

MiCOKit 云数据通信协议

摘要 (Abstract)

本文档主要介绍 MiCOKit 云数据通信协议，即 APP 如何通过 FogCloud 与 MiCO 设备进行数据交互，方便 MiCO-IoT 物联网开发者使用 MiCOKit 开发板及 FogCloud 云进行物联网应用开发。

适用对象 (Status of This Document)

适用于所有 MiCO-IoT 物联网智能硬件开发者学习参考。

获取更多帮助 (More Help)

MiCO 开发团队向您推荐：MiCO 开发者学习网站：<http://mico.io/>（开发者中心），获取更多最新资讯。

手机微信“扫一扫”关注：“MiCO 总动员”公众号，获取 MiCO 团队小伙伴最新活动信息。



登录上海庆科官方网站：<http://mxchip.com/>，获取公司最新产品信息。

版权声明 (Copyright Notice)

Copyright (c) 2015 MWG Trust and the persons identified as the document authors. All rights reserved.

目 录

MiCOKit 云数据通信协议	1
1. 概述	3
2. 设备描述	4
2.1. 设备抽象	4
2.2. 设备描述表	4
3. 数据流	6
3.1. 设备数据通道	6
3.2. 设备访问流程	6
4. 异常处理	8
4.1. 状态码	8
4.2. 异常消息	8
5. UUID	10
5.1. UUID 定义规则	10
5.2. 实例	10
6. 内部 ID (iid)	12
6.1. iid 定义规则	12
6.2. 实例	12
7. 消息体数据格式	13
7.1. JSON 的一层 Key-Value 结构 :	13
7.2. APP 读取属性值请求	13
7.3. APP 写属性值请求 :	14
8. 实例	15
8.1. 设备功能模块	15
8.2. APP 消息通信	18
9. 修订记录	23
List of Figures	
Figure 1. APP 与设备消息流示意图	3
Figure 2. 设备描述表结构	4
Figure 3. 设备描述表	15
Figure 4. APP 与设备间数据交互	21
Figure 5. 设备串口输出数据	21
Figure 6. 设备串口向云端发送数据	22
Figure 7. 云端接收设备串口数据	22

List of Tables

Table 1. 设备数据通道.....	6
Table 2. 获取设备描述表	6
Table 3. 读取设备属性.....	6
Table 4. 写入设备属性.....	7
Table 5. 读取设备服务.....	7
Table 6. 属性通知.....	7
Table 7. 状态码	8
Table 8. 模块和属性的 UUID 定义	10
Table 9. 内部 iid 实例	12

