MTK 智能穿戴入门篇

——疯壳·线下课程系列

Fengke-Team

2017/08/02

目录

官网地址：https://www.fengke.club/GeekMart/views/mall/goodsDetails.html?productId%3D33
配套书籍：https://www.fengke.club/GeekMart/views/mall/goodsDetails.html?productId%3D73
配套视频：http://www.fengke.club/GeekMart/su_fihsGbMhu.jsp
淘宝地址：http://shop115904315.taobao.com/
QQ 群：457586268

MTK 编程——资源

MTK 的资源分为很多种，有图片资源、铃声资源、字符串资源、屏幕资源、菜单资源、定时器资源、nvram 资源，消息接收器等。所有的资源都定义在 .res 文件中，使用 XML 语句书写，所有的资源在 res 文件中都是用<APP id="APP_NAME"></APP>包含，而 APP_NAME 需要在 mmi_pluto_res_range_def.h 文件中设置各个资源的个数。
例如：

```
/*****************************************************************************
* Mainmenu
*****************************************************************************/
MMI_RES_DECLARE(APP_MAINMENU, 300, ".\\MMI\\MainMenu\\MainMenuRes\\")
#define MAIN_MENU_BASE                          ((U16) GET_RESOURCE_BASE(APP_MAINMENU))
#define MAIN_MENU_BASE_MAX                      ((U16) GET_RESOURCE_MAX(APP_MAINMENU))
```

（1）MMI_RES_DECLARE(APP_MAINMENU, 300, ".\\MMI\\MainMenu\\MainMenuRes\\")
APP_MAINMENU 定义在 MainMenuRes.res 文件中，包含所有的资源。300 是指资源的最大个数，这里并不是所有资源相加的，而是图片、字串等资源各自可以添加 300 个。".\\MMI\\MainMenu\\MainMenuRes\\"表示 APP_MAINMENU 所在资源文件的目录。
（2）#define MAIN_MENU_BASE ((U16) GET_RESOURCE_BASE(APP_MAINMENU))
资源 ID 的最小取值，所有资源 ID 的数字大于 MAIN_MENU_BASE
（3）#define MAIN_MENU_BASE_MAX ((U16) GET_RESOURCE_MAX(APP_MAINMENU))
资源 ID 的最大取值，所有资源 ID 的数字小于 MAIN_MENU_BASE，实际上 MAIN_MENU_BASE + 300 = MAIN_MENU_BASE_MAX。
添加完资源之后，都需要使用 make resgen（make FengKe2502C_11C GPRS resgen）编译资源，生成二进制。如果在真机上调试，在 make resgen 之后还需要使用 make r mmiresource 重新编译资源模块，或者直接使用 make u mmiresource 编译，在模拟器上调试可以不用编译 mmiresource。
资源编译之后，都会在 plutommi\Customer\CustomerInc 目录下生成一个 mmi_rp_app_name_def.h 的文件，比如上面的 app_name 为 APP_MAINMENU，则生成的文件名为 mmi_rp_app_mainmenu_def.h 。在该文件中，所有同类型的资源都会生成一个 enum 枚举，枚举命名格式为 mmi_rp_app_name_style_enum，style 为资源类型，比如 mmi_rp_app_mainmenu_str_enum 则为所有的字串资源 ID。如果 res 文件中的 ID，在这个文件里能够找到，说明资源生成成功。习惯上所有资源 ID 名称都用大写。

屏幕资源

在 MTK 设备上，每一个屏幕都对应唯一一个屏幕 ID。屏幕之间的相互切换，屏幕历史堆栈管理，都是通过屏幕 ID 进行的。屏幕资源的添加方法使用如下语句：

<SCREEN id="SCREEN_NAME"/> 或者 <SCREEN id="GROUP_NAME"/>

屏幕 ID 资源的命名方式，习惯上以 SCR_开头，比如 <SCREEN id="SCR_ID_IDLE_MAIN"/>，或者以 GRP_ 开头。

比如 <SCREEN id="GRP_ID_IDLE_MAIN"/>。通常情况下，以 SCR_开头的资源一般作为实际的单个屏幕，而以 GRP_开头的资源都用于屏幕组（group），一个 GRP 屏幕组下面，可以包含多个 SCR 屏幕，当你关闭一个 GRP 屏幕组时，那么其下的所有 SCR 屏幕都被关闭了。实际上 SCR 和 GRP 都是同类型的资源，在使用的时候反过来也没有问题，但我们反对这么做，因为从命名上来看会导致歧义，减弱代码的可读性和可维护性。在 MTK 中所有的屏幕是呈一个树形结构，（见下图）

