

**MTK架构分析**

[讲师手册/学员手册]

康佳集团股份有限公司

[第一章 Framework介绍 3](#_Toc96766744)

[1.1启动流程 3](#_Toc96766745)

[1.1.1模拟器线程创建 3](#_Toc96766746)

[1.1.2消息循环 3](#_Toc96766747)

[1.1.3启动初始化 3](#_Toc96766748)

[1.1.4进入主屏幕 3](#_Toc96766749)

[1.2Queue相关 3](#_Toc96766750)

[1.3Event相关 3](#_Toc96766751)

[1.3.1Event简述 3](#_Toc96766752)

[1.3.2协议栈和硬件中断event 3](#_Toc96766753)

[1.3.3菜单高亮和提示event 3](#_Toc96766754)

[1.4History相关 3](#_Toc96766755)

[第二章 按键相关 3](#_Toc96766756)

[2.1键盘响应 3](#_Toc96766757)

[2.1.1得到键盘消息 3](#_Toc96766758)

[2.1.2消息进行转化后传给MMI 3](#_Toc96766759)

[2.1.3执行一些与按键有关的操作 3](#_Toc96766760)

[2.1.4真正执行按键所对应的操作 3](#_Toc96766761)

[2.2键盘的设置 3](#_Toc96766762)

[2.2.1普通按键的设置 3](#_Toc96766763)

[2.2.2软键盘的设置 3](#_Toc96766764)

[第三章 显示相关 3](#_Toc96766765)

[3.1普通菜单相关 3](#_Toc96766766)

[3.2编辑控件相关 3](#_Toc96766767)

[第四章 举例 3](#_Toc96766768)

[4.1一些模块的重点 3](#_Toc96766769)

[4.1.1闹钟模块 3](#_Toc96766770)

[4.1.2通话模块 3](#_Toc96766771)

[4.2CALLLOG模块分析报告 3](#_Toc96766772)

[第五章 资源的添加 3](#_Toc96766773)

[5.1总述 3](#_Toc96766774)

[5.2字符串资源的添加 3](#_Toc96766775)

[5.2.1 字符串资源添加过程 3](#_Toc96766776)

[5.2.2 字符串资源添加方法 3](#_Toc96766777)

[5.3图片资源的添加 3](#_Toc96766778)

[5.4菜单资源的添加 3](#_Toc96766779)

# 第一章 Framework介绍

## 1.1启动流程

### 1.1.1模拟器线程创建

由于代码是在模拟器中执行，所以代码中都带有一些win32程序的特征。比如模拟器的启动就是用典型的win32程序方式。在PixtelMMI.cpp文件里的函数WinMain就是模拟器的启动入口。里面包括了InitInstance和模拟器的消息循环，这些都是win32程序通用的，我们需要注意的只是InitApplication();

这个InitApplication();函数创建了几个线程，用这几个线程来模拟手机中的不同的任务线程。并且为这些任务创建了不同的消息队列。这些创建出来的线程的一切相关资料都保存在task\_info\_g1（一个结构数组）里面。包括消息队列的信息。此外，它还设置了一个timer,这个timer是模拟器用来模拟整个时钟震荡用的，以后需要使用timer的时候都是使用它的分频，它的周期

是100ms，不过这只是系统模拟的而已。这些创建的线程中其中有一个是主要的，此线程的入口函数是MMI\_task，它在MMITask.c里面。

### 1.1.2消息循环

函数MMI\_task主要完成的是不停的读取protocol task中的消息，然后根据消息和参数进行相应的处理函数。protocol task中的消息其实就是我们刚才在initAppication()里面创建线程时创建的消息队列，这个消息队列的的指针保存在一个消息队列id指针里面，而这个消息队列id指针则保存在**task\_info\_g[i]**结构的task\_ext\_qid项中。晓得了消息队列的所在，读取protocol task就变的可行了，这方面的相关操作我们将在Queue相关里面详细介绍。

在函数MMI\_task里读取了消息后，用一个switch语句来选择处理方式，我们发现很多处理方式的最后都调用了一个ProtocolEventHandler函数，此函数是得到消息中保存对应消息处理函数的函数入口，并根据入口执行相应函数，此方面的东西我们将在event相关里面进行详细介绍。最后，我们来看看当手机系统正常启动时执行的操作，看到case MSG\_ID\_MMI\_EQ\_POWER\_ON\_IND:消息，这个消息就是系统启动的消息。当系统收到这个消息时候，系统根据当前系统的状态来决定启动的方式，有正常启动（用户按键盘后启动），有充电启动，还有闹钟闹时启动，有异常启动。这里所谓的启动，有时候仅仅是显示一个用于表示状态的屏幕，而不全是完整的启动整个系统。

