

# RIL in 1826

---

## *Revision 0.5*

About this document .....	1
1. RIL design in 1826 .....	2
1.1 RIL design overview .....	2
1.2 RIL ubus interface .....	2
1.2.1 RIL object for ril request.....	2
1.2.2 RIL method .....	3
1.2.3 RIL objects for unsolicited message.....	3
1.2.4 Solicited and unsolicited message .....	5
1.3 RIL data flow.....	6
1.4 RIL data parse.....	6

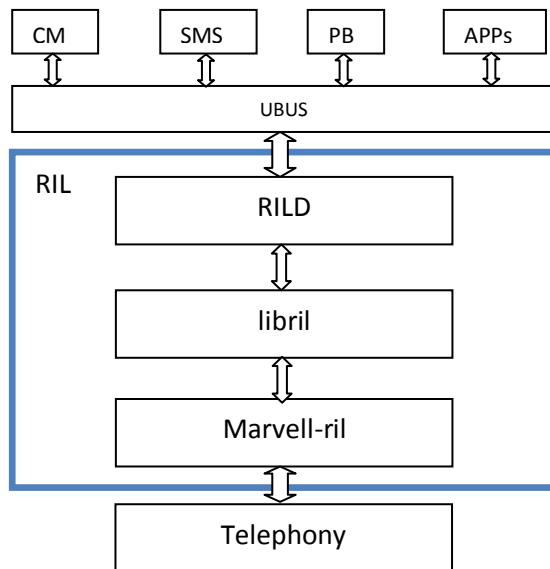
## **About this document**

This document describes the design and usage of RIL based on openwrt in 1826(nezha) Soc. We port RIL from android source code to openwrt platform. According to our openwrt MIFI design, RIL hooks ubus(an IPC mechanism) as an ubus service, any application who wants to use radio need to communicate with RIL through ubus.

We redesign RILD, hooks it to ubus in 1826, keep libril and marvell-ril layers as their original design.

# 1. RIL design in 1826

## 1.1 RIL design overview



**RILD:** a daemon, supply ubus interfaces to any application who wants to use RIL.

**libril:** RIL request's data processing.

ril request is packed as Parcel in android. In 1826, ril request is packed as blob(more information in libubox), this ril request blob contains two things:

1. RIL request id.
2. RIL request data.

**marvell-ril:** RIL request's logic processing.

1. Converts RIL requests to AT commands send to radio,
2. Receives solicited and unsolicited messages from radio.

## 1.2 RIL ubus interface

### 1.2.1 RIL object for ril request

RIL service name is "ril". Any application who wants to use RIL service can lookup RIL's id by RIL's name as this:

```
ubus_lookup_id(ctx, "ril", &id);
```

Definition of RIL service object:

```
static struct ubus_object ril_object = {  
    .name = "ril",
```

```
.type = &ril_object_type,

.methods = ril_methods,

.n_methods = ARRAY_SIZE(ril_methods),

};
```

### 1.2.2 RIL method

#### I. ril\_request

Any application who wants to call this method must use method name “ril\_request” as this:

```
ubus_invoke_async(ctx, "ril_request", b.head, &req);
```

Parameters:

*reqid*: ril request id.

*data*: ril request data.

Application packs ril request like this:

```
blob_buf_init(msg, 0);
```

```
blobmsg_add_u32(msg, "reqid", id); //id is ril request id, "reqid" is the name of request id item
```

```
blob_put(msg, 0, data, len);
```

You can also use pack API in librilutil.so, pack API is:

```
rilutil_makeRequestBlob
```

We recommend you use librilutil’s API to pack data.

Definition of RIL method:

```
static const struct ubus_method ril_methods[] = {

    UBUS_METHOD_NOARG("ril_request", Rild_request),

};
```

### 1.2.3 RIL objects for unsolicited message

RIL receives unsolicited messages from radio, these unsolicited messages are divided as different groups, we define RIL objects according to these groups, application who wants receive unsolicited message from radio can subscribe these RIL objects.