以下介绍几个常用的屏幕相关接口：

1、 MMI_ID mmi_frm_group_create (MMI_ID parent_id, MMI_ID group_id, mmi_proc_func proc, void *user_data)
函数功能：创建一个屏幕组。
参数 parent_id：屏幕组的父类，通常情况下可以填 GRP_ID_ROOT。
参数 group_id：是要创建的屏幕组 ID。
参数 proc：屏幕组的消息处理函数
参数 user_data：用户需要传递的数据，通常情况下用 NULL

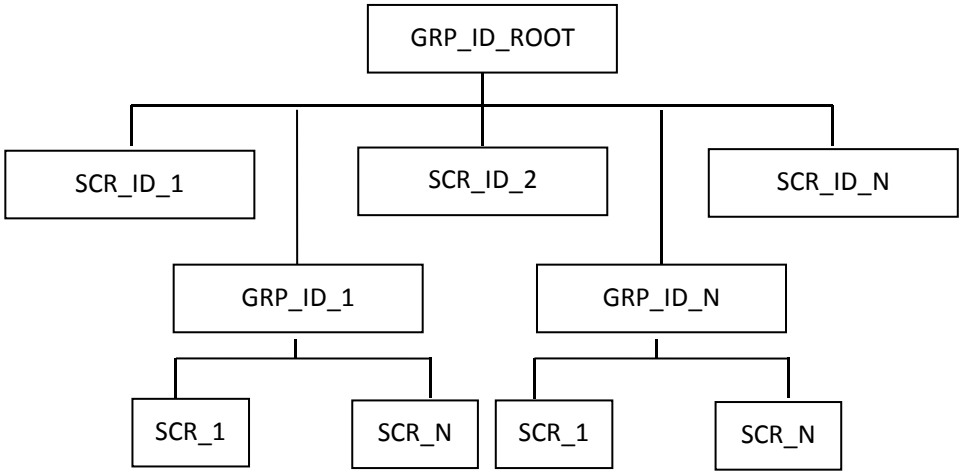
2、 MMI_BOOL mmi_frm_scrn_enter (MMI_ID parent_id, MMI_ID scrn_id, FuncPtr exit_proc, FuncPtr entry_proc, mmi_frm_scrn_type_enum scrn_type)
函数功能：进入一个屏幕。
参数 parent_id：屏幕的父类，可以是我们自定义的屏幕组，也可以填 GRP_ID_ROOT。
参数 scrn_id：屏幕 ID。
参数 exit_proc：退出屏幕时的处理函数，一般用于释放资源。如果没有退出函数，则填 NULL。
参数 entry_proc：屏幕的入口函数，当屏幕从历史堆栈中弹出时，会执行这个函数。
参数 scrn_type：屏幕类型，一般常用的是 MMI_FRM_FULL_SCRN 和 MMI_FRM_UNKNOW_SCRN

3、 mmi_ret mmi_frm_group_close (MMI_ID group_id)
函数功能：关闭屏幕组 group_id

4、 mmi_ret mmi_frm_scrn_close (MMI_ID parent_id, MMI_ID scrn_id)
函数功能：关闭屏幕组 parent_id 下的 scrn_id 屏幕