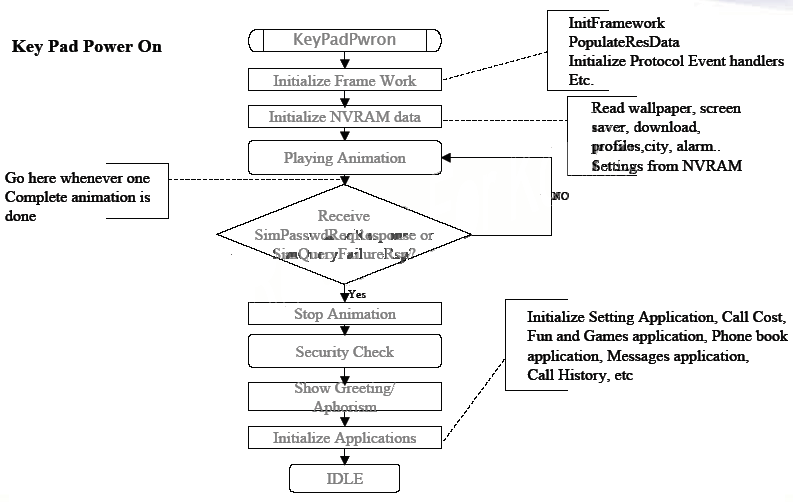
### 1.1.3启动初始化

我们看到正常启动（用户按键盘启动）中首先调用了函数InitializeAll，这个是系统启动的初始化函数。比如协议栈，菜单数组，中断队列的初始化，一些硬件消息所对应的事件设定，还比如一些需要立刻初始化的模块，设定这些模块的菜单相应事件，设定这些模块协议栈消息响应函数。

初始化这些后，接着函数InitNvramData，让一些需要读nvram的模块读出nvram中的数据，以此来初始化这些需要用到nvram中的数据来初始化的模块。

初始化后调用函数fast\_openscreen，这个函数就是系统启动后进入主屏幕的函数。比较复杂。

下面有一幅图，在图中可以很清楚的看清从按键盘启动后到进入主屏幕所要完成的一系列工作：



### 1.1.4进入主屏幕

在fast\_openscreen里开始就播放开机动画及声音，点亮显示屏，并且设置好当开机画面播放完后需要执行的回调函数CallBackPowerOnAnimationComplete。当动画播放结束就直接调用此回调函数。在此回调函数里面检测电池电量，如果电量不购开机就退出，否则则检测sim卡和是否需要密码？这2个状态是由这两个全局变量gSimQueryFailIndFlag，gPasswdReqIndFlag来决定的，由于使用的是回调的方式来调用CallBackPowerOnAnimationComplete，所以并不阻止消息的循环继续进行，所以这两个参数会因为得到了硬件发来的特定消息而改变其值，从而得到当前的状态。如果这两个条件都满足了，则进入EntryIdleScreen函数，这个函数就是程序的初始界面。并且在里面把左软键设置为进入主菜单，右软件设置为进入电话本。这两个入口分别是EntryMainMenuFromIdleScreen，EnterPhbFromIdleScreen。

到这里，一个启动过程总算结束了，以后的事情都交给注册好的事件响应函数和菜单响应函数去做。

## 1.2Queue相关

在我们刚才的讨论中，我们提到了在InitApplication（）时，我们创建不同的线程，同时创建不同的线程所对应的消息队列，并且把对应的消息队列的id指针保存在task\_info\_g1数组里。要说到消息队列，我们还得从全局数组**task\_info\_g1**说起。它是一个osl\_task\_info类型的数组，此类型有个成员变量task\_ext\_qid，它是oslMsgqid类型的。就是这个变量保存着消息队列指针。

对此队列，具有创建，删除，向里面写消息，从里面读消息等操作，其中最主要的就是发消息函数OslIntMsgSendExtQueue。这个就是向协议栈写入消息的函数，其他的几个操作都是模拟出来的，对真实的协议栈没影响。这个函数向协议栈写入一个MYQUEUE类型的消息，这个消息是由用户自己定义的。

这些操作都可以不管，我们只需要知道相应的接口就可以，那就是从队列中收消息OslReceiveMsgExtQ和向队列发消息OslMsgSendExtQueue，而消息是一个MYQUEUE结构类型，它包含发出消息者和消息要到达的地方。一般来讲消息发起者和消息要到达的地方是FRAMEWORK或者PROTOCO STACK/L4，前者发出消息让硬件执行某些操作，比如铃声响。而后者的消息是硬件发出来的，需要软件来进行响应，比如按键按后需要执行的操作。此外，MYQUEUE结构类型除了包含发起和接收者外，还包含消息类型号，用于指示消息的用途的，还有一个oslDataPtr指针，它是一些数据块的指针，指向响应函数需要用到的数据。

