# STK 理论与实现

|  |  |  |
| --- | --- | --- |
| Date  日期 | Version  版本 | Comments  备注 |
| 2019-04-04 | 0.1 | First version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Function  职位 | Name  姓名 | Date  日期 | Signature  签名 |
| Written by  拟定 | SW | 曾灿炫 | 2019-04-04 | 曾灿炫 |
| Verified by  审核 |  |  |  |  |
| Verified by  审核 |  |  |  |  |
| Approved by  批准 |  |  |  |  |

目录

[STK 理论与实现 1](#_Toc5295120)

[1. 概述 4](#_Toc5295121)

[**1.1.** **本文档的目的** 4](#_Toc5295122)

[**1.2.** **背景** 4](#_Toc5295123)

[**1.3.** **缩略语清单** 4](#_Toc5295124)

[**1.4.** **参考文献** 4](#_Toc5295125)

[2. 高通Modem简述 5](#_Toc5295126)

[2.1. STK涉及模块 6](#_Toc5295127)

[3. Profile Download 7](#_Toc5295128)

[3.1. 概念 7](#_Toc5295129)

[3.1.1. 定义 7](#_Toc5295130)

[3.1.2. Terminal Profile解读 7](#_Toc5295131)

[3.2. 关键流程分析 8](#_Toc5295132)

[3.2.1. 整体流程图 8](#_Toc5295133)

[3.2.2. Modem端流程 8](#_Toc5295134)

[3.2.3. 代码分析 9](#_Toc5295135)

[3.3. 小结 14](#_Toc5295136)

[4. Proactive Command 14](#_Toc5295137)

[4.1. 概念 14](#_Toc5295138)

[4.1.1. 定义 14](#_Toc5295139)

[4.1.2. 流程 14](#_Toc5295140)

[4.2. 关键流程分析 15](#_Toc5295141)

[4.2.1 流程图 15](#_Toc5295142)

[4.2.2. log分析 16](#_Toc5295143)

[4.2.3. 时序图 19](#_Toc5295144)

[4.3. 经验分享 20](#_Toc5295145)

[4.4. 小结 20](#_Toc5295146)

[5. Envelope Command 21](#_Toc5295147)

[5.1. 概念 21](#_Toc5295148)

[5.1.1. 简介 21](#_Toc5295149)

[5.1.2. 应用机制 21](#_Toc5295150)

[5.1.3. 命令结构 21](#_Toc5295151)

[5.2. Modem端实现 22](#_Toc5295152)

[5.2.1. 涉及模块 23](#_Toc5295153)

[5.2.2. 关键函数 23](#_Toc5295154)

[5.2.3. log分析 24](#_Toc5295155)

[5.2.4. 时序图 26](#_Toc5295156)

[5.2.5. 小结 27](#_Toc5295157)

[6. Bearer Independent Protocol 27](#_Toc5295158)

[6.1. 概念 27](#_Toc5295159)

[6.1.1. BIP的引入 27](#_Toc5295160)

[6.1.2. 引入BIP的OTA结构图 28](#_Toc5295161)

[6.1.3. BIP示意图 28](#_Toc5295162)

[6.1.4. 使用的STK机制 29](#_Toc5295163)

[6.2. Modem端实现 30](#_Toc5295164)

[6.2.1. 涉及模块 30](#_Toc5295165)

[6.2.2. log解析 31](#_Toc5295166)

[6.2.3. 时序图 36](#_Toc5295167)

[6.3. 小结 36](#_Toc5295168)

[7. 总结 36](#_Toc5295169)

# 概述

* 1. **本文档的目的**

*本文档作为一个高通Modem的学习输出，总结STK模块的基础知识，结合分析高通平台的实现方式。主要包括协议阅读，代码跟进与log分析。*

* 1. **背景**

*STK即SIM Tool Kit，它提供一系列用于移动设备与SIM卡间交互的机制。通过这些机制，支持STK的手机可以操作SIM卡里的应用。STK涉及的机制主要有Profile Download，Envelope Command, Proactive Command, BIP等。*

* 1. **缩略语清单**

*列出文中使用的术语的定义和缩略（语）词的英文全名和中文解释。*

| Term | Explanation |
| --- | --- |
| TP | Terminal Profile |
| TR | Terminal Response |
| ME | Mobile Equipment |
| BIP | Bearer Independent Protocol |

* 1. **参考文献**

| Document | Explanation |
| --- | --- |
| [3GPP TS 11.11] | Technical Specification Group Terminals Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface. |
| [3GPP TS 11.14] | Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment(SIM - ME) interface. |
| [3GPP 31.111] | Universal Subscriber Identity Module (USIM) Application Toolkit (USAT) |
| 80-ng610-1\_k\_gstk\_overview | 高通文档,GSTK模块介绍 |
| 80-nk374-1\_e\_mmgsdi\_overview | 高通文档，MMGSDI模块介绍 |
| BIP Basic Introduction | https://online.mediatek.com/QuickStart/8a747c4a-8807-4fce-9e36-931eee5004b1 |
| 基于BIP 协议的高速OTA 下载系统的设计 | http://www.paper.edu.cn |
|  |  |
|  |  |

# 高通Modem简述

学习相关机制前，先了解一下Modem的基本组成模块，结构图如下（双卡模式）：



## STK涉及模块

STK命令处理过程中，主要涉及的模块如下：

**UIM:**负责卡相关的程序，也是卡与UE间的唯一通信接口，分为UIM driver和UIM server。driver能启动，重启卡，发送指令等；server面向mmgsdi、gstk等，其中一项作用便是支持STK相关的操作，如主动式命令流程中的fetch。参考文档为80-nh940-1\_f。

**MMGSDI(Multimode Generic SIM Driver Interface)：**提供通用库函数，供SIM卡调用，另一方面供上层检索、修改SIM卡的数据。

**GSTK(Generic SIM Application Toolkit）：**STK的核心处理模块，负责STK相关流程，发送TP，解析来自SIM卡的主动式命令和Envelope response，向外提供发送TR和Envelop命令的库函数。参考文档为80-ng610-1\_k。

**QMI CAT：**高通项目中RIL层没有与Modem直接通信，而是通过中间层QMI+QCRIL实现消息的传递与方法调用，QMI (Qualcomm MSM™ Interface )分为QMI UIM和QMI CAT(QMI Card Application Toolkit)，负责STK业务的为QMI CAT，主要文件为qmi\_cat.c。

# Profile Download

## 概念

### 3.1.1. 定义

在SIM卡的初始化过程中，ME通过读取SIM卡的EFPHASE文件确定是否需要完成Profile Download。Profile Download即向SIM卡发送Terminal Profile命令，该命令通知SIM卡ME所能支持的SIM卡功能。比如，通过该过程，SIM卡可以知道ME端支持的具体主动式命令类型。

### 3.1.2. Terminal Profile解读

#### 3.1.2.1. 定义

Terminal Profile 包含30多个字节，每个字节都涉及8个功能，具体定义可查看3GPP TS 31.111。

如第一个字节：

First byte (Download):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | b8 | | b7 | | b6 | | b5 | | b4 | | b3 | | b2 | | b1 | |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | See TS 102 223 [32] clause 5.2 |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | SMS-PP data download |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | Cell Broadcast data download |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | See TS 102 223 [32] clause 5.2 |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | Bit = 1 if SMS-PP data download is supported |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | See TS 102 223 [32] clause 5.2 |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | Bit = 1 if Call Control by USIM is supported |
|  |  | |  | |  | |  | |  | |  | |  | |  | |  | | Bit = 1 if Call Control by USIM is supported |

#### 3.1.2.2. 命令解读

在具体的项目中，相关功能是否支持由pics值决定，Benz的pics表为80-PB971-5。

相关值在modem的gstk\_exp.h中定义(在defect7311073中出现pics表与具体代码值不同的情况，解决方案由高通提供)