5、 void mmi_frm_scrn_close_active_id (void)
函数功能：关闭当前处于激活状态的屏幕，即用户能够看到的屏幕。

6、 U16 GetActiveScreenId(void)
函数功能：获取当前处于激活状态的屏幕 ID。



屏幕 ID 树状结构

接下来我们自己定义一个屏幕 ID，屏幕组 ID 暂时不讲解。在 Idle.res 文件中找到<SCREEN id="SCR_ID_IDLE_SUBLCD"/>，在后面添加一句<SCREEN id="SCR_ID_MY_MTK_FUNC"/>，代码如下：

```
<!----- Screen Resource Area -----------------
<SCREEN id="GRP_ID_IDLE_MAIN"/>
<SCREEN id="SCR_ID_IDLE_MAIN"/>
<SCREEN id="SCR_ID_IDLE_SUBLCD"/>
<SCREEN id="SCR_ID_MY_MTK_FUNC"/>
```

然后用 make resgen 编译，编译完成之后打开 mmi_rp_app_idle_def.h 文件，在 mmi_rp_app_idle_scr_enum 枚举中看是否能找到 SCR_ID_MY_MTK_FUNC，如果能找到说明资源添加成功了。接下来我们将 mmi_my_mtk_func 函数作为一个屏幕入口函数，让它进入一个屏幕，同样打印出"Hello MTK !"，并注册右软键功能关闭该屏幕，然后添加一个屏幕退出函数 mmi_my_mtk_func_exit，在退出该屏幕时，我们打印"Exit Hello MTK !"，具体代码如下：

```
void mmi_my_mtk_func_exit(void)
{
    gui_set_text_color(UI_COLOR_BLACK);/*设置字符打印颜色*/
    gui_move_text_cursor(100, 200);/*设置字符打印坐标*/
    gui_set_font(&MMI_medium_font);/*设置字符显示的字体*/
    gui_print_text(L"Exit Hello MTK !");/*打印字符*/

    gui_BLT_double_buffer(0, 0, UI_DEVICE_WIDTH,UI_DEVICE_HEIGHT);
}

void mmi_my_mtk_func(void)
{
    mmi_frm_scrn_enter(GRP_ID_ROOT, SCR_ID_MY_MTK_FUNC, mmi_my_mtk_func_exit,mmi_my_mtk_func,MMI_FRM_FULL_SCRN);

    gui_set_text_color(UI_COLOR_BLACK);/*设置字符打印颜色*/

    gui_move_text_cursor(100, 150);/*设置字符打印坐标*/

    gui_set_font(&MMI_medium_font);/*设置字符显示的字体*/

    gui_print_text(L"Hello MTK !");/*打印字符*/

    /*刷新屏幕*/
    gui_BLT_double_buffer(0, 0, UI_DEVICE_WIDTH,UI_DEVICE_HEIGHT);

    /*注册右软键事件*/
    SetKeyHandler(mmi_frm_scrn_close_active_id, KEY_RSK, KEY_EVENT_UP);
}

/*****************************************************************************
```

在 VS2008 中编译，运行模拟器。我们点击左软件的时候，屏幕上依旧会显示"Hello MTK !"，当我们点击右软键的时候，屏幕上会显示"Exit Hello MTK !"，但是会闪一下就消失，因为先执行退出函数，然后会立即执行 idle 界面的入口函数，绘制 idle 界面的内容，从而把"Exit Hello MTK !"覆盖掉了。读者可自己使用断点，查看代码执行流程。

字符资源

在上一个例子中，我们显示"Hello MTK !"直接使用的是字符常量，这样的方式在单一语言下是没有问题的，但如果系统包含多国语言，比如从英文切换到中文，如何把"Hello MTK !"直接显示成中文呢？这里就要用到字符资源，字符资源全部添加在 plutommi\Customer\CustResource\PLUTO_MMI\ref_list.txt 文件中，里面包含约 80 种语言字符，我们只要输入对应语言的字符，那么字符显示成何种语言就会随系统的语言设置而变化。MTK 提供了一个专用于修改字符资源的工具——plutommi\Customer\ STMTView.exe。接下来我们把"Hello MTK !"添加成字符资源，然后通过 GetString 函数接口提取字符串。添加字符资源的语句格式为：