下图是关于协议栈和MMI间通过2条队列来传送消息的示意图



## 1.3Event相关

### 1.3.1Event简述

手机程序中的事件和PC程序中的事件一样，都是一些对消息的响应过程。这里要说到的event并不是具体的某个程序对某个消息的响应，而是对消息响应的通用过程，是对事件进行管理的过程。

按照event对消息响应的不同，我们可以把它分为两大类：一类是对协议栈和硬件中断消息的响应，一类是对菜单高亮时和显示提示时进行消息响应。我们看这两类消息的来源，我们也可以说这两类事件分别响应的是硬件消息和软件消息。其中硬件消息范围很广，包括键盘按键消息，来电消息，短信消息等等。

下面将分别就这两类事件进行详细说明。

### 1.3.2协议栈和硬件中断event

这类事件的管理主要是对两个结构数组的管理，也可以认为是两条链。它们的结构类型都只是包含了两个成员变量，一个是ID,一个是入口函数指针。这两个数组的名称分别是protocolEventHandler和interruptEventHandler。对硬件消息的响应就是靠对这两个数组的管理来实现的。

首先看到函数void InitFramework(void)，在里面系统把这两个数组都清空了，初始化了这两个数组。在这里我们还可以看到两个对管理这两个数组有用的两个全局变量maxProtocolEvent和maxInterruptEvent，分别表示数组中有多少个元素。由于这两种数组的管理其实都差不多，所以我们将只对其中的protocolEventHandler进行讨论。

在很多时候给硬件发消息，不过在发消息前往往常常会用到这个函数SetProtocolEventHandler。这个函数的作用是设置一个函数对某个硬件消息进行响应。用在给硬件发消息前通常的作用就是设置好对硬件回应消息的处理。我们看其实现过程，其实很简单，无非就是看数组中是否有相应的消息序号，如果没有则把消息序号和响应函数指针设置进数组。

这是其设置方式，设置好了后我们来看看这些消息响应事件到底是怎么触发的。说到其触发方式，我们可以联想到在系统启动时，系统的消息循环中不断的得到消息，并根据不同的消息对其进行不同的处理。我们可以从那里找到其触发方式。我们看到那里调用了函数void ProtocolEventHandler(U16 eventID,void\* MsgStruct)来响应不同的消息，而其中主要又是调用了ExecuteCurrProtocolHandler函数，可以知道ExecuteCurrProtocolHandler函数完成了其响应过程。当有消息到时，有时还会有对应消息的一些数据MsgStruct，ExecuteCurrProtocolHandler函数根据消息ID在两个数组中检索相应的响应函数指针，当检索得到函数指针后把参数MsgStruct传给函数指针让其去执行，这样就完成了对消息的响应过程。

还有响应函数的清除，ClearProtocolEventHandler，其实就是把对应消息的响应函数指针设置为空。等等一些操作都是在对这俩个数组进行操作。

### 1.3.3菜单高亮和提示event

对菜单高亮和显示提示的消息响应的管理也是通过一个数组来完成的。这个数组是maxHiliteInfo，它是一个hiliteInfo结构类型，其中只包含两个响应函数入口地址，一个是菜单高亮显示时需要转到的入口，一个是提示出现时需要转入的入口。它和protocolEventHandler不同，它的结构里面不包含索引用的ID，而是以它的下标作为索引，而且都是以菜单资源的ID号来做下标。索引只有找到了菜单的资源ID号就可以找到其相应的响应高亮显示和提示的函数入口，菜单ID和响应函数通过这种方式对映起来。

首先也是它的初始化，在InitFramework里把这个数组清空。我们常常在程序中可以看到用SetHiliteHandler函数把菜单和菜单高亮显示执行的函数联系起来，这个函数就是这个作用，把自己定义的函数的地址赋值给以菜单ID为下标的项中的高亮显示入口地址。这样就完成了它们的联系。

再来看当高亮显示后需要执行相应的函数的过程。晓得一个菜单ID要执行其相应的高亮显示函数很简单。void ExecuteCurrHiliteHandler(S32 hiliteid)函数完成了执行菜单高亮显示后的操作。不过，其参数不时菜单ID，而是当前菜单所在的兄弟菜单中的位置。通过这个ID，再晓得当前菜单的父菜单ID就可以找出当前菜单的ID,这些都不是问题，当前菜单的父菜单ID也有专门的变量来保存。所以这个函数的实现也很简单。