具体的TP解析则需要借助文档GCF-CC-3720-F3.4\_USAT support进行转化。



## 关键流程分析

### 整体流程图

Profile Download流程图如下：

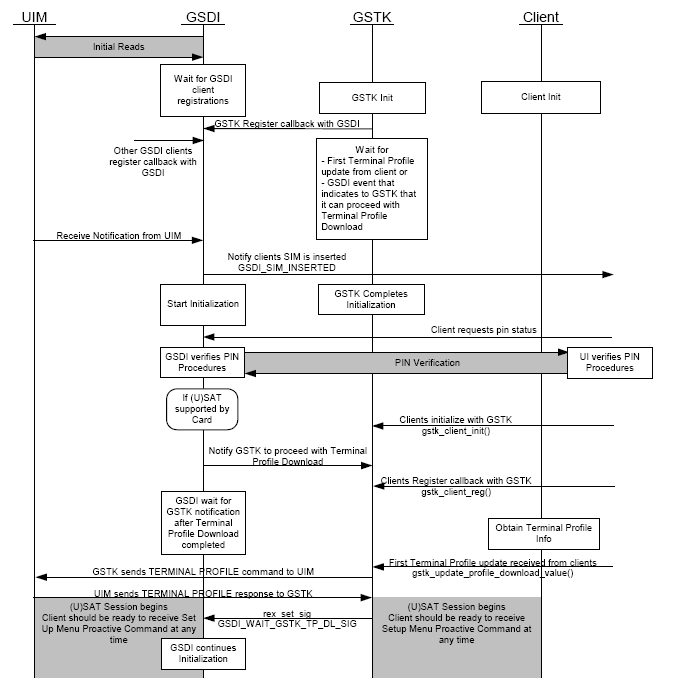


Phase文件指的是SIM卡所支持的GSM标准的级数，值为3则指SIM卡支持**GSM phase 2和profile download(3GPP 11.11)**。

1. ME通过读取EFphase文件确定SIM卡所提供的功能。
2. 向SIM发送Termianl Profile命令，告知SIM卡ME所能支持的功能。

### Modem端流程

1. 高通modem端的完整时序图(从卡插入到发送TP)如下：



1-TP时序图(80-ng610-1\_k\_gstk\_overview)

### 3.2.3. 代码分析

通过log的追踪，在modem端进行具体的代码跟进。

#### mmgsdi->gstk（构建TP）



如上为mmgsdi部分的时序图，mmgsdi通知gstk去构建TP并提供回调函数用于通知自身TP已发送完毕。

**mmgsdi\_card)init.c:**

1. 调用mmgsdi\_card\_init\_gstk\_terminal\_profile\_download去促使gstk发送TP

2. 函数调用mmgsdi\_card\_init\_is\_toolkit\_allowed去读取service table确定当前SIM卡是否支持STK。

3. 返回值为true则调用gstklib.c的gstk\_gsdi\_set\_fetch\_status设置fetch的状态(能否fetch命令)。

4. 调用gstklib.c的gstk\_slot\_send\_terminal\_profile\_dl\_command让gstk完成Terminal Profile的流程。这里提供**mmgsdi\_gstk\_terminal\_profile\_cb**回调函数，使gstk完成TP发送后通知mmgsdi。

**gstklib.c：**

1. 获取一个gstk的消息buffer(gstk\_task\_cmd\_type)，构建该buf并调用gstk.c的gstk\_task\_put\_cmd\_buf(设置其comman\_group为GSTK\_MMGSDI\_CMD)将其发送至gstk的命令队列。

#### gstk->uim（发送TP）

进入gstk，发送TP的时序图如下：



**gstk.c :**

1. gstk\_init\_post\_task\_startup gstk初始化时调用gstk\_update\_default\_app\_specific\_tp构建初始TP.(该函数内可跟进TP各个位的初始值)

2.gstk\_task获取队列的command，调用gstk\_process\_state(gstk\_state.c)检测当前状态。

**gstk\_state.c:**

1.读取当前gstk状态为GSTK\_NO\_SIM\_S，gstk\_state\_no\_sim( gstk\_s\_no\_sim.c)

**gstk\_s\_no\_sim.c：**

1.读取command\_group，为mmgsdi传来的GSTK\_MMGSDI\_CMD，进而确定command\_id为 GSTK\_MMGSDI\_TERMINAL\_PROFILE\_DL\_EVT，调用gstk\_pack\_tp\_cmd\_to\_uim

(gstk\_term\_profile)开始打包发送TP。

**gstk\_term\_profile.c:**

1.调用gstk\_update\_base\_tp更新gstk初始化时构建的TP，根据cat\_version限定TP的长度。

2.调用gstk\_send\_cmd\_to\_uim\_server(gstkutil.c)开始发送uim cmd。

**gstkutil.c:**

1.调用uim\_cmd将cmd\_ptr发送至uim的消息队列。

2.根据command类型，提供函数gstk\_uim\_terminal\_profile\_report(gstk\_term\_profile.c)，

当uim收到sim卡的respone时调用该函数构建gstk\_term\_profile\_rsp\_type，改变gstk状态，从GSTK\_NO\_SIM\_S到GSTK\_TERM\_PROFILE\_RSP\_WAIT\_S。gstk的状态机再根据命令类型执行对应的流程。

到了这一步，TP已经发送至SIM卡了，接下来就是等待SIM卡的response，执行发送后的流程。

#### uim->gstk（接收完成）

gstk\_uim\_terminal\_profile\_report主要部分如下：

task\_cmd = gstk\_task\_get\_cmd\_buf();

if ( task\_cmd != NULL ) { /\* successfully obtained command buffer \*/

/\* build the terminal profile response \*/

task\_cmd->cmd.terminal\_profile\_rsp.message\_header.sim\_slot\_id =

(gstk\_slot\_id\_enum\_type)report->slot;

task\_cmd->cmd.terminal\_profile\_rsp.message\_header.command\_group = GSTK\_TERMINAL\_PROFILE\_RSP;

task\_cmd->cmd.terminal\_profile\_rsp.message\_header.command\_id = (int)GSTK\_PROFILE\_DL\_IND\_RSP;

task\_cmd->cmd.terminal\_profile\_rsp.message\_header.user\_data = report->user\_data;

if (report->rpt\_status == UIM\_PASS) {

task\_cmd->cmd.terminal\_profile\_rsp.success = TRUE;

if (GSTK\_IS\_VALID\_SLOT\_ID(report->slot)) {

GSTK\_RETURN\_IF\_NULL\_PTR(gstk\_instances\_ptr[(uint8)report->slot - 1]);

gstk\_instances\_ptr[(uint8)report->slot - 1]->is\_tp\_rsp\_received = TRUE;

}

}

/\* set GSTK\_CMD\_Q\_SIG and put on GSTK queue \*/

gstk\_task\_put\_cmd\_buf(task\_cmd);

}

该函数与mmgsdi通知gstk的原理相似，都是获取gstk消息体gstk\_task\_cmd\_type并进行构建发送至队列。

回到gstk\_status.c，进入不同分支，执行发送TP后的流程，时序图如下：



**gstk.c :**

1.gstk\_task获取队列的command，调用gstk\_process\_state(gstk\_state.c)检测当前状态。

**gstk\_state.c:**

1.读取当前gstk状态为GSTK\_TERMINAL\_PROFILE\_RSP，调用gstk\_state\_tp\_rsp\_wait

(gstk\_s\_term\_profile\_rsp\_wait.c)

**gstk\_s\_term\_profile\_rsp\_wait.c :**

1.调用uim\_set\_proactive\_uim\_slot(uim.c)指示UIM为主动式uim

2.调用gstk\_util\_send\_message\_to\_clients(gstkutil.c)向所有注册了该事件的client发送response。

**gstkutil.c:**

1.调用estk\_bip\_full\_clean\_up清除BIP table

2.通过回调函数estk\_gstk\_evt\_cb清除estk存储的关于刚处理完的指令的数据。

#### gstk->mmgsdi（发送完成）

回到3.2.1第三步，当gstk\_slot\_send\_terminal\_profile\_dl\_command执行完毕，回调mmgsdi\_gstk\_terminal\_profile\_cb进入TP发送完毕的处理流程。该函数主要是构建一个mmgsdi的消息结构体mmgsdi\_task\_cmd\_type(MMGSDI\_TP\_DOWNLOAD\_COMPLETE\_REQ)，传入来自gstk的TP response，并放到mmgsdi的command quene当中。

