”信息和“下行”信息。因为在涉及到云通信的命令里，“上”代表发往云端，“下”代表从云端发来。



一个典型的上行例子（温度传感器上报温度）：

1. 传感器探知当前温度为 26 度
2. 传感器调用 MIIO 芯片的 props 命令，上报温度属性：props temp 26

一个典型的下行例子（手机 app 打开电源开关）：

1. 手机 app 发出打开开关指令
2. MIIO 芯片接收到 app 的下行消息，并翻译为文本命令：power_on
3. 主控 MCU 不停的使用 get_down 命令轮询 MIIO 芯片，得到了 power_on 命令
4. 主控 MCU 完成电源打开的操作
5. 主控 MCU 调用 result ”ok“，告知 MIIO 芯片成功完成
6. MIIO 芯片回复 ok，告知结果已收到

芯片接口命令

串口层默认配置为 115200,8,1，N,N

props

- 参数：<prop_name_1> <value_1> <prop_name_2> <value_2> ...
- 返回：ok 或 error
- 说明：参数至少要有一个名值对。
- 举例：

↑props temp 21 hum 50 location "home"
↓ok
↑props location office (格式非法)
↓error

- 限制：属性名的字符个数不得多于 31 个（可等于 31）

event

- 参数：<event_name> <value_1> <value_2>...
- 返回：ok 或 error
- 说明：发送上行事件。
- 举例：

↑event overheat 170
↓ok
↑event button_pressed
↓ok
↑event fault "motor stuck" "sensor error"
↓ok
↑event fault motor stuck (格式非法)
↓error
↑event fault "motor stuck"
↓ok ... (短时间内大量 event）
↑event fault "motor stuck" （此时，事件队列已满）
↓error

- 限制：

1. 事件名的字符个数不得多于 31 个（可等于 31）

2. 如果 MIIO 芯片断网，事件会在芯片里缓存最长 10 分钟，如果这段时间内网络未恢复正常，则事件会被丢弃。
3. MIIO 芯片最多缓存 8 条事件，如果短时间内大量调用 event 命令，则会造成缓存列表填满，后续事件会被丢弃

get_down

- 参数：无
- 返回：down <method_name> <arg1>,<arg2>,<arg3> ...
- 说明：获取下行指令。如果有下行方法，则返回该方法，如没有，则返回 none。如果对上一个方法还没有给出结果，则返回 error。MCU 在获得下行方法后，有 1s 时间处理，并用 result/error 命令上传结果。超过 1s 后，则 wifi 模块认为调用失败。
- 返回值说明：如果没有下行命令，则返回 down none。如果有下行命令，则返回命令名、命令参数（如果有参数）。命令名和参数之间用空格隔开，多个参数之间用逗号隔开。参数可以是双引号括起的字符串，或是数字。

- 举例：

↑get_down
↓down none
↑get_down
↓down power_on "light_on",60,0,1
↑result "ok"
↓ok
↑get_down
↓down power_off
↑get_down
↓error

result

- 参数：<value1> <value2> ...
- 返回：成功返回 ok，失败返回 error
- 说明：发送下行指令的执行结果，如果有多个返回值，则以逗号分开。发送成功后，返回 ok。
- 举例：

↑result 123 "abc"
↓ok

error

- 参数：<message> <error_code>
- 返回：成功返回 ok，失败返回 error
- 说明：如果下行指令执行出错，可用该命令发送错误信息。发送成功后，返回 MIIO 芯片回复 ok。message 必须是双引号括起的字符串，错误码必须是-9999 到 -5000 之间的整数（含边界值）。
- 举例：

↑error "memory error" -5003
↓ok
↑error "stuck" -5001
↓ok

json_get_down

- 参数：无
- 返回：down <json_string>
- 说明：获取 Json 格式的下行指令。如果有下行方法，则返回该方法，如没有，则返回 none。如果对上一个方法还没有给出结果，则返回 error。MCU 在获得下行方法后，有 1s 时间处理，并用 result/error 命令上传结果。超过 1s 后，则 wifi 模块认为调用失败。
- 返回值说明：如果没有下行命令，则返回 down none。

- 举例：

↑json_get_down
↓down none
↑json_get_down
↓down{“method":"power_on","params":["light_on",60,0,1],"id":123456}
↑json_ack {"result":"ok"}
↓ok

json_ack

- 参数：<json_string>
- 返回：成功返回 ok，失败返回 error
- 说明：json_ack 与 json_get_down 相对应，用来回复下行指令执行的结果。
- 举例：

↑json_ack {"id":1,"error":{"code":-3,"message":"device timeout"}}
↓ok
↑json_ack {"id":1,"result":1}
↓ok

json_send

- 参数：<json_string>
- 返回：ok
- 说明：用来以 json 格式发出上行消息。
- 举例：

↑json_send{"method":"props","params":{"prop1":100,"prop2":"string"},"id":123}
↓ok

log

- 参数：<log_name> <arg1> <arg2> <arg3> ...
- 返回：ok
- 说明：用于上报以记录为目的的数据
- 举例：

↑log pure_water_record "2015-5-14 19:27:31" 210 33 500
↓ok

call

- 参数：<method> <params>
- 返回：ok
- 说明：用于调用云端的方法，其结果以下行的 return 命令被 get_down 获取

- 举例：

↑call get_start_time ↓ok
↑get_down
↓down none ...(一段时间后）
↑get_down
↓down result“1434446397” （成功） 或
↓down error“xxxxxxxxx” （失败）

version

- 参数：无
- 返回：版本号，当前为 2
- 说明：串口协议（本协议）的版本
- 举例：

↑version
↓2

net

- 参数：无
- 返回： offline 或 local 或 cloud 或 updating 或 uap 或 unprov
- 说明：询问网络状态。返回值分别代表：连接中（或掉线）、连上路由器但未连上服务器、连上小米云服务器、固件升级中、uap 模式等待连接、关闭 wifi（一小时未快连）。
- 举例：

