ooxWord://word/media/image76.png

**移动网络相关接口说明**

版本： V1.2

日期：2018-10-18

目 录

[版本历史 4](#_Toc561393066)

[1. 链接移动网络 6](#_Toc946103307)

[1.1 接口调用方式如下： 6](#_Toc1177373068)

[1.2 接收到responsed的处理过程 7](#_Toc1377721993)

[2. 获取运营商信息 8](#_Toc1952986688)

[2.1 网络制式和漫游状态 8](#_Toc1924531701)

[2.1.1 接口调用方式如下： 8](#_Toc215674725)

[2.1.2 接收到responsed的处理 9](#_Toc319481893)

[2.2 网络名称 11](#_Toc1744395673)

[2.2.1 接口调用方式如下： 11](#_Toc330962084)

[2.2.2 接收到responsed的处理 12](#_Toc1425091237)

[3 数据连接激活的接口 13](#_Toc324142216)

[3.1 接口调用方式如下： 13](#_Toc445619833)

[3.2 接收到responsed的处理流程 15](#_Toc246791237)

[4. 数据连接断开的接口 16](#_Toc1089789876)

[4.1 接口调用方式如下： 16](#_Toc1840835873)

[4.2 接收到responsed的处理流程 17](#_Toc898454979)

[5. 获取信号强度的接口 18](#_Toc488204636)

[5.1 接口调用方式如下： 18](#_Toc281639989)

[5.2 接收到responsed的处理过程 19](#_Toc735666727)

[6. 获取sim卡状态的接口 21](#_Toc1992217814)

[6.1 接口调用方式如下： 21](#_Toc796993616)

[6.2 接收到responsed的处理过程 22](#_Toc1980133495)

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 备注 |
| V1.1 | 2018.9.26 | Yanyan.An |  |
| V1.2 | 2018.10.18 | Yanyan.An |  |
|  |  |  |  |
|  |  |  |  |

声明

本文件所含数据和信息都属于紫光展锐机密及紫光展锐财产，紫光展锐保留所有相关权利。当您接受这份文件时，即表示您同意此份文件内含机密信息，且同意在未获得紫光展锐同意前，不使用或复制、整个或部分文件。紫光展锐有权在未经事先通知的情况下，对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与文件相关的直接或间接的、任何伤害或损失。

# 1. 链接移动网络

## 1.1 接口调用方式如下：

#define RIL\_REQUEST\_RADIO\_POWER 23

static void radioPower(bool isOn) {

int ret;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_RADIO\_POWER;

rnode->isFree = false;

Parcel p;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

p.writeInt32(1);

p.writeInt32(isOn?1:0);

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

ALOGI("write 'RadioPowerOn' into socket OVER!");

}

其中：

1). RIL\_REQUEST\_RADIO\_POWER 宏是固定值，不可随意更改。

2). s\_serial 的值是唯一的，在receive responsed时，需要用到此值。

3). 参数isOn：true表示开协议栈，false表示关协议栈。

## 1.2 接收到responsed的处理过程

p.readInt32(&token);

p.readInt32(&err);

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是23）。

Error = 0 表示请求处理的结果是success，其它值表示fail

**注：具体的代码实现可以参见 app\_demo.cpp**

# 2. 获取运营商信息

## 2.1 网络制式和漫游状态

### 2.1.1 接口调用方式如下：

#define RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE 20

static void getVoiceNetworkRegistrationState(){

int ret;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_VOICE\_REGISTRATION\_STATE;

rnode->isFree = false;

Parcel p;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

}

### 2.1.2 接收到responsed的处理

p.readInt32(&num);

ALOGI("num = %d", num);

for(index = 0; index < num; index++){

responsed[index] = strdupReadString(p);

}

value = atoi(responsed[0]);

regState = (RIL\_RegState)value;

ALOGI("net reg status: %d", value);

**//If condition is true, indicates that the current network is roaming.**

if (regState == RIL\_REG\_STATE\_ROAMING){

s\_is\_roaming = true;

}

ALOGI("Is if roaming? %d", s\_is\_roaming);

**//current network type**

if(regState == RIL\_REG\_STATE\_HOME){

s\_net\_type = atoi(responsed[3]);

ALOGI("net type: %d", s\_net\_type);

}

for(index = 0; index < num; index++){

free(responsed[index]);