<STRING id="STR_NAME">"default string"</STRING> 或 <STRING id=" STR_NAME"/>

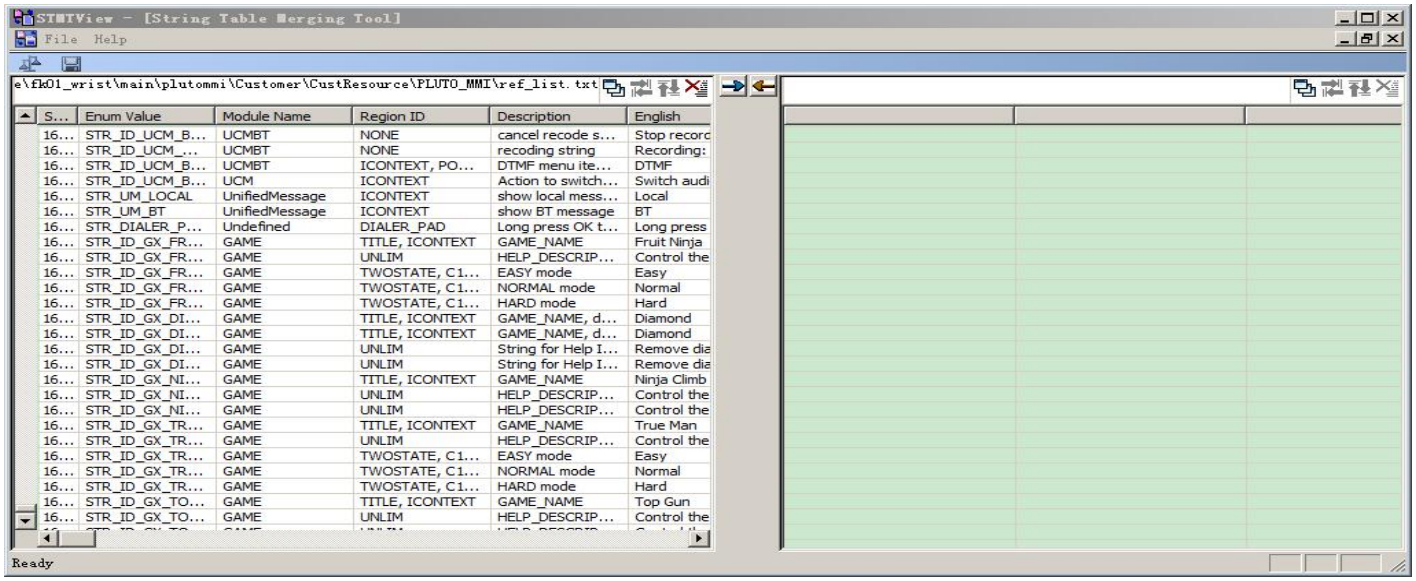
字符资源 ID 命名通常以“STR_”开头，这两种方法的区别在于前者设置了默认值，如果 STR_NAME 在 ref_list.txt 不存在的话，那它在任何语言下都会显示"default string"（注意：此处添加的默认值只能是英文字符）。而后者没有默认值，如果 STR_NAME 在 ref_list.txt 不存在的话，请读者自己测试会显示什么内容。

我们采用设置默认值的方式添加字符资源，在 idle.res 文件中添加字符资源 ID——STR_ID_HELLO_MTK，代码如下：

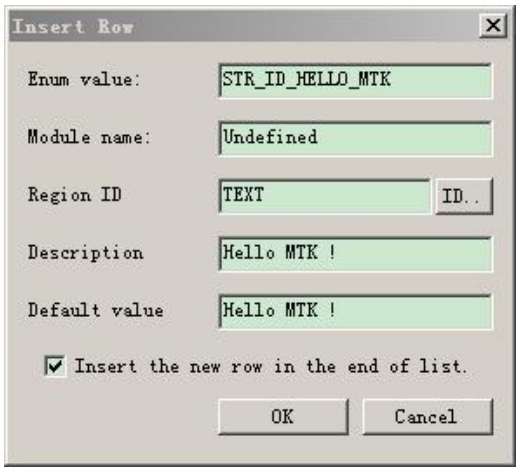
```
#endif
#endif /* defined(__MMI_IDLE_SCREEN_ICON_SHORTCUT__) */

#if defined(__MMI_BT_DIALER_SUPPORT__)
    <STRING id="STR_ID_IDLE_BTDIALER_CONNECT"/>
    <STRING id="STR_ID_IDLE_BTDIALER_DISCONNECT"/>
#endif /* defined(__MMI_BT_DIALER_SUPPORT__) */

    <STRING id="STR_ID_HELLO_MTK">"Hello MTK !"</STRING>

    <!----- Image Resource Area ----------
#if defined(__MMI_IDLE_SCREEN_ICON_SHORTCUT__)
    <IMAGE id="IMG_ID_IDLE_ICON_SHORTCUT_AUDIO_DOWN">CUST_IMG_PATH"\
```