Current we define 8 groups, echo group have a RIL object for application to subscribe:

- I. {name = "ril.unsol.cc"}
- II. {name = "ril.unsol.dev"}
- III. {name = "ril.unsol.mm"}
- IV. {name = "ril.unsol.msg"}
- V. {name = "ril.unsol.ps"}

- VI. {*.name* = "*ril.unsol.sim*"}
- VII. {*.name* = "*ril.unsol.ss*"}
- VIII. {*.name*= "*ril.unsol*"}

The map of unsolicited messages and groups:

<i>{RIL_UN SOL_SIGNAL_STRENGTH,</i>	<i>SERVICE_MM},</i>
<i>{RIL_UN SOL_RESPONSE_VOICE_NETWORK_STATE_CHANGED,</i>	<i>SERVICE_MM},</i>
<i>{RIL_UN SOL_VOICE_RADIO_TECH_CHANGED,</i>	<i>SERVICE_MM},</i>
<i>{RIL_UN SOL_NITZ_TIME_RECEIVED,</i>	<i>SERVICE_MM},</i>
<i>{RIL_UN SOL_RESPONSE_CALL_STATE_CHANGED,</i>	<i>SERVICE_CC},</i>
<i>{RIL_UN SOL_DATA_CALL_LIST_CHANGED,</i>	<i>SERVICE_PS},</i>
<i>{RIL_UN SOL_RESPONSE_IMS_NETWORK_STATE_CHANGED,</i>	<i>SERVICE_PS},</i>
<i>{RIL_UN SOL_RESPONSE_NEW_SMS,</i>	<i>SERVICE_MSG},</i>
<i>{RIL_UN SOL_RESPONSE_NEW_SMS_ON_SIM,</i>	<i>SERVICE_MSG},</i>
<i>{RIL_UN SOL_RESPONSE_NEW_SMS_STATUS_REPORT,</i>	<i>SERVICE_MSG},</i>
<i>{RIL_UN SOL_SIM_SMS_STORAGE_FULL,</i>	<i>SERVICE_MSG},</i>
<i>{RIL_UN SOL_RESPONSE_NEW_BROADCAST_SMS,</i>	<i>SERVICE_MSG},</i>
<i>{RIL_UN SOL_RESPONSE_SIM_STATUS_CHANGED,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_CP_PHONEBOOK_INITED,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_CALL_SETUP_STATUS,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_PROACTIVE_COMMAND,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_EVENT_NOTIFY,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_SESSION_END,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_CALL_SETUP_RESULT,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_SEND_SM_STATUS,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_SEND_SM_RESULT,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_STK_SEND USSD_RESULT,</i>	<i>SERVICE_SIM},</i>
<i>{RIL_UN SOL_SIM_REFRESH,</i>	<i>SERVICE_MM},</i>
<i>{RIL_UN SOL_SUPP_SVC_NOTIFICATION,</i>	<i>SERVICE_SS},</i>
<i>{RIL_UN SOL_ON USSD,</i>	<i>SERVICE_SS},</i>

Unsolicited messages aren't in this map will be included "ril.unsol" group.

Application subscribe like this:

```
ubus_lookup_id(ctx, "ril.unsol.ps", &id); //lookup id by name firstly
```

```
ubus_subscribe(ctx, &ril_events, id); //subscribe id
```

#### 1.2.4 Solicited and unsolicited message

Both solicited and unsolicited message are format as blob, this blob contains 3 things:

1. RIL request id for solicited message; unsolicited id for unsolicited message;
2. RIL response error code for RIL request, as follow; for unsolicited message, errorcode is 0;

```
RIL_E_SUCCESS = 0,  
RIL_E_RADIO_NOT_AVAILABLE = 1, /* If radio did not start or is resetting */  
RIL_E_GENERIC_FAILURE = 2,  
RIL_E_PASSWORD_INCORRECT = 3, /* for PIN/PIN2 methods only! */  
RIL_E_SIM_PIN2 = 4, /* Operation requires SIM PIN2 to be entered */  
RIL_E_SIM_PUK2 = 5, /* Operation requires SIM PIN2 to be entered */  
RIL_E_REQUEST_NOT_SUPPORTED = 6,  
RIL_E_CANCELLED = 7,  
RIL_E_OP_NOT_ALLOWED_DURING_VOICE_CALL = 8, /* data ops are not allowed during voice  
call on a Class C GPRS device */  
RIL_E_OP_NOT_ALLOWED_BEFORE_REG_TO_NW = 9, /* data ops are not allowed before device  
registers in network */  
RIL_E_SMS_SEND_FAIL_RETRY = 10, /* fail to send sms and need retry */  
RIL_E_SIM_ABSENT = 11, /* fail to set the location where CDMA subscription  
shall be retrieved because of SIM or RUIM  
card absent */  
RIL_E_SUBSCRIPTION_NOT_AVAILABLE = 12, /* fail to find CDMA subscription from specified  
location */  
RIL_E_MODE_NOT_SUPPORTED = 13, /* HW does not support preferred network type */  
RIL_E_FDN_CHECK_FAILURE = 14, /* command failed because recipient is not on FDN list */  
RIL_E_ILLEGAL_SIM_OR_ME = 15, /* network selection failed due to  
illegal SIM or ME */  
RIL_E_MISSING_RESOURCE = 16, /* for NFC smartcard API 2.3.2 */  
RIL_E_NO_SUCH_ELEMENT = 17, /* for NFC smartcard API 2.3.2 */  
RIL_E_INVALID_PARAMETER = 18 /* for NFC smartcard API 2.3.2 */
```

3. Data. Since there is no Parcel in openwrt, we use raw data from response functions (responseXXX).

Message is packed as this:

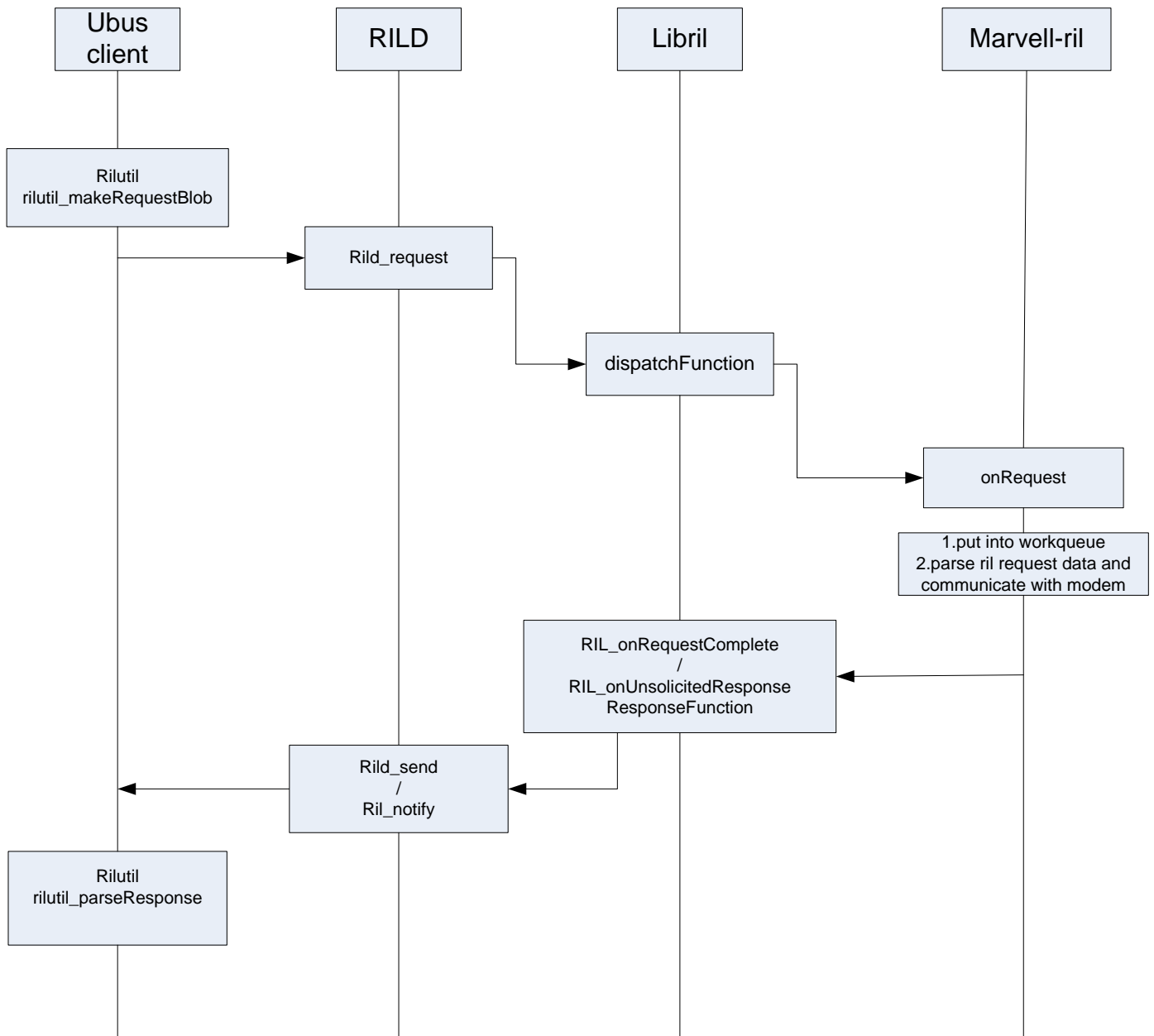
```
blob_buf_init(msg, 0);
```

```
blobmsg_add_u32(msg, "rilid", id);
```

```
blobmsg_add_u32(msg, "resperrno", rilresperrcode);
```

```
blob_put(msg, 0, data, len);
```