进入mmgsdi的command处理函数mmgsdi\_process\_command，可看到根据command为MMGSDI\_TP\_DOWNLOAD\_COMPLETE\_REQ，函数又调用了mmgsdi\_process\_tp\_download\_complete，该函数主要部分如下：

mmgsdi\_status = mmgsdi\_util\_get\_client\_request\_table\_free\_index(&index);

if (mmgsdi\_status != MMGSDI\_SUCCESS)

{

return mmgsdi\_status;

}

mmgsdi\_status = mmgsdi\_util\_populate\_client\_request\_table\_info(

index,

&req\_ptr->request\_header,

NULL,

NULL);

if (mmgsdi\_status != MMGSDI\_SUCCESS)

{

mmgsdi\_util\_free\_client\_request\_table\_index(index);

return mmgsdi\_status;

}

mmgsdi\_status = mmgsdi\_util\_queue\_mmgsdi\_uim\_report\_rsp(

index,

MMGSDI\_TP\_DOWNLOAD\_COMPLETE\_REQ,

mmgsdi\_status);

可看到函数获取了client request table中可用的索引，然后通过该索引对mmgsdi\_client\_req\_table\_info\_ptr进行数据的填充，并调用mmgsdi\_util\_queue\_mmgsdi\_uim\_report\_rsp构建对应的mmgsdi\_task\_cmd\_type发至comand quene(MMGSDI\_UIM\_REPORT\_RSP)

进入到mmgsdi\_process\_response，根据command类型为MMGSDI\_TP\_DOWNLOAD\_COMPLETE\_REQ调用mmgsdi\_process\_tp\_dl\_complete\_response。

该函数检测卡当前可用的卡类型,并根据卡profile发送card inserted事件。

## 3.3. 小结

Terminal Profile相关的协议文档只做了定义规范，具体实现需要查看相关的pics表。而高通文档对于该部分的流程介绍不多，这个模块的学习更多依赖的是modem代码的跟进。且该流程是在Modem端运作，有必要去了解Modem的架构与相关模块的作用。

# 4. Proactive Command

## 4.1. 概念

### 4.1.1. 定义

主动式命令是由主动式SIM卡发出，SIM卡通过主动式命令让UE端执行对应的操作，如插入联通卡后出现的弹框消息为主动式命令Display Text。

主动式命令有数十种，常见的如Setup Menu, Display Text。具体可查看3GPP TS 11.11。

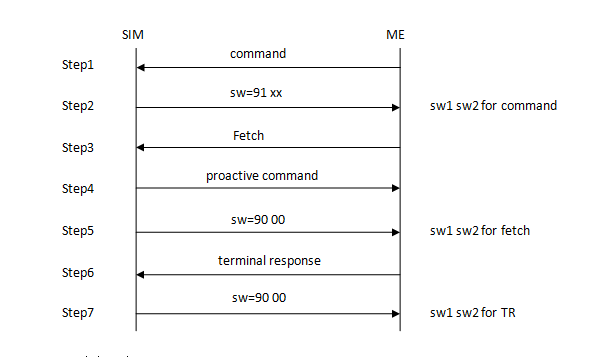
主动式SIM卡通过状态字SW1 SW2实现主动权，相关的状态字与解释如下：

|  |  |  |
| --- | --- | --- |
| SW1 | SW2 | 解释 |
| 90 | 00 | 正常的成功报告，SIM卡无其他SAT命令 |
| 91 | xx | 命令成功执行，有一个长为xx的proactive command |
| 93 | 00 | SIM卡正忙于执行前一个proactive command |
| 9E | xx | SIM数据下载时出错，有一个长为xx的回应 |

’90 00’表示命令正常结束。’91 XX’除了像’90 00’一样通知ME SIM卡成功执行完前一个命令，还包含了一个SIM卡的命令，告诉ME还需要执行一个动作。然后ME使用fetch命令来取得这个数据，XX表示相应数据的长度。

### 4.1.2. 流程

一个完整的Proactive Command包括Fetch, Proactive Command, Terminal Response以及状态字，具体的时序图如下：



step1-7称为一个sat session.

1．ME总是命令的发起者，首先ME发送SAT命令给SIM卡

2． SIM卡执行完命令后，返回Status Words（91XX），告诉ME有一个proactive command需要执行

3．ME用fetch命令取得proactive command

4．SIM将proactive command数据传送给ME

5．SW1 SW2 for Fetch

6．ME发送 Terminal Response给SIM卡

7．SW1 SW2 for Terminal Response

Step4-6是ME执行proactive command的时间：如显示数据，让用户输入，播放声音，发送短信等等。

下文将根据这个图分析Proactive Command的处理流程。

## 4.2. 关键流程分析

### 4.2.1 流程图

在高通项目中，一个主动式命令的主要流程如下图，自下向上依次涉及的模块有UIM、GSTK与MMGSDI、QMI\_CAT。



### 4.2.2. log分析

#### 关键log

以Benz手机为例，插入卡抓取set up menu的modem log，截取主要部分如下：

**Fetch下发：**

1.[uim.c 1795] UIM\_1: cmd status 0x0 SW1 0x91,SW2 0x66, Response data length 0x32

2.[uimgen.c 3453] UIM\_1: Reporting UIM generic command 0x104

3.[gstklib.c 3525] MMGSDI FETCH allow Status is 0x1

4.[gstklib.c 3528] Setting proactive\_cmd\_in\_progress flag to 0x1

5.[uimgen.c 2737] UIM\_1: Received Internal Fetch command 0x111

6.[uimdrv.c 2391] UIM\_1: uim\_send\_command

7.[gstklib.c 5368] gstk\_gsdi\_set\_fetch\_status: Slot = 0x1, status = 0x1

**获取主动式命令：**

8.[uim.c 1795] UIM\_1: cmd status 0x0 SW1 0x90,SW2 0x0, Response data length 0x1c

9.[uimgen.c 3453] UIM\_1: Reporting UIM generic command 0x111

10.[gstklib.c 2734] UI or NW proactive session in progress 0xff

11.[gstk\_s\_idle.c 686] Received Proactive cmd

12.[gstk\_proactive\_cmd.c 878] In gstk\_process\_proactive\_command() ui\_nw\_slot\_in\_progress = 0xff, cmd slot\_id 0x1

[gstk\_proactive\_cmd.c 927] Cache the proactive cmd

[gstk\_proactive\_cmd.c 598] COMMAND DETAILS CMD NUM: 0x1, TYPE OF CMD: 0x25, QUALIFIER: 0x0

13.[qmi\_cat.c 6204] qmi\_cat\_cache\_raw\_proactive\_command: command\_id=0x25, slot\_index=0x

[qmi\_cat.c 12608] Sending QMI CAT Indication to QMI Framework: clid:0x1, ind\_id:0x1

**发送Terminal Response:**

14. [uim.c 1795] UIM\_1: cmd status 0x0 SW1 0x90,SW2 0x0, Response data length 0x1c

15. [ gstk\_terminal\_rsp.c 2872] IN gstk\_send\_raw\_terminal\_response

16. [gstkutil.c 7722] 0x81 0x3 0x1 0x25 0x0 0x2 0x2 0x82 0x81

17. [uimgen.c 2750] UIM\_1: Received Terminal Response command 0x112

18. [uimdrv.c 2391] UIM\_1: uim\_send\_command

19. [uim.c 1795] UIM\_1: cmd status 0x0 SW1 0x90,SW2 0x0, Response data length 0x0