不过，一般这个函数常常会和另外一个函数联合起来执行，这个函数是RegisterHighlightHandler，这个函数涉及到GUI方面的东西，不做深入讨论，只要知道其作用效果就可以，它把当前高亮显示菜单所在的兄弟菜单中的位置传递给ExecuteCurrHiliteHandler，并且执行。

其它的操作很简单了，比如清除对应的函数地址只需要把它设置为NULL就可以。

## 1.4History相关

History保存访问过的页面的信息。它保存的信息包括：访问过的屏幕的id，访问过屏幕的入口函数地址，访问过的屏幕的GUIBuffer，访问过的屏幕如果有输入框的话，还要保存输入框inputBuffer大小及内容。

历史记录主要是靠数组historyData来管理的，它的类型是结构historyNode，这个结构包含了4个参数，屏幕ID，入口函数地址，GUIBuffer指针和InputBuffer指针。

一般将要离开一个SCREEN的时候需要用到添加历史消息操作。这里再插一些关于离开一个SCREEN的操作。在当前screen的对应入口函数的最后设置好当离开此屏幕时需要执行的函数，用函数SetExitHandler来设置，这个函数其实是把当前屏幕和它的离开函数保存在两个全局变量中，当进入一个新屏幕，其入口函数一开始就会调用ExecuteCurrExitHandler函数，实际上就是执行了全局变量中的离开函数。这个离开函数一般是用来保存历史记录的，以便于执行返回时跳回到访问过的屏幕。

在离开函数中添加历史记录常用函数是AddHistory，即AddHistoryReference函数，它把screenID，入口函数，GUIbuffer和Inputbuffer都保存起来，其中后2个是用动态生成内存的方式保存起来，不过像inputbuffer的保存一般都不是用这个函数，而是用AddNHistory函数，这个函数除了需要一个history结构的参数外还需要一个用来表示Inputbuffer大小的参数，inputbuffer大小和inputbuffer的内容联合在一起保存在它生成的动态内存中，保存它的地址在guiBuffer项中。

在历史记录中还有个关键的东西，用来指示出当前最近的历史记录所在数组的下标。其实就是数组使用到的地方currHistoryIndex。这个全局变量起到很关键作用，在添加历史记录的时候这个全局变量+1。当程序要回到上一个屏幕时只要调用GoBackHistory，它调用ExecutePopHistory函数，此函数就执行把历史记录的currHistoryIndex-1，并且把保存的历史屏幕的入口函数再执行一遍。再次执行入口函数时怕重复写历史记录，所以需要一个IsBackHistory参数来指示是否是返回，如果是返回历史记录，则不需要写入历史记录了。

还有GoBackToHistory函数，它的参数是屏幕ScreenID，根据这个去整个数组中搜，搜到了有和这个屏幕ID对应的ID在的话，就转到那个入口函数中去。还有些函数都比较有用，不一一说明了。

总之历史记录只是对一个数组的操作及维护。很多操作可以自己想出来应该怎么做。

# 按键相关

## 2.1键盘响应

### 2.1.1得到键盘消息

当用户单击键盘上任意一个键时，都会触发到一个硬件中断，然后L4将会把这个中断通过消息的方式发给Framework，Framework将要根据消息对其进行响应，我们这里主要就是讨论其响应过程。

当硬件发消息MSG\_ID\_MMI\_EQ\_KEYPAD\_DETECT\_IND时，说明就是键盘有键被按，这时候将触发void L4KeyHandle(void \*paraBuff)事件，在这个函数里对这个消息进行处理，我对那个消息的具体东西（paraBuff）不熟悉，只能凭借它后来对其的处理来进行猜测，程序把此指针转化为一个结构指针mmi\_eq\_keypad\_detect\_ind\_struct，这个结构里面有一个函数指针和两个其他的数据，根据他后面的使用情况，我猜测此函数指针是对一个按键消息缓冲区进行读操作，并且把读出来的数据放在一个类型为kbd\_data的变量里面。此结构包含2个变量，一个是用来表示按键的状态（是按住状态还是释放状态），一个用来表示按的是什么键。根据读出来的数据，进入ProcessKeyEvent来对读出的键盘状态进行处理。

### 2.1.2消息进行转化后传给MMI

在void ProcessKeyEvent(U32 MsgType, U16 KeyMapIndex)中有2个参数，第一个参数是按键消息的类型（up,down,longpress），第二个是按的键的索引。通过这个索引我们可以在一个全局数组中找到对应的按键的相关信息。这个全局数组是一个KeyPadMap类型，它里面包含了一些按键的状态信息，用来管理整个键盘的按键操作。在里面我们可以看到它对在模拟器上运行和直接在机器上运行进行了分开处理。在模拟器上运行还需要另外一个时钟函数来模拟键盘的长按，而直接在机器上则不用这样，而是直接的进行处理，其中多了一个DRV\_WM\_KEYLONGPRESS状态，个人感觉是硬件传递消息的时候直接就可以传送这个状态消息。把这些状态转化成MMI熟悉的状态后传给函数KeyEventHandler((struct KEYBRD\_MESSAGE \*)&KeyBrdMsg);去执行。