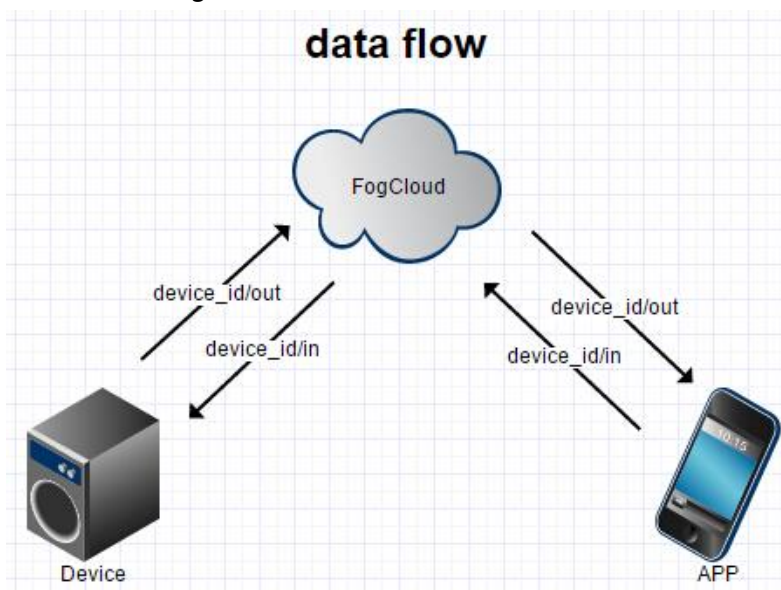
1. 概述

本文档主要描述 APP 如何通过 FogCloud 与 MiCO 设备进行数据交互的协议；

APP 和设备之间的消息收发采用 MQTT 协议；

APP 开发者根据此协议完成 APP 对已经连接上 FogCloud 的 MiCO 设备的远程读写操作，从而完成对设备的远程控制操作。

Figure 1. APP 与设备消息流示意图



2. 设备描述

2.1. 设备抽象

本协议将 MiCO 设备所具有的功能模块（如开关、LED、串口等外设）抽象成可访问的服务（service）；将每个模块所具有的功能（开关的状态、LED 灯的亮度值等）抽象成可读写的属性（property）。每一个 service/property 都分配一个固定的 iid，作为访问标识。

2.2. 设备描述表

将每个 MiCO 设备所具有的全部功能模块（services/properties）使用一个设备描述表（service_table）来表示。APP 从设备获取到该描述表之后，就可以描绘出整个设备所具有的功能模块，展示给用户。

设备描述表采用 JSON 格式，使用一个 services 对象数组表示设备的模块列表；每一个 service 对象中使用一个 properties 对象数组表示该模块所具有的所有属性；每一个属性中使用不同的字段表示该属性的特征。

设备描述表结构如下：

Figure 2. 设备描述表结构

```
{
  "services": [
    {
      "type": "UUID",                // service1 UUID
      "iid": <integer>,             // service iid
      "properties": [
        { // property 1
          "type": "UUID",           // property UUID
          "iid": <integer>,         // property iid
          "value": <value>,         // property current value
          "format": "data_type",    // property value type
          "perms": [               // property permission
            "pr",                  // can read
            "pw",                  // can write
            "ev"                   // will notify
          ],
          "maxValue": <value>,      // max value for int/float
          "minValue": <value>,      // min value for int/float
          "minStep": <value>,       // min step value for int/float
          "maxStringLen": <integer> // max string length in byte
          "unit": "unit string"     // property data unit
        },
        { // property 2
          ...
        }
      ]
    },
    { // service2
      ...
    },
    ...
  ]
}
```

其中:

- 蓝色标识的属性字段为可选字段;
- UUID 为 service 或者 property 的类型, APP 根据此类型给用户提供相应的功能;
- iid 为该设备上所有 services 和 properties 的编号, 是一个正整数;
- UUID 和 iid 的详细说明见后续 “UUID” 和 “内部 ID” 部分。

3. 数据流

APP 和 MiCO 设备之间通过 FogCloud 云的不同的数据通道进行数据交互。

3.1. 设备数据通道

Table 1. 设备数据通道

类型	通道	消息流向
读取设备	<device_id>/in/read/<session_id>	APP ==> dev
写入设备	<device_id>/in/write/<session_id>	APP ==> dev
读取响应	<device_id>/out/read/<session_id>	Dev ==> APP
写入响应	<device_id>/out/write/<session_id>	Dev ==> APP
设备异常消息	<device_id>/out/err	设备异常状态输出到该通道

其中，<session_id> 表示请求的来源（由 APP 决定），设备根据此 session_id 回复请求方，不设置则表示广播该消息。

3.2. 设备访问流程

（1）获取设备描述表

Table 2. 获取设备描述表

类型	发送端	消息通道	消息数据
获取设备描述表	APP	<device_id>/in/read/<session_id>	{}
设备响应	device	<device_id>/out/read/<session_id>	JSON 格式设备描述表
异常消息	device	<device_id>/out/err	成功：{ "status" :0} 失败：见“异常处理”部分