}

说明：

static int s\_net\_type = -1; //Record current network type

static bool s\_is\_roaming = false; //Record current network if roaming

其中s\_net\_type枚举值如下:

typedef enum {

RADIO\_TECH\_UNKNOWN = 0,

RADIO\_TECH\_GPRS = 1,

RADIO\_TECH\_EDGE = 2,

RADIO\_TECH\_UMTS = 3,

RADIO\_TECH\_IS95A = 4,

RADIO\_TECH\_IS95B = 5,

RADIO\_TECH\_1xRTT = 6,

RADIO\_TECH\_EVDO\_0 = 7,

RADIO\_TECH\_EVDO\_A = 8,

RADIO\_TECH\_HSDPA = 9,

RADIO\_TECH\_HSUPA = 10,

RADIO\_TECH\_HSPA = 11,

RADIO\_TECH\_EVDO\_B = 12,

RADIO\_TECH\_EHRPD = 13,

RADIO\_TECH\_LTE = 14,

RADIO\_TECH\_HSPAP = 15, // HSPA+

RADIO\_TECH\_GSM = 16, // Only supports voice

RADIO\_TECH\_TD\_SCDMA = 17,

RADIO\_TECH\_IWLAN = 18,

RADIO\_TECH\_LTE\_CA = 19

} RIL\_RadioTechnology;

测试结果：

01-02 19:41:55.099 1259 1260 I AIPenBox: Is if roaming? 0 //是否处于漫游

01-02 19:41:55.099 1259 1260 I AIPenBox: net type: 14 //当前网络制式

**注：具体的代码实现可以参见 app\_demo.cpp**

## 2.2 网络名称

### 2.2.1 接口调用方式如下：

#**define** RIL\_REQUEST\_OPERATOR 22

static void getOperatorPlmn(){

int ret;

Parcel p;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_OPERATOR;

rnode->isFree = false;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if(ret < 0){

ALOGE("send at error");

}

ALOGI("write 'get operator plmn' into socket over!");

}

### 2.2.2 接收到responsed的处理

p.readInt32(&num);

ALOGI("num = %d", num);

for(index = 0; index < num; index++){

responsed[index] = strdupReadString(p);

ALOGI("num[%d]: %s", index, responsed[index]);

}

//finally free mem

for(index = 0; index < num; index++){

free(responsed[index]);

}

其中，responsed[0]是运营商长名称，responsed[1]是运营商短名称，responsed[2]是mcc+mnc

测试结果：

01-02 19:41:55.099 1259 1260 I AIPenBox: get operator plmn success!

01-02 19:41:55.099 1259 1260 I AIPenBox: num = 3

01-02 19:41:55.099 1259 1260 I AIPenBox: num[0]: CHN-UNICOM

01-02 19:41:55.099 1259 1260 I AIPenBox: num[1]: CUCC

01-02 19:41:55.099 1259 1260 I AIPenBox: num[2]: 46001

**注：具体的代码实现可以参见 app\_demo.cpp**

# 3 数据连接激活的接口

## 3.1 接口调用方式如下：

#define RIL\_REQUEST\_SETUP\_DATA\_CALL 27

static void setupDataCall(const char\* radioTechnology, const char\* profile, const char\* apn, const char\* user, const char\* password, const char\* authType, const char\* protocol) {

int ret;

Parcel p;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_SETUP\_DATA\_CALL;

rnode->isFree = false;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

p.writeInt32(7);

writeStringToParcel(p, radioTechnology);

writeStringToParcel(p, profile);

writeStringToParcel(p, apn);

writeStringToParcel(p, user);

writeStringToParcel(p, password);

writeStringToParcel(p, authType);

writeStringToParcel(p, protocol);

ALOGI("setup data call");

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

ALOGI("setup data call OVER!");

}

其中：

1). RIL\_REQUEST\_SETUP\_DATA\_CALL 宏是固定值，不可随意更改。

2). s\_serial的值是唯一的，在receive responsed时，需要用到此值。

3). 参数信息如下：

**@param** radioTechnology

\* indicates whether to setup connection on radio technology CDMA

\* (0) or [GSM](http://10.0.1.79:8081/source/s?path=GSM/)/[UMTS](http://10.0.1.79:8081/source/s?path=GSM/UMTS) (1)

\* **@param** profile

\* Profile Number or NULL to indicate default profile

\* **@param** apn

\* the APN to connect to if radio technology is [GSM](http://10.0.1.79:8081/source/s?path=GSM/)/[UMTS](http://10.0.1.79:8081/source/s?path=GSM/UMTS).

\* Otherwise null for CDMA.

\* **@param** user

\* the username for APN, or NULL

\* **@param** password

\* the password for APN, or NULL

\* **@param** authType

\* the PAP / CHAP auth type. Values is one of SETUP\_DATA\_AUTH\_\*

\* **@param** protocol

\* one of the PDP\_type values in TS 27.007 section 10.1.1.

\* For example, "IP", "IPV6", "IPV4V6", or "PPP".