### 2.1.3执行一些与按键有关的操作

在KeyEventHandler中，系统先判断声音是否关闭，键盘是否上锁，判断好了后再根据判断进行一些比如播放不同音调的动作，这些不同的声音都保存在一个KeyToneMap的数组中，这个数组也是通过按键的不同nKeyCode进行索引，到最后这些附加的键盘响应动作结束后则进入函数ExecuteCurrKeyHandler来对真正的键盘对应的不同的事件，也是传两个参数，eventKey->nKeyCode和 eventKey->nMsgType。

### 2.1.4真正执行按键所对应的操作

在函数ExecuteCurrKeyHandler里，我们可以发现真正起主要作用的一个全局数组currKeyFuncPtrs，而其类型是函数指针。它有两维，而且分别就是按键状态和按键类型，就是说用传入的参数来作为其索引，可以找到这个按键处于这个状态的时候需要执行的事件。

说到这里，一个键盘被按后执行的操作大概就清楚了，可是最后说到的那个数组currKeyFuncPtrs，它的用处很，有一些函数专门对其进行维护。下面要讲到的软件盘的使用也与它有关。

## 2.2键盘的设置

### 2.2.1普通按键的设置

上面说到了currKeyFuncPtrs，我们说它用处很大，凡是需要有按键的地方，我们都要用到函数SetKeyHandler，这个函数就是用来设置一个用户定义的按键响应函数。我们在里面可以发现，其中只是对currKeyFuncPtrs中给定按键状态和按键类型的项进行赋值操作。把需要响应的函数地址赋给对应的项。

除了SetKeyHandler还有GetKeyHandler，只是把对应的项的值返回来，返回一个函数地址，还有SetGroupKeyHandler，也只是用一个循环来把一些项赋同一个函数地址指针。还有ClearKeyHandler，把对应的项改成NULL。

### 2.2.2软键盘的设置

有了上面对普通键盘设置的说明，对软键盘的设置就比较简单了。软键盘的设置其实就是对键盘上的KEY\_LSK和KEY\_RSK按键进行特殊的按键响应事件设置。包括了左软键和右软键的设置。我们只以左软键作为例子来进行分析说明，右软键基本和左软键处理方式一样。

左软键的设置，使用函数SetLeftSoftkeyFunction，它包含有两个参数，一个是左软键响应的函数地址指针，一个是对左软件的哪个状态进行设置响应事件。我们看到函数内部。它里面包含了两个函数。这两个函数过程都很简单。读完后我们发现其中左软键和一般的按键消息响应其实也没太大区别，只是它在响应时同时对屏幕上的按钮进行了一些操作。还有就是它用到了一个全局数组变量left\_softkey\_functions。用来记录按下或弹起对应的事件。在按钮对应的事件响应中执行left\_softkey\_functions保存的函数。

# 显示相关

## 3.1普通菜单相关

这个平台的framework中已经订制好了大量的通用的菜单资源，也可以叫做Category。用这些Category我们可以很简单的搭建好自己需要的程序框架，剩下的工作就是把这些框架填满，写这些框架的菜单响应事件，正是因为有了这些Category，使我们节省了很多与GUI打交道的时间。下面将用一个系统主菜单来演示使用这些Category的一般过程。

在第一章说到了进入主屏幕IdleScreen，进入后，设置好主菜单入口为左软键盘。当点击左软件就进入到主菜单goto\_main\_menu。我们现在就从这里开始讨论，具有代表性，大多数的菜单都是这样按这个流程来的。

首先是执行ExecuteCurrExitHandler，这个函数我们讨论过了，执行上个屏幕的结束函数，一般是把上个屏幕保存到历史记录中，也可以自己定义一些需要处理的工作。

然后需要注意一个byte \*history\_buffer;，一个指针，虽然在别的菜单屏幕中会用不同的名字，像gui\_buffer，但它们的作用都一样，用来保存此屏幕的一些gui信息，比如哪个菜单项被高亮显示了，有了这个，在显示的时候都会按照保存的信息来显示，如果没有，则只是按照初始化的方式来显示，而且这个一般是在退出屏幕时写入历史记录的时候才保存起来。所以每次屏幕开始时都要去查一下看gui\_buffer是否已经在历史记录中了,如果在其读出来用在后面的显示中.