（2）读取设备属性

Table 3. 读取设备属性

类型	发送端	消息通道	消息数据
读取设备属性	APP	<device_id>/in/read/<session_id>	{ "iid1" : <no use>, "iid2" :<no use>, ...}
设备响应	device	<device_id>/out/read/<session_id>	{ "iid1" : value1, "iid2" :value2}
异常消息	device	<device_id>/out/err	成功：{ "status" :0} 失败：见“异常处理”部分

(3) 写入设备属性

Table 4. 写入设备属性

类型	发送端	消息通道	消息数据
写入设备属性	APP	<device_id>/in/write/<session_id>	{ "iid1" : value1, "iid2" :value2, ...}
设备响应	device	<device_id>/out/write/<session_id>	{ "iid1" :value1, "iid2" :value2, ...}
异常消息	device	<device_id>/out/err	成功 : { "status" :0} 失败 : 见 "异常处理" 部分

(4) 读取设备服务 (读取 iid 服务下的所有属性)

Table 5. 读取设备服务

类型	发送端	消息通道	消息数据
读取设备服务	APP	<device_id>/in/read/<session_id>	{ "iid1" : <no use>}
设备响应	device	<device_id>/out/read/<session_id> >	{ "iid2" :value2, "iid3" :value3, ...}
异常消息	device	<device_id>/out/err	成功 : { "status" :0} 失败 : 见 "异常处理" 部分

(5) 属性通知 (设备自动上报)

Table 6. 属性通知

类型	发送端	消息通道	消息数据
属性通知	device	<device_id>/out/read	{ "iid2" : value2, "iid3" :value3, ...}
异常消息	device	<device_id>/out/err	成功 : { "status" :0} 失败 : 见 "异常处理" 部分

4. 异常处理

设备响应状态输出到<device_id>/out/err 消息通道，APP 可从该通道中获取命令执行状态。

4.1. 状态码

Table 7. 状态码

Value	Description
0	操作成功
-70101	读取失败
-70102	写入失败
-70103	部分读取失败
-70104	部分写入失败
-70401	属性不可读
-70402	属性不可写
-70403	服务/属性不存在
-70404	Get function 未设置
-70405	Set function 未设置
-70406	Notify check function 未设置
-70501	数据格式错误
-70502	不支持的操作

4.2. 异常消息

异常消息体数据格式如下：

执行成功：

```
{  
    "status" : 0  
}
```

执行异常：

```
{  
    "status" : <err_code>,  
    "properties" : {  
        "iid1" : <err_code>,  
    }  
}
```

```
"iid2" : <err_code>,  
  
"iid3" : <err_code>,  
  
...  
  
}
```

其中：

properties 对象中表示执行异常的 properties 及错误码。

5. UUID

5.1. UUID 定义规则

使用 UUID (Universally Unique Identifier) 来表示设备上不同 services 和 properties 的类型，APP 根据这些事先定义好的类型向用户展示相应的设备功能。

注意：目前 UUID 码暂时未定，暂时仍使用唯一的字符串表示。格式如下：

`<public/private>.map.<service/property>.<module>`

其中：

(1) public 表示公开定义好的 service 或者 property，如：

`public.map.service.dev_info` 为每个设备必须有的第一个 service；

`public.map.property.name` 表示一个模块的名字的字符串。

(2) private 表示某一类型的设备所特定的 service 或者 property，如：

`private.map.service.xxx` 表示该设备有一个特定类型的模块 xxx；

`private.map.property.yyy` 表示该设备某个模块的一个特定的属性 yyy。

(3) "map" 为 MicoKit Accessary Protocol 的缩写。

(4) `<service/property>` 表示是一个模块还是一个模块的属性。

(5) `<module>` 表示具体的类型，如 adc, rgb_led, button 等。

5.2. 实例

模块和属性的 UUID 定义[暂时用字符串]

Table 8. 模块和属性的 UUID 定义

Service/Property Type	Description	Category
"public.map.service.base_info"	设备描述表	System
"public.map.service.dev_info"	设备基本信息	
"public.map.property.name"	名称	
"public.map.property.manufacturer"	制造商	
"public.map.property.serial_number"	序列号	
"public.map.property.hd_version"	硬件版本号	
"public.map.property.fw_version"	固件版本号	
"public.map.property.mac"	MAC 地址	