20. [uimgen.c 3453] UIM\_1: Reporting UIM generic command 0x112

#### 分析

根据上文所列序号对log挨个分析：

**Fetch下发：**

1. uim\_command\_response处理SIM卡上报的response,读取返回状态字为SW1SW2:9166，即上一条命令执行成功，有一条长度0x66的主动式命令。
2. 由uim\_generic\_command\_response确定返回的是命令0x104的执行结果。
3. uim.c调用uim\_process\_card\_response进行处理，通过uim\_command\_response解读response确定stauts为fetch，调用uim\_interface\_gstk\_uim\_fetch\_allow确定是否能发送fetch，uim\_interface\_gstk\_uim\_fetch\_allow实际调用了gstklib的gstk\_uim\_fetch\_allow：

case:UIM\_CMD\_FETCH

/\* Mimic the case of no-FETCH to UIM\_CMD\_COMPLETED case \*/

if(status == UIM\_CMD\_FETCH &&

!**uim\_interface\_gstk\_uim\_fetch\_allow**((gstk\_slot\_id\_enum\_type)cmd\_slot)) {

status = UIM\_CMD\_COMPLETED;

}

1. 进入gstklib的gstk\_uim\_fetch\_allow，根据注释解释该函数提供给UIM去确定能否发fetch，这里返回1，不改变状态值；继而调用GSTK\_SET\_PROACTIVE\_CMD\_IN\_PROGRESS\_FLAG设置标志表示主动式命令处理中。

/\* Mimic the case of no-FETCH to UIM\_CMD\_COMPLETED case \*/

if(status == UIM\_CMD\_FETCH &&

!uim\_interface\_gstk\_uim\_fetch\_allow((gstk\_slot\_id\_enum\_type)cmd\_slot))

{

status = UIM\_CMD\_COMPLETED;

}

1. 根据状态值仍为UIM\_CMD\_FETCH，确定执行fetch命令：

/\* else if a proactive command is pending \*/

else if (status == UIM\_CMD\_FETCH)

{

/\* Send a FETCH command to the SIM \*/

**uim\_fetch (i\_mask, uim\_ptr);**

}

1. 跟进uim\_fetch，最终调用了uimdrv.c的uim\_send\_command去发送fetch，即如上文所说通过UIM driver发送fetch命令：

case UIM\_FETCH\_ST: /\* Fetch State \*/

{

/\* Build a Fetch APDU \*/

uim\_req\_buf\_static\_ptr->apdu\_hdr.instrn = FETCH;

uim\_req\_buf\_static\_ptr->instrn\_case = UIM\_INSTRN\_CASE\_2;

/\* P1 = P2 = 0 and data field is empty in UICC implies that the

command applies on the current EF \*/

uim\_req\_buf\_static\_ptr->apdu\_hdr.p1 = 0;

uim\_req\_buf\_static\_ptr->apdu\_hdr.p2 = 0;

if (uim\_ptr->command.cmd\_ptr->hdr.protocol == UIM\_UICC)

{

uim\_req\_buf\_static\_ptr->apdu\_hdr.uim\_class = UIM\_UICC\_INSTRUCTION\_CLASS2;

}

else

{

uim\_req\_buf\_static\_ptr->apdu\_hdr.uim\_class = UIM\_INSTRUCTION\_CLASS;

}

/\* Number of bytes of Fetch command \*/

uim\_req\_buf\_static\_ptr->apdu\_hdr.p3 = uim\_ptr->command.cmd\_ptr->fetch.command\_size;

uim\_req\_buf\_static\_ptr->rsp\_ptr = &uim\_ptr->command.rsp\_buf;

uim\_send\_command(uim\_req\_buf\_static\_ptr, uim\_ptr);

} /\* end case - UIM\_FETCH\_ST \*/

break;

1. 由mmgsdi调用gstk\_gsdi\_set\_fetch\_status设置当前允许执行fetch命令，这里是由回调事件MMGSDI\_SESSION\_CHANGED\_EVT触发。

**获取主动式命令：**

1. uim\_command\_response处理SIM卡上报的response,读取返回状态字为9000，上一条命令执行成功。
2. 返回的是命令0x111的执行结果，根据uim\_cmd\_name\_type结构体的定义，0x111为UIM\_INTERNAL\_FETCH\_F，即fetch执行成功，开始主动式命令的处理。
3. 调用gstklib.c的gstk\_send\_proactive\_cmd将命令传至gstk。
4. gstk当前为空闲模式，调用gstk\_process\_proactive\_command开始处理主动式命令，
5. 进入gstk\_process\_proactive\_command ，确定是新的主动式命令，调用gstk\_preprocess\_proactive\_cmd进行解析，确定命令类型为0x25，即set up menu。

由gstk\_setup\_menu.c将命令发至客户端（qmi\_cat）：

/\* Send command to clients \*/

gstk\_status = gstk\_util\_send\_message\_to\_clients(&request);

1. 由qmi\_cat.将主动式命令发至上层，这里的Indication我理解为由SIM卡发起的命令，在RIL层由RadioIndication.java处理。

**发送Terminal Response:**

1. 主动式命令发送结束。

15-18. 发送Terminal Respone， 类型为0x25，即set up menu的Terminal Response。

19-20. Terminal Reponse发送完成，0x112即Terminal Reponse。

### 4.2.3. 时序图

下图为set up menu上报的过程（不包含发送TR）



## 4.3. 经验分享

Defect：7098243

Summary: [STK]STK cannot enter when first time hot insert the SIM card in to ME

问题分析：

1. 第一次开机后无法进入stk应用，这种情况首先可以确定是set up menu相关，通过logcat可查看RILJ以及应用层的log，发现没有任何主动式命令相关的log，确定问题出现在底层。
2. 接下来就可根据上文所提的关键log进行排查，通过QXDM发现modem端正常执行了fetch下发，获取主动式命令的过程。
3. 在qmi\_cat.c没有预见的log（第13点），而是出现了异常的log: [ qmi\_cat.c 7817] QMI-CAT: No client to send indication: command\_id=0x11。 对比正常机发现nv项65683值错误，正常值应为2：[ qmi\_cat.c 12425] qmi\_cati\_get\_cached\_proactive\_cmd: config\_mode=0x3即问题出现在nv65683（qmi\_cat\_mode）。

排错：

1. 跟进代码确定qmi\_cat\_mode在qmi初始化时通过qmi\_cati\_read\_configuration读值。
2. 查看配置文件/nv/item\_files/modem/qmi/cat/qmi\_cat\_mode发现对应值为2，确定是读值失败。
3. 查看qmi\_cati\_read\_configuration确定读值失败情况下返回0x03：

return CATI\_CONFIG\_MODE\_DECODED;该值为0x03

## 4.4. 小结

本文总结了Proactive Comment的知识点，通过协议文档简述其基础知识点，结合高通modem的代码与log介绍主动式命令的运作流程。

涉及主动式命令的问题大多与modem相关，了解主动式命令的处理过程与参与模块能保证及时高效的解决问题。

# Envelope Command

## 概念

### 简介

Envelope中文指信封，顾名思义，该命令使用来携带数据给SIM卡或者让SIM卡执行某些动作。与主动式命令相反，它是由UE端发起的命令；同时区别于Terminal Response，并不是某一条主动式命令的执行结果。Envelope涉及STK多个机制，如点击STK菜单条目进入下一级，下文将举例做简单的介绍。

### 应用机制

**Data Download:** 分为SMS-PP DATA DOWNLOAD与Cell Broadcast DATA DOWNLOAD，不管是哪一种，ME端在收到满足要求的message时都会将先其通过Envelope发给SIM卡处理，而不是显示给手机用户。

**Menu Selction:** SIM卡初始化时可通过set up menu提供多个菜单，当用户通过应用点击某一项或者求助选项，ME会通过Envelope将该点击事件发至SIM卡。该命令与主动式命令select item对应，SIM卡根据具体的选项发送二级菜单。

**Event Download：**这个涉及到主动式命令set up event list, SIM卡通过该命令让ME建立一个事件表，当对应的事件发生时，通过Envelope命令通知SIM卡。具体事件如WLAN Access status event，Location status event，Call disconnected event等。

**具体的其他机制可查阅3GPP TS31.111。**

### 命令结构

一条完整的Envelope命令应包含表中所列项（以Menu Selection为例，表格截至3GPP TS 11.14）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Description** | **Section** | **M/O** | **Min** | **Length** |
| Menu Selection tag | 13.1 | M | Y | 1 |
| Length (A+B+C) | - | M | Y | 1 or 2 |
| Device identities | 12.7 | M | Y | A |
| Item identifier | 12.10 | M | Y | B |
| Help request | 12.21 | O | N | C |

主要关注点是Menu Selection tag、Device identities、Item identifier：

其中第一位为Command tag，这里是**Menu Selection tag**，具体值 在13.1中定义，查阅确定值为D3。

**Device identities** 这个值记录的是命令由谁发起，发到哪，具体值如下：

Coding: both Source and Destination device identities are coded as follows:

- '01' = Keypad

- '02' = Display

- '03' = Earpiece

- '10' to '17' = Additional Card Reader x (0 to 7). Value assigned by ME.