↑net
↓offline
↑net
↓local
↑net
↓cloud

time

- 参数：无 或 posix
- 返回：当前日期和时间，格式见举例。时间为 UTC+8 时间，即中国的时间。
- 举例：

↑time
↓2015-06-04 16:58:07
↑time posix
↓1434446397

model

- 参数：无 或 <model_name_string>

- 返回：无参数时，返回当前 model 字符串。带参数时，将芯片 model 设为参数字符串，并返回 ok。如果参数字符串不合法，则返回 error。
- 说明：合法的 model 字符串有 3 部分构成：公司名、产品类别名、产品版本。3 部分以“.”连接，总长度不能超过 23 个字符。
- 举例：

```
↑model
↓xiaomi.dev.v1
↑model xiaomi.prod.v2
↓ok
↑model
↓xiaomi.prod.v2
↑model company.product_name_that_is_too_long.v2
↓error
↑model plug
↓error
```

ota_state

- 参数：无 或 <ota_state_name_string>
- 返回：无参数时，返回当前 ota_state 字符串。带参数时，将芯片 ota_state 设为参数字符串，并返回 ok。如果参数字符串不合法，则返回 error。
- 说明：合法的 model 字符串有 idle，表示 MCU 目前允许升级；busy，表示 MCU 不允许升级
- 举例：

```
↑ota_state
↓idle
↑ota_state busy
↓ok
↑ota_state
↓busy
↑ota_state downloading
↓error
```

注：如果当前 wifi 模组正在升级，即 ota_state 不是 idle 或 busy，该命令会返回 ok，但是当前设置其实无效。主要不能让 MCU 的设置打乱 wifi 模组的升级状态。

mcu_version

- 参数：<mcu_version_string>

- 返回：如果参数合法，则返回 ok，非法则返回 error
- 说明：上报 MCU 固件版本。要求必须是 4 位数字。MCU 应该在应用固件引导后，第一时间调用该命令上报版本。另外，如果下行命令收到 MIIO_mcu_version_req，也应该立即上报版本。
- 举例：

↑mcu_version 0001
↓ok
↑mcu_version A001
↓error
↑mcu_version 1
↓error

reboot

- 参数：无
- 返回：ok
- 说明：MIIO 接收到该命令后，将在 0.5 秒内重启。
- 举例：

↑reboot
↓ok

restore

- 参数：无
- 返回：ok
- 说明：MIIO 接收到该命令后，将清除 wifi 配置信息，并在 0.5 秒内重启。
- 举例：

↑restore
↓ok

factory

- 参数：无
- 返回：ok
- 说明：MIIO 芯片收到该命令后，在 0.5 秒内重启，并进入厂测模式。该模式下，芯片会按照预设置信息连接路由器。预设路由器 SSID：miio_default，密码：0x82562647
- 举例：

↑factory
↓ok

get_pin_level

- 参数：无 或 <mask_1> <mask_2> <mask_3>
- 返回：<hex_1> <hex_2> <hex_3>
- 说明：获取管脚电平，返回的 3 个 32 位 16 进制数的每一位对应 MC200 的一个 GPIO 的电平状态，0 代表低电平，1 代表高电平。hex_1 的低位到高位分别对应 GPIO_0 到 GPIO_31，hex_2 对应 32～63，hex_3 对应 64～79。参数为 GPIO mask。例如，只想获得 GPIO-2 的电平，则参数为 0x4 0x0 0x0
- 举例：

↑get_pin_level 0x4 0x0 0x0
↓0x00000004 0x00000000 0x00000000

echo

- 参数：on 或 off
- 返回：ok
- 说明：打开串口回显功能。该功能开启时，MIIO 芯片的串口输出会对输入进行回显。这在使用串口工具（比如 SecureCRT 或 Xshell）时，提供方便。

help

- 参数：无
- 返回：所有支持的命令和参数格式

dl

- 参数：uart url
- 返回：ok/error
- 说明：uart 为透传数据接口，目前版本固定为 uart。 url 为需要下载的 url。
- 当使用 uart 模式时，数据传输协议为 xmodem，波特率 230400,传输速率约 10 KB/s

set_low_power

- 参数："on"/"off"
- 返回：ok/error
- 说明：打开模组的低功耗功能。目前只有乐鑫 Esp8266 上有实现。

保留的下行命令

MIIO 芯片保留了一些命令名，用来通知主控 MCU 特定的事件。这类命令同样可用 get_down 得到，但无需 result 或 error 回复结果。

MIIO_mcu_version_req

- 表示 MIIO 芯片向 MCU 询问 MCU 固件版本号。MCU 收到该命令后，应调用 mcu_version 命令，告知自己的版本号。
- 举例：

↑get_down
↓down MIIO_mcu_version_req
↑mcu_version 1234
↓ok

MIIO_model_req

- 表示 MIIO 芯片向 MCU 询问设备种类名（即 model）。MCU 收到该命令后，应该调用 model 命令报告种类名。
- 举例：

↑get_down
↓down MIIO_model_req
↑model xiaomi.demo.v1
↓ok

MIIO_net_change

- 表示 MIIO 芯片的网络连接状态发生了变化。其后的参数代表了芯片最新的网络状态。网络状态请参考 net 命令。
- 举例：

↑get_down
↓down MIIO_net_change cloud （改变 LED 灯的状态等等...）
[建议不要使用该命令，MIIO_net_change 是单次失效的命令，如果 MCU 获取到了 net_change，但又不处理，等到它真正想获取的时候却又获取不到]