Service/Property Type	Description	Category
"public.map.property.ip"	IP 地址	Modules
...	<其他设备属性>	
"public.map.service.rgb_led"	RGB LED	
"public.map.service.adc"	ADC	
"public.map.service.uart"	串口模块	
"public.map.service.light_sensor"	光线传感器	
"public.map.service.infrared"	红外线反射传感器	
"public.map.service.temperatur"	温度传感器	
"public.map.service.humidity"	湿度传感器	
"public.map.service.motor"	直流震动电机	
"public.map.service.ht_sensor"	温湿度传感器	
"public.map.service.proximity_sensor"	距离传感器	
"public.map.service.atmosphere_sensor"	大气压传感器	
"public.map.service.montion_sensor"	三轴加速度传感器	
...	<公共模块>	
"private.map.service.<xxxx>"	<自定义模块>	
"public.map.property.value"	数值类型属性	
"public.map.property.brightness"	亮度属性	
"public.map.property.saturation"	饱和度属性	
"public.map.property.hues"	色相属性	
...	<公共属性>	
"private.map.property.<yyyy>"	<自定义属性>	
"public.map.service.common"	<通用模块>	APP 自动在 UI 上显示一个通用模块 (支持 3 个属性值)

6. 内部 ID (iid)

6.1. iid 定义规则

- (1) 每一个 service 和 property 都会分配一个固定的 iid , 并且在一个设备上唯一 ;
- (2) 设备描述表的 iid=0 , 设备的基本信息的 iid=1 , 后续 services/properties 按顺序分配 ;
- (3) APP 可以一次使用多个 iid 读取/设置多个不同的 property ;
- (4) APP 可以一次使用多个 iid 读取多个 service 下的所有 properties ;
- (5) 设置某个 property 的值必须指定该 property 的 iid ;
- (6) iid 由固件程序按照设备描述表中所列出的所有 services 和 properties 的顺序自动分配。

6.2. 实例

Table 9. 内部 iid 实例

Services	Properties	iid	Type(UUID)	description
Device description	-	-	-	-
Device base information	-	1	"public.map.property.dev_info"	Service1: 设备基本信息 (固定保留)
-	"name"	2	"public.map.property.name"	设备名称
-	manufacturer	3	"public.map.property.manufacturer"	设备制造商
RGB LED	-	4	"public.map.service.rgb_led"	Service2: RGB LED
-	hues	5	"public.map.property.hues"	LED 颜色的 hues 分量
-	saturation	6	"public.map.property.saturation"	LED 颜色的 saturability 分量
-	brightness	7	"public.map.property.brightness"	LED 颜色的 brightness 分量
-	switch	8	"public.map.property.switch"	LED 的开关量
ADC	-	9	"public.map.service.adc"	Service3: ADC 模块
-	adc value	10	"public.map.property.value"	ADC 的采样值

7. 消息体数据格式

APP 和设备之间消息体的数据格式采用 JSON 格式 ,每个 property(或 service)使用一个 key-value 对表示。key 为请求或者返回的 service/property 的 iid , value 为相应的属性值。

7.1. JSON 的一层 Key-Value 结构 :

```
{  
  
    "iid1" : <value1>,  
  
    "iid2" : <value3>,  
  
    "iid3" : <value3>,  
  
    ...  
  
}
```

7.2. APP 读取属性值请求

其中 , k-v 的 k 值为请求的 iid 的字符串 :

```
{  
  
    "iid1" : <no use>,  
  
    "iid2" : <no use>,  
  
    "iid3" : <no use>  
  
}
```

- 设备读取成功响应数据 :

```
{  
  
    "iid1" : value1,  
  
    "iid2" : value2,  
  
    "iid3" : value3  
  
}
```