- '21' to '27' = Channel x (1 to 7). Value assigned by ME (if class "e" is supported).

- '81' = SIM

- '82' = ME

- '83' = Network

根据定义，一个Device identities需包含Device identities tag，Length = '02'，Source device identity，Destination device identity共4位。

**Item identifier**指用户在STK应用中所点击的选项，是该命令要传输的值。这一值占3位，即tag，length，itentifier。

## Modem端实现

### 涉及模块



上图为Envelope主要涉及的模块，可以看到与Proactive Command经过相同模块的处理，即UIM,GSTK,QMI\_CAT。其中GSTK做STK的主要逻辑处理，其余两模块负责与上下层对接。

### 关键函数

下图为高通文档80-ng610-1\_k\_gstk\_overview 提供的Menu Selection的完整处理流程，从上层至SIM卡依次涉及的关键函数为：

**QMI\_CAT:**

1. gstk\_send\_raw\_envelope\_command:这里与图示函数有差异，Benz项目里以raw代指所有的命令类型，直接发至GSTK，不分命令类型做处理。
2. qmi\_cati\_gstk\_raw\_env\_cb env\_rsp\_ptr：回调函数，供GSTK发送Envelope Respnse。

**GSTK:**

1. gstk\_send\_cmd\_to\_uim\_server：发送命令至UIM。
2. gstk\_uim\_envelope\_report：回调函数，供UIM发送Envelope Response。

**UIM:**

1. uim\_send\_command:发送命令至SIM卡。
2. uim\_generic\_command\_response：处理SIM卡返回的两位状态字。



### log分析

#### 关键log

以Menu Selection为例，分析Benz的modem端log，这里只截取了关键的log。

**发送Envelope:**

1.[ qmi\_cat.c 8684] Handling cmd:0x22

2.[ gstklib.c 1931] GSTK client send raw envelope of len 0x9

3.[gstk.c 4552] Queuing cmd: 0x2 0x74

4.[gstk\_s\_idle.c 724] Received Envelope cmd

5.[ gstk\_envelope\_cmd.c 3989] gstk\_process\_envelope\_cmd: command = 0x74

6.[ gstk\_envelope\_cmd.c 4045] Rec'd Raw Envelope: GSTK\_RAW\_ENVELOPE\_IND

7.[gstkutil.c 7722] 0xd3 0x7 0x82 0x2 0x1 0x81 0x90 0x1 0x1

8.[uimgen.c 2722] UIM\_1: Received Envelope command 0x110

9.[uimdrv.c 2391] UIM\_1: uim\_send\_command

**处理Envelope Response:**

10.[uim.c 1796] UIM\_1: cmd status 0x0 SW1 0x91,SW2 0x83, Response data length 0x0

11.[gstk\_envelope\_rsp.c 297] GSTK recv UIM envelope rsp, 0x91, 0x83, Report Status 0x1

12.[gstklib.c 3528] Setting proactive\_cmd\_in\_progress flag to 0x1

13.[uimgen.c 2738] UIM\_1: Received Internal Fetch command 0x111

14.[uimdrv.c 2391] UIM\_1: uim\_send\_command

15.[gstk\_envelope\_rsp.c 814] IN GSTK\_RAW\_ENVELOPE\_RSP()

16.[qmi\_cat.c 6842] Sending response for raw env rsp

17.[qmi\_cat.c 12608] Sending QMI CAT Indication to QMI Framework: clid:0x2, ind\_id:0x22

**发送Proactive Command:**

18.[uim.c 1796] UIM\_1: cmd status 0x0 SW1 0x90,SW2 0x0, Response data length 0x83

19.[uimgen.c 8777] UIM\_1: Processsing uim\_generic\_command\_response for state 0x23

20.[gstk\_s\_idle.c 686] Received Proactive cmd

21.[gstk\_proactive\_cmd.c 598] COMMAND DETAILS CMD NUM: 0x1, TYPE OF CMD: 0x24, QUALIFIER: 0x0

22.[qmi\_cat.c 6204] qmi\_cat\_cache\_raw\_proactive\_command: command\_id=0x24, slot\_index=0x0, is\_recovery\_cache=0x1

23.[qmi\_cat.c 12608] Sending QMI CAT Indication to QMI Framework: clid:0x2, ind\_id:0x1

#### 分析

**发送Envelope:**

1. qmi\_cat调用qmi\_cat\_process\_framework\_cmd处理上层传来的命令，cmd = 0x22。查看结构体qmi\_cati\_cmd\_callbacks确定0x22为CATI\_CMD\_VAL\_SEND\_ENVELOPE\_CMD，调用qmi\_cati\_send\_envelope\_cmd处理消息。

static qmi\_svc\_cmd\_hdlr\_type qmi\_cati\_cmd\_callbacks[] =

{

......

CAT\_HDLR( **CATI\_CMD\_VAL\_SEND\_ENVELOPE\_CMD**, qmi\_cati\_send\_envelope\_cmd ),

......

/\* NOTE: When adding an item to this table, also add it to

qmi\_cat\_is\_command\_disabled() \*/

};

1. gstk\_send\_raw\_envelope\_command被qmi\_cati\_send\_envelope\_cmd调用，将命令传至gstk quene。
2. 确定当前处理命令为0x74，查看对应的结构体gstk\_cmd\_enum\_type确定0x74为GSTK\_RAW\_ENVELOPE\_IND，如上文所说，这里也不分具体的命令类型，作为raw envelope command统一处理。

typedef enum {

.......

GSTK\_RAW\_ENVELOPE\_IND = 0x74, /\*\*< Envelope command: Raw Envelope Command. \*/

.......

}

gstk\_cmd\_enum\_type;

1. gstk处于空闲模式，进入gstk\_s\_idle.c开始处理Envelope command。
2. 根据命令类型调用gstk\_envelope\_cmd.c的gstk\_process\_envelope\_cmd。
3. 打印命令具体数据：d3 07 82 02 01 81 90 01 01

根据2.3所提做命令的解析：

1. Menu Selection tag：d3
2. 长度：07，（Menu Selectiong tag 与长度除外）
3. Device identities：82 02 01 81

其中82为Device identitiies tag， 02 为长度，01 81指该命令由keypad发至SIM。

1. 收到gstk传来的命令，0x110即UIM\_ENVELOPE\_F。（uim\_cmd\_name\_type）

/\* Command types \*/

typedef enum {

......

UIM\_ENVELOPE\_F, /\* Send a toolkit command to the UIM \*/

......