然后在 plutommi\Customer\CustResource\FengKe2502C_11C_MMI\MMI_features_switchFengKe2502C_11C.h 文件中找到宏开关 CFG_MMI_LANG_SM_CHINESE 把后面的(__OFF__)改为(__ON__)，这里是让系统支持简体中文，为方便我们验证字符资源的多国语言翻译问题。接下来还要为系统添加字库，在 FengKe2502C_11C_GPRS.mak 文件中设置 FONT_ENGINE = FONT_ENGINE_ETRUMP，使用矢量字库。

打开 plutommi\Customer\STMTView.exe，点击 File—>Open STMT… 会显示两个表格栏，随便选择一个，单击点击红色标注的图标  ，在弹出的文件选择对话框中，选择 plutommi\Customer\CustResource\PLUTO_MMI\ref_list.txt 然后打开，出现如下界面：



在打开文件的一栏中单击鼠标右键，选择 Insert new row。如下图所示填充弹出框：



1、Enum value：字符资源的枚举 ID，必须跟.res 文件中添加的资源 ID 一一对应，否则资源无法加载。

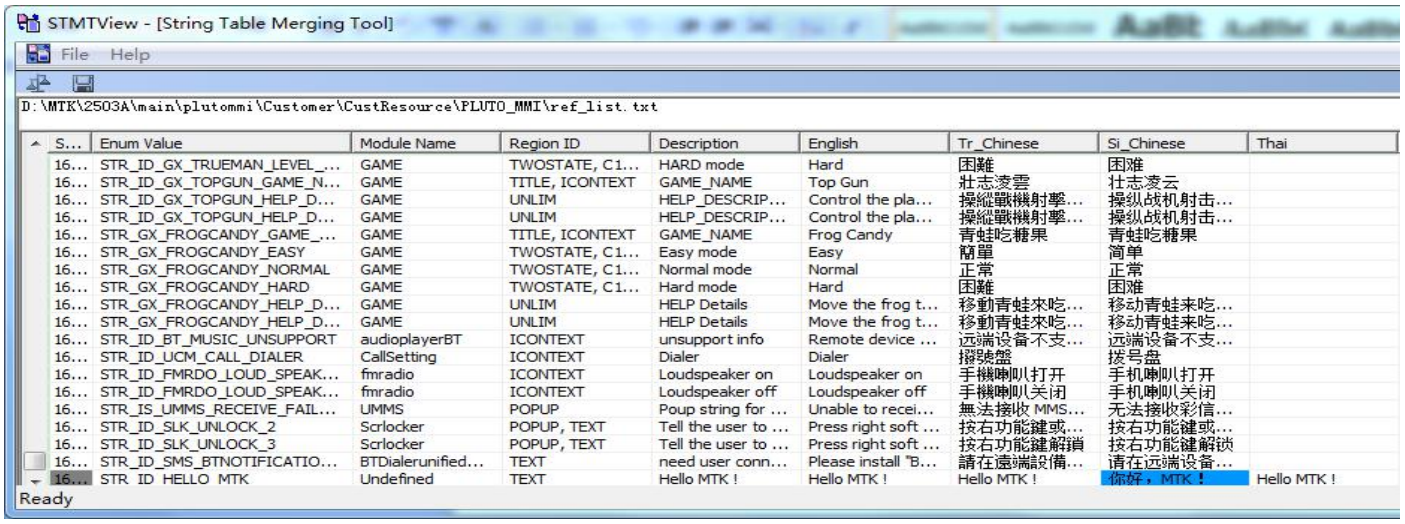
2、Mode name：字符资源所在的功能模块，这个可以使用默认的值，仅作为描述。