7.3. APP 写属性值请求：

```
{  
  
    "iid1" : value1,  
  
    "iid2" : value2,  
  
    "iid3" : value3  
  
}
```

- 设备写入成功响应数据：

```
{  
  
    "iid1" : value1,  
  
    "iid2" : value2,  
  
    "iid3" : value3  
  
}
```

其中：

- (1) value 为读取或写入的属性值；
- (2) 返回状态码见“异常处理”部分的说明。

8. 实例

8.1. 设备功能模块

1. 设备具有三个模块：rgb_led, adc, uart。其中 ADC 自动采集数并上报，uart 和云端实现数据透传。

设备描述表如下：

Figure 3. 设备描述表

```
{
  "services": [
    {
      "type": "public.map.service.dev_info",
      "iid": 1,
      "properties": [
        {
          "type": "public.map.property.name",
          "iid": 2,
          "value": "MiCOKit-3288",
          "format": "string",
          "perms": [
            "pr",
            "pw"
          ],
          "maxStringLen": 16
        },
        {
          "type": "public.map.property.manufacturer",
          "iid": 3,
          "value": "MXCHIP",
          "format": "string",
          "perms": [
            "pr"
          ],
          "maxStringLen": 16
        }
      ]
    }
  ],
}
```



```
{
  "type": "public.map.service.rgb_led",
  "iid": 4,
  "properties": [
    {
      "type": "public.map.property.switch",
      "iid": 5,
      "value": false,
      "format": "bool",
      "perms": [
        "pr",
        "pw"
      ]
    },
    {
      "type": "public.map.property.hues",
      "iid": 6,
      "value": 120,
      "format": "int",
      "perms": [
        "pr",
        "pw"
      ],
      "maxValue": 360,
      "minValue": 0,
      "minStep": 1,
      "unit": "degree"
    },
    {
      "type": "public.map.property.saturation",
      "iid": 7,
      "value": 100,
      "format": "int",
      "perms": [
        "pr",
        "pw"
      ],
      "maxValue": 100,
      "minValue": 0,
      "minStep": 1,
      "unit": "percentage"
    }
  ],
}
```

```
{
  "type": "public.map.property.brightness",
  "iid": 8,
  "value": 100,
  "format": "int",
  "perms": [
    "pr", "pw"
  ],
  "maxValue": 100,
  "minValue": 0,
  "minStep": 1,
  "unit": "percentage"
}
],
},

{
  "type": "public.map.service.adc",
  "iid": 9,
  "properties": [
    {
      "type": "public.map.property.value",
      "iid": 10,
      "value": 2088,
      "format": "int",
      "perms": [
        "pr", "ev"
      ],
      "maxValue": 4095,
      "minValue": 0,
      "minStep": 1
    },
  ],
},
],
},
```

```
{
  "type": "public.map.service.uart",
  "iid": 11,
  "properties": [
    {
      "type": "public.map.property.message",
      "iid": 12,
      "value": "",
      "format": "string",
      "perms": [
        "pr",
        "pw",
        "ev"
      ],
      "maxStringLen": 32,
      "unit": "byte"
    }
  ]
}
```

8.2. APP 消息通信

(APP 使用 MQTT 调试工具代替 : <http://api.easylink.io/tools/mqtt/>)

(1) APP 请求设备描述表 :

发送通道 : <device_id>/in/read/app123

发送数据 : {}

设备响应 :