} uim\_cmd\_name\_type;

1. 由uimdrv.c将命令发至SIM卡。

**处理Envelope Response:**

1. uim收到状态字9183，即有长度为83的主动式命令。
2. uim调用gstk\_uim\_envelope\_report开始处理response。
3. 准备处理主动式命令，设置proactive\_cmd\_in\_progress flag为1。
4. uim收到gstk发来的fetch请求，0x111即UIM\_INTERNAL\_FETCH\_F。
5. 执行fetch命令。

15-17.将Envelope response发至上层。

**发送Proactive Command:**

18-23．收到Proactive Command，发至上层。

在返回状态字9000后，modem开始处理主动式命令的流程，具体的主动式命令是Select Item，用于返回二级菜单。（0x25）

### 时序图

根据log分析，画出对应的时序图如下：



### 小结

1. Envelope在不同模块中具体的定义值不同，分别是

QMI:0x22（qmi\_cati\_cmd\_callbacks）;

GSTK:0x74（gstk\_cmd\_enum\_type）;

UIM:0x110（uim\_cmd\_name\_type）

1. SIM卡在收到Envelope命令后会返回状态字91XX，通知Modem下发fetch。
2. 与Proactive Command类似，以raw代替具体的命令类型，即不同的Envelope命令的处理过程大部分是相同的

Modem端主要作用是将该命令发至SIM卡。目前没接触过Envelope的相关问题，但从log分析可以确定涉及到Modem端的问题分析主要集中在命令是否收发成功、命令携带的信息是否正确。命令携带的信息需通过3GPP TS 11.14确定具体的内容。

# Bearer Independent Protocol

## 概念

### BIP的引入

首先介绍一下OTA，OTA(Over The Air)技术是一种基于短消息方式的空中下载技术。将OTA 技术应用于移动通信增值业务平台,用户可以随时随地的对手机STK(SIM Tool Kit,智能卡)卡内置菜单进行更新,实现对STK 卡菜单的个性化设置, SP(Service provider,服务提供商)可以方便低成本地进行新业务的快速推广,运营商也可以对SP 业务进行统一快捷的管理。

BIP即承载无关协议，它允许SIM卡通过UE与服务器建立数据传输通道（SIM卡不拥有IP地址），实现空中下载技术（OTA），与传统的SMS-PP(点对点短消息)承载方式不同，BIP是面向连接的，具有传输实时性，系统安全性以及数据完整性的优势。

同时，在SIM卡没有IP地址的情况下,BIP让智能卡实现智能卡服务器(SCWS)的功能。

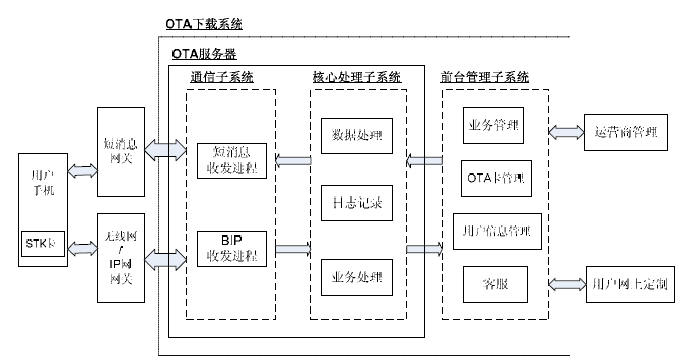
智能卡Web 服务器（SCWS）是一个在嵌入在移动设备中的智能卡（如，SIM

卡 (U)SIM、UICC、R-UIM、CSIM）中运行的HTTP 服务器。它能够提供动态和静态的WEB页面，让智能卡与手机终端中基于HTTP的应用（如WEB浏览器）进行交互。它允许网络运营商给他们的用户提供最新的智能卡业务。

### 引入BIP的OTA结构图

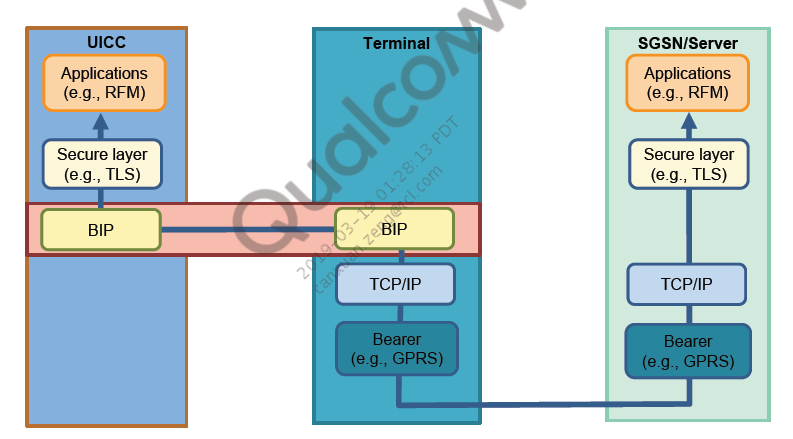
从例图可以看出，运营商可以通过两种方式向用户发布业务、应用等。即短消息方式

以及IP报文方式。



### BIP示意图

高通提供的示意图更能直观的了解BIP的运作方式：



### 使用的STK机制

**Proactive commands**

OPEN CHANNEL：要求ME打开指定的数据通道，通过命令指定具体的通道属性。

CLOSE CHANNEL：要求ME关闭指定标识符对应的数据通道。

SEND DATA：要求ME发送SIM卡提供的数据给指定的数据通道。

RECEIVE DATA：要求ME返回来自指定标识符对应的通道里的数据。

GET CHANNEL STATUS：该命令请求终端返回每个专用信道标识符所对应的信道状态。

**Envelope commands**

Event Download

Data available:如果新数据到达时，目标通道缓冲区时为空。

1）send data 执行后，且通道可用，返回该Envelope

Channel status:ME检测到通道发生错误，且不是由主动式命令导致的，通过该Envelope通知。

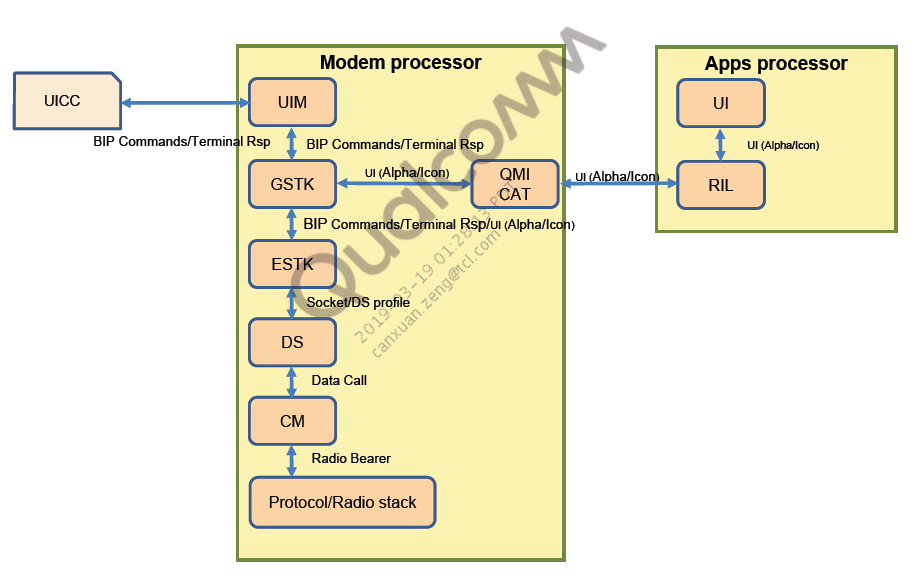
因此，在**Profile Download**时，SIM卡需要确定当前的ME是否支持以上功能。