## 3.2 接收到responsed的处理流程

1. 首先从Parcel中依次读取出 token及error值。

p.readInt32(&token);

p.readInt32(&err);

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是27）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

1. 反馈的数据部分按照提供的demo code进行解析。

注：如果error值表示失败，那么需要查询失败的cause，根据cause值来确定是否需要重试操作。相关的demo code也已经提供。

**注：具体的代码实现可以参见 app\_demo.cpp**

# 4. 数据连接断开的接口

## 4.1 接口调用方式如下：

#define RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL 41

static void deactivateDataCall(int cid){

int ret;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL;

rnode->isFree = false;

Parcel p;

char temp\_buf[4];

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

p.writeInt32(2);

memset(temp\_buf, 0, sizeof(temp\_buf));

snprintf(temp\_buf, sizeof(temp\_buf), "%d", cid);

writeStringToParcel(p, temp\_buf); // cid

writeStringToParcel(p, "0"); //reason

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

ALOGI("write 'deactivate data call' into socket OVER!");

}

其中：

1). RIL\_REQUEST\_DEACTIVATE\_DATA\_CALL 宏是固定值，不可随意更改。

2). s\_serial的值是唯一的，在receive responsed时，需要用到此值。

3). 参数信息如下：

**@param** cid

\* indicates which one pdp link.

## 4.2 接收到responsed的处理流程

1. 首先从Parcel中依次读取出token及error值。

p.readInt32(&token);

p.readInt32(&err);

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是41）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

1. 反馈的数据部分是NULL，不用关注。

**注：具体的代码实现可以参见 app\_demo.cpp**

# 5. 获取信号强度的接口

## 5.1 接口调用方式如下：

#define RIL\_REQUEST\_SIGNAL\_STRENGTH 19

static void getSignalStrength(){

int ret;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_SIGNAL\_STRENGTH;

rnode->isFree = false;

Parcel p;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

ALOGI("write 'get signal strength' into socket OVER!");

}

其中：

1). RIL\_REQUEST\_SIGNAL\_STRENGTH 宏是固定值，不可随意更改。

2). s\_serial的值是唯一的，在receive responsed时，需要用到此值。

## 5.2 接收到responsed的处理过程

1） 首先从Parcel中依次读取出 token及error值。

p.readInt32(&token);

p.readInt32(&err);

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是19）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 接下来继续从Parcel中反序列化出下面signalStrength类型的结构体数据：

typedef struct {

RIL\_GW\_SignalStrength GW\_SignalStrength;

RIL\_CDMA\_SignalStrength CDMA\_SignalStrength;

RIL\_EVDO\_SignalStrength EVDO\_SignalStrength;

RIL\_LTE\_SignalStrength\_v8 LTE\_SignalStrength;

RIL\_TD\_SCDMA\_SignalStrength TD\_SCDMA\_SignalStrength;

} RIL\_SignalStrength\_v10;

**目前可以只关注LTE\_SignalStrength.rsrp的值(即信号强度的值)，它的单位是dBm.**

**注：具体的代码实现可以参见 app\_demo.cpp**

# 6. 获取sim卡状态的接口

## 6.1 接口调用方式如下：

#define RIL\_REQUEST\_GET\_SIM\_STATUS 1

static void getIccCardState(){

int ret;

requestList\* rnode;

rnode = findFreeListNode();

if(rnode == NULL) return;

rnode->serial = s\_serial++;

rnode->requestId = RIL\_REQUEST\_GET\_SIM\_STATUS;

rnode->isFree = false;

Parcel p;

p.writeInt32(rnode->requestId);

p.writeInt32(rnode->serial);//token

ret = sendRequest(p.data(), p.dataSize(), s\_socket\_fd);

if (ret < 0) {

ALOGE("send at ERROR!");

}

ALOGI("write 'get sim status' into socket OVER!");

}

其中：

1). RIL\_REQUEST\_GET\_SIM\_STATUS 宏是固定值，不可随意更改。

2). s\_serial的值是唯一的，在receive responsed时，需要用到此值。

## 6.2 接收到responsed的处理过程

1） 首先从Parcel中依次读取出 token及error值。

p.readInt32(&token);

p.readInt32(&err);

token：即我们调用接口时传递下去的值，根据此值来判断是哪个请求ID（比如这里的token值对应的请求ID值就是1）。

Error = 0 表示请求处理的结果是success, 其它值表示fail

2） 接下来继续从Parcel中反序列化出RIL\_CardStatus类型的结构体数据：

for(index = 0; index < cardstatus.num\_applications; ++index){

p.readInt32(&value);

cardstatus.applications[index].app\_type = (RIL\_AppType)value;

p.readInt32(&value);

**cardstatus.applications[index].app\_state = (RIL\_AppState)value;**

p.readInt32(&value);

cardstatus.applications[index].perso\_substate = (RIL\_PersoSubstate)value;

cardstatus.applications[index].aid\_ptr = strdupReadString(p);

cardstatus.applications[index].app\_label\_ptr = strdupReadString(p);

p.readInt32(&cardstatus.applications[index].pin1\_replaced);

p.readInt32(&value);

cardstatus.applications[index].pin1 = (RIL\_PinState)value;

p.readInt32(&value);

cardstatus.applications[index].pin2 = (RIL\_PinState)value;

}

**目前可以只关注结构体中applications[index].app\_state的值(即sim状态的值，它是一个RIL\_AppState枚举结构)，具体定义可见sprd\_ril.h。**

**注：具体的代码实现可以参见 app\_demo.cpp**