数据通道 : <device_id>/out/read/app123

返回数据 : 设备描述表

状态通道：<device_id>/out/err

状态数据：{ "status": 0 }

(2) APP 读取设备基本信息：

发送通道：<device_id>/in/read/app123

发送数据：{"1":0}

设备响应：

数据通道：<device_id>/out/read/app123

返回数据：{ "2": "MicoKit3288", "3": "MXCHIP" }

状态通道：<device_id>/out/err

状态数据：{ "status": 0 }

(3) APP 读取 rgb_led 开关状态：

发送通道：<device_id>/in/read/app123

发送数据：{"5":5}

设备响应：

数据通道：<device_id>/out/read/app123

返回数据：{ "5": false }

状态通道：<device_id>/out/err

状态数据：{ "status": 0 }

(4) APP 设置 rgb_led 灯为蓝色, 饱和度 100, 亮度 100：

发送通道：<device_id>/in/write/app123

发送数据：{"5":true, "6":240, "7":100, "8":100}

设备响应：

数据通道：<device_id>/out/write/app123

返回数据：{ "5": true, "6": 240, "7": 100, "8": 100 }

状态通道：<device_id>/out/err

状态数据：{ "status": 0 }

(5) 设备 ADC 自动上报 ADC 采样数据：

上报通道：<device_id>/out/read

上报数据：{ "10": 3700 }

(6) 关闭 ADC 自动上报：

发送通道：<device_id>/in/write/app123

发送数据：{ "11": false }

设备响应：