接着就是完成一些显示菜单前的准备工作,比如得到一些菜单字符串,菜单图标等等,不过这一些对不同的菜单不都相同,总之是根据下面要用到的用来显示菜单的函数的需要来决定的.

然后是重点,调用函数ShowCategory14Screen,这个函数就是开始说到的显示Category的那些模板函数,这些函数都已经把菜单的显示过程都封装好了,只需要用户根据需要去调用就是了,里面的参数比较复杂,不过很多都通用,比如菜单项的字符串,图标,还有历史GuiBuffer等,基本上每个菜单都会有,其他的参数则根据菜单的不同而出现不同的差异.不过也都很好理解.

最后显示完后,设置软键盘事件,保存好退出函数的句柄.这样,一个屏幕的显示过程就算结束了.

## 3.2编辑控件相关

编辑控件,就是为了响应用户的输入而设置的一些专门GUI空间,它的使用往往跟ShowCategory57Screen函数联系在一起的. Category57就是专门为显示输入控件来设置的专门屏幕.里面可以放置一系列的输入响应控件,就好像一个容器一样.

下面我们以闹钟的编辑为例来具体说明Category57及编辑控件的使用.

我们略过前面的一切屏幕显示函数,直接来到函数EntryAlmEditScreen,这个是用户点击了编辑闹钟后进入的屏幕.在里面,我们可以看到它跟一般的屏幕的显示看起来好像差不多,只是多了一些什么东西.首先我们会发现它多了一个inlinebuffer,这个东西和guibuffer差不多,都是用来记录历史状态的,比如哪个处于高亮显示状态,没什么好说的.然后我们看到一个wgui\_inline\_item类型的数组wgui\_inline\_items,这个数组就是用来保存那些输入控件的一些信息的,它的每一个项代表一个输入控件,我们只要把需要用到的输入控件,通过一些控制函数加到这个数组中,然后把这个数组传给ShowCategory57Screen函数显示出来就可以了.听起来很简单吧,这主要是由于framework帮我们做了大量的工作.

下面我们看看对数组wgui\_inline\_items进行的设置,也就是添加控件的过程.在函数AlmFillInlineEditStruct里,我们看到对于不同的控件进行了不同的设置,用到的设置函数各不相同.我们这里只对那个时间控件的设置进行说明.

首先是SetInlineItemActivation,设置控件是否需要一些按键来让它激活.一般都是自动激活,这个函数是通用的,对其他输入函数同样有效.然后是关键函数SetInlineItemTime,此函数不同的控件有不同的函数,这里设置时间为这个函数,这个函数主要看到其参数gAlmHourBuf和gAlmMinBuf,这两个参数就是传进去用于接收用户设置的时间的指针.当用户输入后,时间就保存在这两个全局变量中.虽然其他输入控件的设置方式各不相同,但大多都有一个需要传入的指针用来接收用户输入的内容.

# 举例

## 4.1一些模块的重点

### 4.1.1闹钟模块

不管此模块中的各种菜单显示,直接找到函数AlmUpdateRTCInfoForAT,这个函数只在手机硬件上运行时才有效果,在模拟器上无效.这个函数是闹钟设置函数,把用户设置好的闹钟更新到硬件系统中去.不过我们看到它发到硬件的消息是这个消息MSG\_ID\_MMI\_EQ\_SET\_RTC\_TIME\_REQ,这个消息同样是用做设置系统时间的消息,区分它们是靠mmi\_eq\_set\_rtc\_time\_req\_struct结构中的rtc\_type来区分,不同的类型控制设置不同的时间.

### 4.1.2通话模块

在CallManagementIncoming.c里面的ProcessIncomingEvents函数,是来电管理函数的核心.协议栈消息和键盘消息都往这个函数发,这个函数负责把这些消息分配到对应的.

## 4.2CALLLOG模块分析报告

此模块的进入分两个步骤，首先是在系统启动时初始化，然后是从主菜单选择进入。不过主要的还是在初始化时完成了大部分工作，后面从主菜单进入只是响应已经注册的相关事件。

首先是系统在启动InitializeAll的时候，就进行CALLLOG模块的InitCallLogs函数。在InitCallLogs里面遇到的第一个函数就是SetProtocolEventHandler( ReadyLogIndication, PRT\_MMI\_PHB\_LAST\_NUMBER\_READY\_IND);。此函数注册了一个协议栈消息的响应函数。我估计当系统启动后，检测结束发现通话记录部分已经准备好了，Protocol stack/L4就向MMI Queue写消息PRT\_MMI\_PHB\_LAST\_NUMBER\_READY\_IND。所以为了能接收到这个状态消息，CallLog模块的初始化需要及时进行。