## Modem端实现

### 6.2.1. 涉及模块

#### 模块图

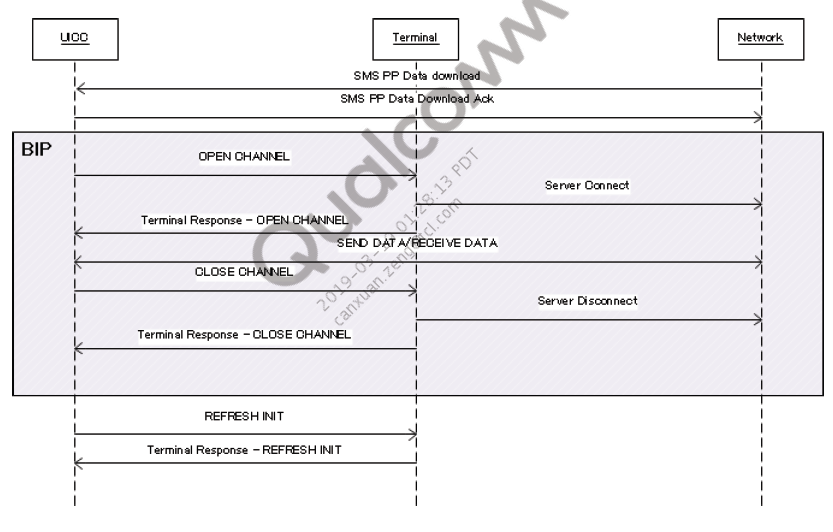
BIP相关的命令（Proactive Command与Envelope Command）在执行时涉及的模块图如下：



其中UIM, GSTK, QMI CAT模块在学习其他命令时有涉及。根据代码注释，ESTK为Enhanced STK layer，用于处理bip, refresh、setup menu、sms、ss等命令，也是有该模块调用DS的函数来建立数据通道（TCP/IP）的链接。

#### 关键流程

举个例子，一个BIP流程可以为：Open Channel, Send Data, Data Available, Receiver Data, Close Channel. 对应的用例图如下：



### log解析

这里以高通提供的log做简单的学习：

#### Open Channel

**//收到命令，进入GSTK开始解析，命令类型为0x40，即Open Channel**

00:00:54.093 gstk\_open\_ch.c389HIN GSTK\_OPEN\_CH\_REQ

00:00:54.093 gstk\_proactive\_cmd.c384HTLV\_length = 53

00:00:54.093 gstk\_proactive\_cmd.c538HCOMMAND DETAILS CMD NUM: 0x1

00:00:54.093 gstk\_proactive\_cmd.c540HCOMMAND DETAILS TYPE OF CMD: 0x40

00:00:54.093 gstk\_proactive\_cmd.c542HCOMMAND DETAILS QUALIFIER: 0x1

00:00:54.093 gstk\_proactive\_cmd.c673HREMAIN CMD LEN: 0x4a, offset: 0xb

00:00:54.093 gstk\_open\_ch.c439H.... Immediate link establishment

**//打印命令具体消息:** **4F 70 65 6E 20 43 68 61 6E 6E 65 6C 20 66 6F 72 20 55 49 43 43 3F 35 010339 02 05 78 47 09 08 76 7A 77 61 64 6D 69 6E 0D 08 F4 55 73 65 72 4C 6F 67 0D 08 F4 55 73 65 72 50 77 64 3C 03 02 AD 9C 3E 05 21 01 01 010190 00**

00:00:54.093 gstkparser.c7007H\*\* Parsing 0x5 into GSTK\_ALPHABET\_TYPE

00:00:54.093 gstkparser.c7154H============ Raw Data

.......

**//命令信息解码，消息为open channel for uicc ?**

00:00:54.094 gstkparser.c7158H============ Decoded alphabet

00:00:54.094 gstkparser.c7160Halpha[0]: 0x4f, O

00:00:54.094 gstkparser.c7160Halpha[1]: 0x70, p

00:00:54.094 gstkparser.c7160Halpha[2]: 0x65, e

00:00:54.094 gstkparser.c7160Halpha[3]: 0x6e, n

.......

**//开始解析执行命令所需的消息，承载类型、网络接入名称等。**

00:00:54.094 gstkparser.c6326H\*\* Parsing bearer description

00:00:54.094 gstkparser.c6429HBearer Type: DEFAULT

00:00:54.094 gstkparser.c6547H\*\* Parsing buffer size

00:00:54.094 gstkparser.c6621HRaw Data 0x5, 0x78

00:00:54.094 gstkparser.c6622HBuffer Size 0x578

00:00:54.094 gstkparser.c8600H\*\* Parsing Network access name into GSTK\_ALPHABET\_TYPE

00:00:54.094 gstkparser.c7463H\*\* Parsing other address

00:00:54.094 gstkparser.c885Esim tlv tag: 0xd != exp tag: 0x3e (M/O = 1)

00:00:54.094 gstkparser.c661HDone Parsing (O) TLV w/ Tag 0x3e

00:00:54.094 gstkparser.c8084H\*\* Parsing 0xd into GSTK\_ALPHABET\_TYPE

00:00:54.094 gstkparser.c1129HIn GSTK\_DECODE\_TEXT\_STRING\_BODY\_EXT

.......

00:00:54.095 estk\_bip.c4210Hestk\_process\_open\_ch\_req

00:00:54.095 estk\_bip.c4324HALPHA TLV present in OPEN Channel command

00:00:54.095 estk.c1766HIn estk\_send\_alpha\_to\_display\_client()

00:00:56.310 estk.c1158HIn estk\_process\_cmd()

00:00:56.310 estk.c354HSwitch to ESTK instance 0x0.

00:00:56.310 estk\_bip.c4051HIn estk\_process\_bip\_user\_cnf\_rsp()

00:00:56.310 estk\_bip.c4083HIcon disp is 0x0

00:05:00.468 estk\_bip.c4568HUser accepted open channel request

**//开始处理命令：**

00:05:00.469 estk\_bip.c4191HAPN present in open channel command

00:05:00.469 estk\_bip.c4201HAPN length = 9

00:05:00.469 estk\_bip.c4214HAPN[0] = 0x54

00:05:00.469 estk\_bip.c4214HAPN[1] = 0x65

00:05:00.469 estk\_bip.c4214HAPN[2] = 0x73

00:05:00.469 estk\_bip.c4214HAPN[3] = 0x74

00:05:00.469 estk\_bip.c4214HAPN[4] = 0x47

00:05:00.469 estk\_bip.c4214HAPN[5] = 0x70

00:05:00.469 estk\_bip.c4214HAPN[6] = 0x2e

00:05:00.469 estk\_bip.c4214HAPN[7] = 0x72

00:05:00.469 estk\_bip.c4214HAPN[8] = 0x73

00:05:00.469 estk\_bip.c4224HSave login name

00:05:00.469 estk\_bip.c4240HUserLogin[2] = 0x65

00:05:00.469 estk\_bip.c4240HUserLogin[3] = 0x72

00:05:00.469 estk\_bip.c4240HUserLogin[4] = 0x4c

00:05:00.469 estk\_bip.c4240HUserLogin[5] = 0x6f

00:05:00.469 estk\_bip.c4240HUserLogin[6] = 0x67

00:05:00.469 estk\_bip.c4255HSave password

**// IP地址**

00:05:00.469 estk\_bip.c624H======== IPv4 Addr========

00:05:00.469 estk\_bip.c692HIP Addr[0] = 0x31(1)

00:05:00.469 estk\_bip.c692HIP Addr[1] = 0x2e(.)

00:05:00.469 estk\_bip.c692HIP Addr[2] = 0x31(1)

00:05:00.469 estk\_bip.c692HIP Addr[3] = 0x2e(.)

00:05:00.469 estk\_bip.c692HIP Addr[4] = 0x31(1)

00:05:00.470 estk\_bip.c692HIP Addr[5] = 0x2e(.)

00:05:00.471 estk\_bip.c3102HDid not find a re-usable profile

00:05:00.473 estk\_bip.c3182HESTK\_BIP: Profile number created 2

00:05:00.475 estk\_bip.c3534HSet profile PDP type to IPV4V6

00:05:00.475 estk\_bip.c3538HGSTK\_CFG\_FEATURE\_ESTK\_BIP\_IP\_DEFAULT NV set -PDP type IP

00:05:00.476 estk\_bip.c2851HGSTK\_CFG\_FEATURE\_ESTK\_BIP\_IP\_DEFAULT NV set -set PDN type v4

00:05:00.480 estk\_bip.c3744Hset profile num0x6b to net policy!