数据通道：<device_id>/out/write/app123

返回数据：{ "11": false }

状态通道：<device_id>/out/err

状态数据：{ "status": 0 }

(7) APP 向设备串口发送数据：

发送通道：<device_id>/in/write/app123

发送数据：{ "13": "message from cloud!" }

设备响应：

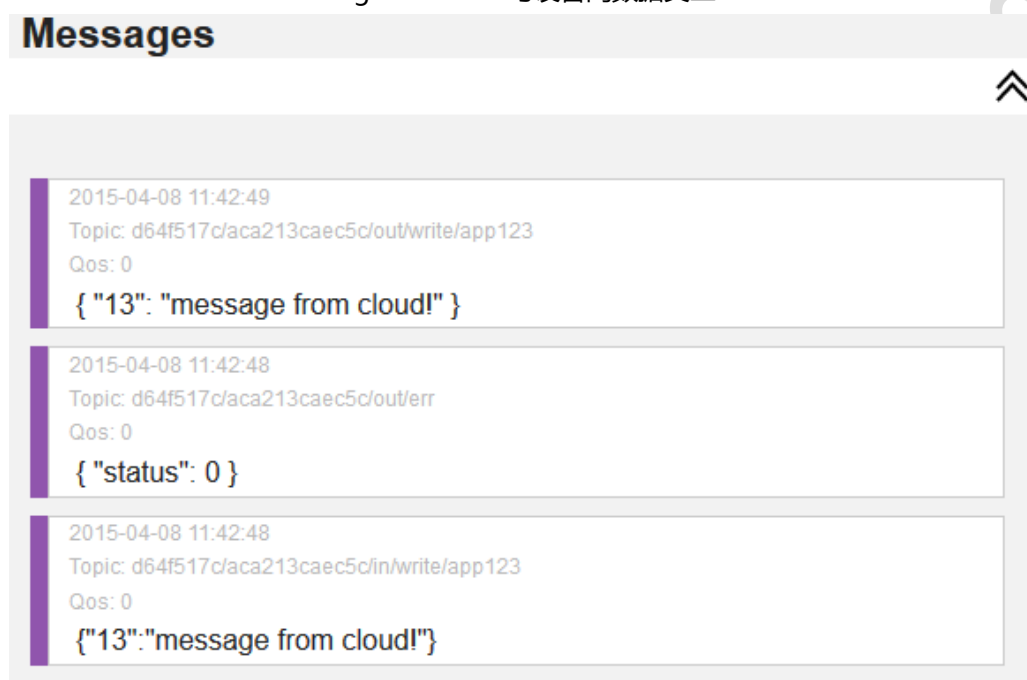
数据通道：<device_id>/out/write/app123

返回数据：{"13": "message from cloud!"}

状态通道：<device_id>/out/err

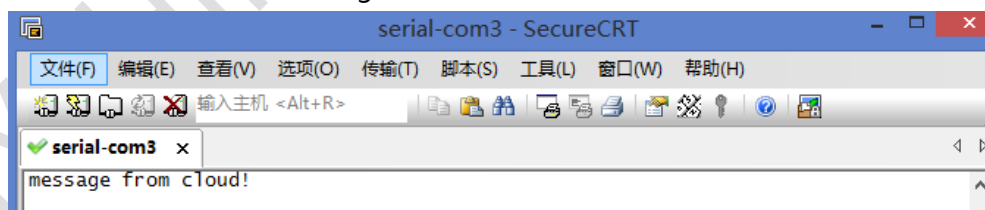
状态数据：{ "status": 0 }

Figure 4. APP 与设备间数据交互



并且设备串口输出数据：

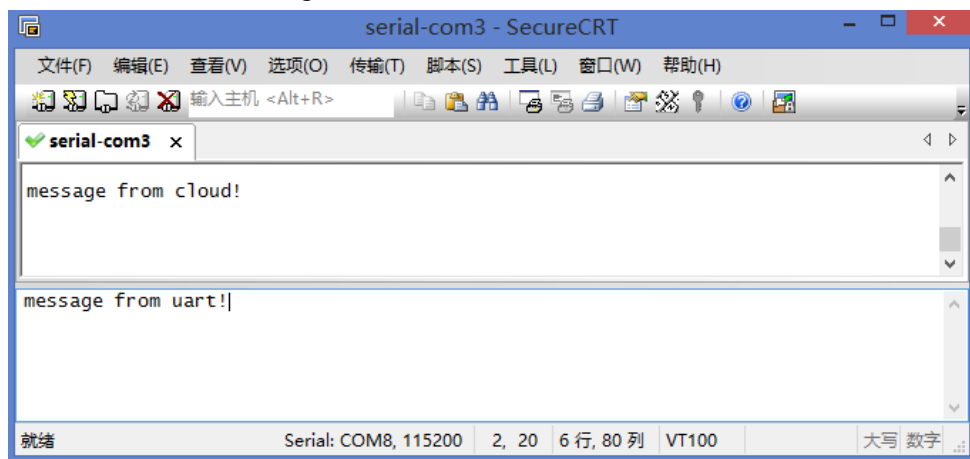
Figure 5. 设备串口输出数据



(8) 从设备串口向云端发送数据：

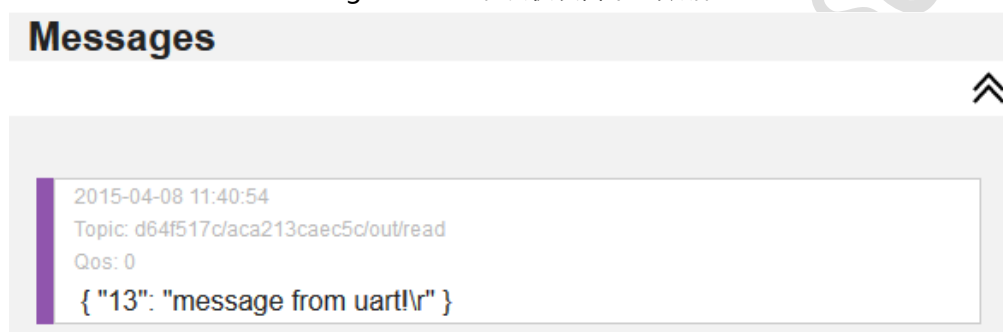
- 设备串口发送：

Figure 6. 设备串口向云端发送数据



- APP 收到消息：

Figure 7. 云端接收设备串口数据



9. 修订记录

版本	作者	时间	内容
V0.1.0	wanges@mxchip.com	2015.3.11	初始版本
V0.1.3	wanges@mxchip.com	2015.4.3	新版本协议，使用设备描述表
V0.1.5	wanges@mxchip.com	2015.4.8	添加 err 通道，添加 APP 操作实例
V0.1.6	wanges@mxchip.com	2015.4.24	设备描述表删除 value 字段，修改文档名称
V0.1.7	wanges@mxchip.com	2015.6.9	设备描述表恢复 value 字段，添加 common 模块