### 1.3 RIL data flow



### 1.4 RIL data parse

In android, Parcel is used in RIL, but in openwrt, it doesn't have Parcel, so we must re-define the data parse related functions instead of Parcel. For "dispatchXXX" functions, they receive raw data from applications, raw data must be format as a right structure that can be use in dispatch functions.

We create a layer named "rilutil" to pack and unpack blob data for ril request and response, any application want to use ril may use this layer's APIs to pack and unpack blob. Rilutil layer is compiled to "librilutil.so".

#### **Pack API:**

```
int rilutil_makeRequestBlob(struct blob_buf *msg, unsigned int requestid, const void * data, int len);
```

#### **Unpack API:**

```
int rilutil_parseResponse(struct blob_attr *msg, unsigned int *requestid, unsigned int *rilerrno, void
**response, int *responselen);
```

Example:

```
#include "rilutil.h"

rilutilstrings strings;

strings.num = 2;

strings.str[0] = "abcdef";

strings.str[1] = "12345667890";

//pack data into blob

rilutil_makeRequestBlob(&b, 9, &strings, sizeof(rilutilstrings)); //pack data into blob_buf "b".

//send to rild

ubus_invoke(ctx, id, "ril_request", b.head, a_data_cb_from_m1, 0, 0);


//unpack blob from rild

unsigned int rilid = 0;

unsigned int rilerror = 0;

void * data = NULL;

int datalen = 0;

rilutil_parseResponse(msg, &rilid, &rilerror, &data, &datalen); //unpack blob

if(data) free(data); //free data if need
```

Users can also add their pack and unpack functions in "rilutil.c", pack functions named "dispatchXXX", unpack functions named "responseXXX", see "rilutil.c" for details. If you add dispatch functions in "rilutil.c", you need to insure dispatch functions in "ril.c" can parse the blob you packed.

Ril request blob:

Ril request id
Raw Data

Ril response blob:

Ril request id  /Unsolicited id
Error code
Raw data