**//通道建立成功，返回TR：**

00:05:00.516 estk\_bip.c4317HBIP link successfully established

00:05:00.621 estk\_bip.c6044HBIP session in progress is 1

00:05:00.640 estk\_bip.c1902Hestk\_bip\_network\_cbwith error\_num200

00:05:00.640 estk\_bip.c1921HNetwork CB got DS\_ENETISCONN event

00:05:00.640 estk\_bip.c2499Hestk\_perform\_net\_connect

00:05:00.645 estk\_bip.c2539HGet updated QOS parameters from DS for P

00:05:00.649 estk\_bip.c2425HIn estk\_bip\_copy\_pdn\_qos\_parameters

00:05:01.213 estk\_bip.c2622Hdss\_connectsucceeded for TCP

00:05:01.213 estk\_bip.c2662Hdss\_connectsuccess 0

00:05:01.214 gstklib.c1155HClient send terminal rsp, ref\_id:0x0, command #:0x1, enum:0x58

00:05:01.214 gstk\_terminal\_rsp.c1801HOPEN CHANNEL rsp, result=0x0 additional info len=0x0

#### Send Data

只有在Open Channel成功执行后，SIM卡才会发出该命令。

**//命令类型0x43：Send Data**

00:01:01.863 gstk\_send\_data.c278 HIN GSTK\_SEND\_DATA\_REQ

00:01:01.863 gstk\_proactive\_cmd.c384 HTLV\_length = 13

00:01:01.863 gstk\_proactive\_cmd.c538 HCOMMAND DETAILS CMD NUM: 0x1

00:01:01.863 gstk\_proactive\_cmd.c540 HCOMMAND DETAILS TYPE OF CMD: 0x43

00:01:01.863 gstk\_proactive\_cmd.c542 HCOMMAND DETAILS QUALIFIER: 0x1

00:01:01.863 gstk\_proactive\_cmd.c673 HREMAIN CMD LEN: 0xa, offset: 0xb

00:01:01.863 gstk\_send\_data.c355 H.... Send Data Immediately

**//打印发送的数据**

00:01:01.864 estk\_bip.c879 Hestk\_bip\_write\_sock called

00:01:01.864 estk\_bip.c900 HSend Data buf 0 is 0x0

00:01:01.864 estk\_bip.c900 HSend Data buf 1 is 0x1

00:01:01.864 estk\_bip.c900 HSend Data buf 2 is 0x2

00:01:01.864 estk\_bip.c900 HSend Data buf 3 is 0x3

00:01:01.864 estk\_bip.c900 HSend Data buf 4 is 0x4

00:01:01.864 estk\_bip.c900 HSend Data buf 5 is 0x5

00:01:01.864 estk\_bip.c900 HSend Data buf 6 is 0x6

00:01:01.864 estk\_bip.c900 HSend Data buf 7 is 0x7

**//TR**

00:01:01.870 gstk\_terminal\_rsp.c 2008 HSEND DATA rsp, result=0x0 additional info len=0x0

#### Data Available

当有新的数据时（send data），如果目标通道缓存为空，ME会通过Envelope Command 向SIM卡发Data Available的消息。

具体的消息解析结果为：

\*\*\*\*APDU Parsing\*\*\*\*

ENVELOPE

Logical Channel:0

UICC instruction class CLA-No SM used between terminal and

Event List :**Data available**

Source ID :terminal

Destination ID :SIM

Channel Status :Channel Identifier 1

For CS, PDP, local & Default bearer:Link established or PDP context activated

For UICC Server Mode :TCP in ESTABLISHED state

For Term. Server Mode & TCP:TCP in ESTABLISHED state

Channel Data Length :8

Status Words-0x90 0x00 -Normal ending of the command

#### Receive Data

在SIM卡收到Data Available的消息后，上报该命令让ME开始接收数据。

具体命令解析，接收通道标识符指定的数据：

00:02:26.981 [00] 0x1098 RUIM Debug

RX 12 D0 0C 81 03 01 42 00 82 02 81 21 B7 01 C8 90 00

\*\*\*\*APDU Parsing\*\*\*\*

FETCH

ICC instruction class CLA -No SM used between terminal and card

Command Number :1

Command Type :RECEIVE DATA

Source ID :SIM

Destination ID :Channel 1

Channel Data Length :200

Status Words -0x90 0x00 -Normal ending of the command

#### Close Channel

RX 12 D0 09 81 03 01 41 00 82 02 81 21 90 00

\*\*\*\*APDU Parsing\*\*\*\*

FETCH

Logical Channel:0

UICC instruction class CLA-No SM used between terminal and card

Command Number :1

Command Type :**CLOSE CHANNEL**

Source ID :SIM

Destination ID :**Channel 1**

Status Words-0x90 0x00 -Normal ending of the command

**//收到命令，关闭指定的通道**

01:18:32.410 estk\_bip.c 5030 H Found channel to close

01:18:32.410 estk\_bip.c 5065 H De-register for Link down

01:18:32.410 estk\_bip.c 1671 H estk\_bip\_reg\_ds\_iface\_eventsuccessful

01:18:32.410 estk\_bip.c 1174 H estk\_bip\_process\_sock\_close\_evt

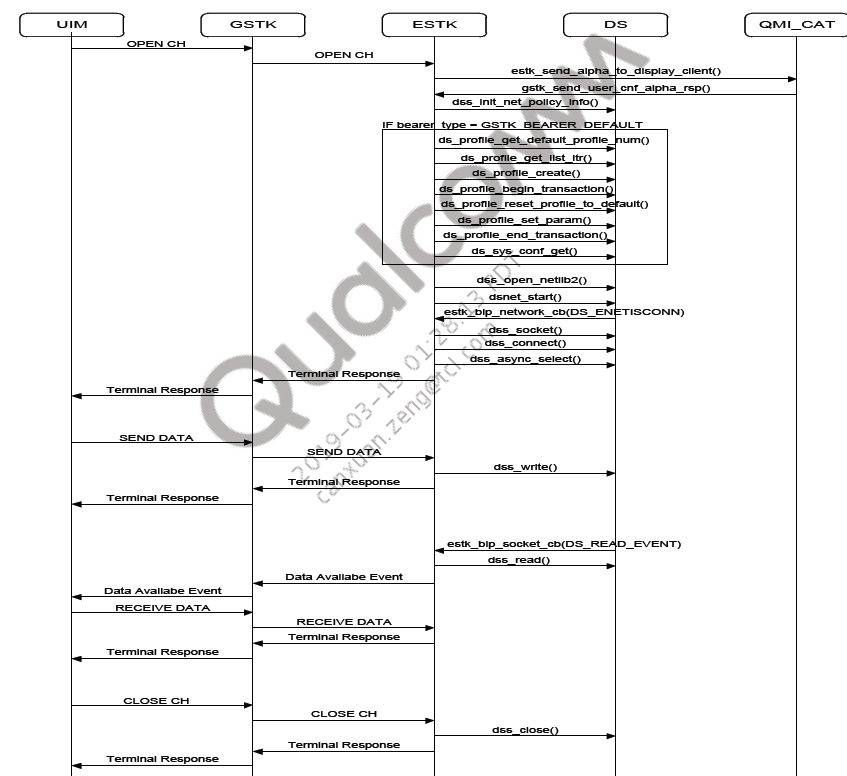
01:18:32.416 estk\_bip.c 1281 H dsnet\_stop succeeded

01:18:32.419 estk\_bip.c 1315 H dsnet\_release\_handle returns 0 with error\_num 0

01:18:32.419 estk\_bip.c 1316 H estk\_bip\_process\_sock\_close\_evt sending TR

01:18:32.439 gstk\_terminal\_rsp.c 1852 H CLOSE CHANNEL rsp, result=0x0 additional info len=0x0

### 时序图



## 小结

BIP这一模块相关资料较少，且没有接触过相关的问题，只能通过高通的文档进行log的学习分析。

BIP涉及Proactive Command与Envelope Command，掌握这两种命令的知识能更好的理解BIP的运作方式。

# 总结

以上便是STK几个主要机制的基本概念以及高通上的大致流程，这几个机制的相关性强，都会涉及到其他的模块。