然后进入ReadyLogIndication（）函数，这个当系统准备好就执行的函数将完成一系列的检测和初始化。先把callLogready设置为TRUE，然后依次把已拨电话，未接电话，和已接电话全部接收一遍后把first\_time设置为1。

下面我们用一个查询已拨电话来说明CallLog模块的运行。

1. 在初始化时注册已拨电话的菜单高亮显示事件。HighlightDialedCallsHandler（）；当高亮显示此菜单时运行此函数。此函数里注册了一些响应键盘消息的函数，我们只关心ReqReadDailNumByKey（）。
2. ReqReadDailNumByKey（）此函数相对查询已接电话和未接电话流程是多出来的，主要是根据first\_time和callLogready来判断是否能进入查询流程，这里假设可以就进入了ReqReadDailNum()。
3. ReqReadDailNum()，这个函数是查询已拨电话比较关键的一步，它是建立消息并向NS/L4 Queue发送查询消息，并且注册得到查询结果后的处理函数。消息号是：PRT\_MMI\_PHB\_GET\_LAST\_NUMBER\_REQ，这个消息号是提供所有历史记录号码的查询的。还要建一个GET\_LAST\_NUMBER\_REQ类型数据指针，并给他分配相应大小的内存，然后把PHB\_LND赋给GET\_LAST\_NUMBER\_REQ结构中的type，表示需要查询的是已拨电话的号码。把建好的指针赋给消息的数据指针，这样，消息就算建完了。注册消息返回响应函数RspReadDailNum(void \*info)，把查询消息发送出去。
4. 当查询有结果时，执行RspReadDailNum(void \*info)，其中此函数包含了一个指针参数。如果没猜错的话，此指针就是我们发出的查询消息里面包含的数据指针oslDataPtr。只是它指向的内存区域已经被查询结果填满。我们在这里要做的就是一个个的把查询结果读出来，然后转化为我们自己定义的电话记录格式。此过程通过一个循环加一个转化函数CovertPhbLastNum2DialCall完成，并依次把所得到的结果放在全局数组dialedCalls里。然后要说到一个状态变量callLogProcess，它表示程序处于什么工作状态，比如在读取，在写入历史记录等。说到写入历史记录。CallLog模块包含了几个函数提供给call management模块。比如:LogDialedCall提供写入已拨电话。LogDialedCallDuration提供写入总的通话时间。当不在写入时把已拨电话全部显示出来的函数EntryScrViewDialedCallList，
5. 在EntryScrViewDialedCallList中把已拨电话全列出来，并且注册成菜单。并注册菜单高亮显示时的回调函数GetCurrDialCallIndex用来得到目前所选中的菜单，并把序号保存在currDialCallIndex。这是一种很常用的方法。很多地方都有。注册右键点击函数EntryScrViewDialedCall。用来查看消息信息。
6. 在EntryScrViewDialedCall里，显示出一个已拨电话的详细信息，包括时间，号码姓名。显示后注册左键菜单函数EntryScrDialedListOptions
7. 在EntryScrDialedListOptions（）中显示一个菜单，到这里，一个流程基本结束。这个函数显示的菜单是可以对已拨号码进行的各种行为，包括：删除，保存，修改，拨打。这个函数没有直接去注册各菜单行为的入口函数。而是用此函数RegisterHighlightHandler( ExecuteCurrHiliteHandler );，这个函数一次性执行了所有菜单的注册高亮显示的入口函数。那就是在InitCallLogs中注册的那些。

以上是查询已拨电话的一个流程，至于未接电话和已接电话记录的查询都是和这个类似

的。下面是对记录的电话号码可以进行的操作。在上面最后步骤中弹出了选择菜单。而其中的菜单项的高亮显示函数都在InitCallLogs中注册了。下面也是通过用已拨电话号码来分析这些菜单项的实现。

1. 首先高亮显示拨打时，执行HighlightDialedCallHandler函数，在InitCallLogs中注册的。里面注册好快捷键并且画上左，右键并注册好他们的入口函数。左键的入口函数是CallToDialedCallNumber，在点选左键时进入此函数。此函数中通过全局变量dialedCalls和currDialCallIndex 可以找到当前正在处理的电话记录。并通过记录中的号码或姓名来拨打电话，调用的函数是MakeCall（PS8）;
2. 高亮显示保存时，流程和上面一样，只是左键入口函数是SaveDialedCall，在这个函数中调用电话薄模块的SaveNumberFromIdleScrn函数，把号码显示在屏幕上，此函数在此暂不做讨论。
3. 高亮显示删除时候，流程和上面2个还是一样，只是左键入口函数是DeleteDialedCallConfirmation，在里面弹出一个删除确认窗口，并注册左键为DeleteDialedCall函数。在DeleteDialedCall里，只是简单的调用了一个ReqDeleteDialedNum函数。在ReqDeleteDialedNum中，又是建立消息，然后再向NS/L4 Queue发送。只是这次消息是PRT\_MMI\_PHB\_DEL\_LAST\_NUMBER\_REQ，消息的数据指针指向的数据类型是DEL\_LAST\_NUMBER\_REQ型。它包含要删除的号码的类型是PHB\_LND和要删除的号码的索引index。然后注册返回消息的处理函数为RspDeleteDialedNum，再把消息发出去。当收到返回消息时，根据result.flag判断是否已经删除成功。0表示成功。如果成功，则把全局数组dialedCalls中的对应项删除。
4. 高亮显示编辑时，开始流程和前面3个都一样，左键入口函数变为EditDialedCall，在里面调用IdleScreenDigitHandler，把号码显示出来用于编辑，此函数不是本模块，暂时不讨论。

下面我们看看从菜单的入口。菜单的入口在CallHistoryMain.c里面，里面的InitCallHistory包含了其他几个模块的初始化函数，它也是在系统启动时的InitializeAll中执行的。在函数EntryScrCallHistoryMainMenu中，显示菜单，并且相应的菜单响应都已经注册好了。

# 资源的添加

## 5.1总述

在手机程序中我们常常需要用到一些外部的资源，这些资源包括字符串，图片，铃声，菜单等等，这些资源被引入到手机程序中使用，只提供一个资源的ID，让程序员可以通过这些ID访问到资源。

手机上要用到的资源，都被编译成一个DLL文件。而且这些资源的添加方式都是通过外部工具来添加的。这些外部工具把资源文件及它们的ID要联系起来，生成用来编译成资源DLL的代码。然后才能把这些资源编译成DLL文件。

## 5.2字符串资源的添加

### 5.2.1 字符串资源添加过程

为了更清楚的看清字符串添加的整个过程，这里提供了一张图片，画的是流程图：

从图中我们可以看到添加一个字符串的一般过程。可以分为两个部分，

第一个部分是CustStrList.txt和Ref\_list.txt的连接。CustStrList.txt中包含的语言只有英语，而Ref\_list.txt可以由用户来自己定义语言。这两个通过MergeStrRes.exe链接后，找到两个文件中具有相同ID的字符串，把他们保存在CustStrList\_out.txt中，这个文件包括了多国语言。

第二部分是在我们需要使用字符串资源的地方对字符串资源名称进行定义，在enum\_list.h中。然后由一个专门的程序在第一个过程生成的CustStrLis\_out.txt中找我们定义了的需要使用的字符串名，找到后根据这些字符串生成两个我们在程序中可以直接使用的.c文件，这两个文件就包含了我们定义的字符串资源。



### 5.2.2 字符串资源添加方法

上面说了字符串添加的过程通过这个过程，通过这个过程，我们可以看出，实际上我们添加字符串资源文件，只需要先在CustStrList.txt和Ref\_list.txt中按照它们的格式，添加需要使用的字符串资源，然后再在enum\_list.h中添加这些字符串资源的定义。其实这样还没结束，还需要在Res\_xxx.c中用ADD\_APPLICATION\_STRING2命令来添加需要使用的字符串。

## 5.3图片资源的添加

图片资源的添加比较简单，首先是把需要使用的图片放在\Customer\Images目录下，最好是在此目录下自己对不同的模块建立不同的目录。然后找到可以在这个Customer\CustResource\SCORPIO\_MMI\Res\_MMI目录下找到相应模块的资源文件添加文件Res\_xxx.c，使用函数ADD\_APPLICATION\_IMAGE2，把图片引入到程序中，其中图片ID需要自己定义。

## 5.4菜单资源的添加

1. 菜单资源的添加和图片文字不同，它不必从外界来引入别的文件，只需要用函数来指明各种菜单之间的父子兄弟关系。也是在Res\_xxx.c中添加，菜单资源的添加函数ADD\_APPLICATION\_MENUITEM(args),其中的args包括:1.菜单名2。其父菜单名。3。其子菜单的个数。4。依次列出其子菜单名。5。显示或隐藏参数可以取值（SHOW,HIDE）6。其他显示参数一，可以取值的都在枚举MOVEORFIXED中规定了，常用的有MOVEABLEACROSSPARENT。7。其他显示参数二，可以取的值都在枚举DISPATTRIBUTE中规定，DISP\_LIST 是常用的参数。8。菜单中显示的字符串的资源ID号。9。菜单中的图片资源ID号。