3、Region ID：这个也仅是描述作用，可以手动输入，也可以点击旁边的按钮选择，也可以不填。

4、Description：这个也仅是描述作业，可以描述字符 ID 的用途。

5、Default value：字符资源 ID 的默认值，在任何语言下都是会默认使用这个值。

下面的复选框打勾，把资源 ID 添加到文件的最后一行，点击 OK。在新增的一行，对应 Si_Chinese 列把"Hello MTK !"改成“你好，MTK !”，添加效果如图：



在 mmi_my_mtk_func 函数中把 L"Hello MTK !" 改成 GetString(STR_ID_HELLO_MTK)，暂且把 mmi_my_mtk_func_exit 函数中的代码清空，具体代码如下：

```
void mmi_my_mtk_func_exit(void)
{
}

void mmi_my_mtk_func(void)
{
    mmi_frm_scrn_enter(GRP_ID_ROOT, SCR_ID_MY_MTK_FUNC, mmi_my_mtk_func_exit,mmi_my_mtk_func,MMI_FRM_FULL_SCRN);

    gui_set_text_color(UI_COLOR_WHITE);/*设置字符打印颜色*/

    gui_move_text_cursor(10, 15);/*设置字符打印坐标*/

    gui_set_font(&MMI_medium_font);/*设置字符显示的字体*/

    gui_print_text((UI_string_type)GetString(STR_ID_HELLO_MTK));/*打印字符*/

    /*刷新屏幕*/
    gui_BLT_double_buffer(0, 0, UI_DEVICE_WIDTH,UI_DEVICE_HEIGHT);

    /*注册右软键事件*/
    SetKeyHandler(mmi_frm_scrn_close_active_id, KEY_RSK, KEY_EVENT_UP);
}

/*****************************************************************************
 * FUNCTION
 *  mmi_idle_set_handler
 * DESCRIPTION
 *  This function sets the default handler according to the capability of the
 *  idle object.
 * PARAMETERS
 *  obj          : [IN]        Idle object
 * RETURNS
 *  void
 *****************************************************************************/
void mmi_idle_set_handler(mmi_idle_obj_struct *obj)
```

上面我们修改了 MMI_features_switchFengKe2502C_11C.h 和 FengKe2502C_11C_GPRS.mak 这两个文件，只要修改了这两个文件的任意一个，都必须执行 make new 编译，再用 make gen_modis 重新生成模拟器。如果这两个文件没有修改，则 make resgen 就可以了。编译完后打开 plutommi\Customer\ResGenerator\debug\string_resource_usage.txt 文件看是否能找到 STR_ID_HELLO_MTK，系统中所有的字符资源 ID 编译成功之后都可以在这个文件中找到，如果找不到则说明资源添加失败。字符资源编出来之后就是 UCS2 编码。在 VS2008 中运行模拟器，点击 KEY_LSK，运行效果如下：

系统默认是英语环境，所以依旧显示"hello MTK !"，接下来我们把系统切换成中文。打开 GeneralSettingSrv.c 文件，在 srv_setting_get_language 函数中把 return data;改为 return 1;再次运行模拟器，点击 KEY_LSK，运行效果如下：

字符资源的最大优势，就是可以支持多国语言，而对于程序员来说，只有一个资源 ID。在调用 GetString 时，它会根据系统环境的语言设置，获取对应的字符串指针。

图片资源

MTK 中显示的图片有三种，一是通过资源加载的，二是显示磁盘中的文件，三是直接把图片转成二进制数组显示，这种方式常用于 SP 游戏或应用开发。此处我们只讲图片资源。图片资源跟字符资源一样，也是添加在 .res 文件中，添加图片资源的语句格式为：

<IMAGE id="IMG_NAME">CUST_IMG_PATH"image_filepath\\\\image_filename"</IMAGE>

图片资源 ID 命名通常以“IMG_”开头，所有的图片资源都放在 plutommi\Customer\Image 文件夹中，在这个文件夹中包含不同屏幕尺寸的图片资源，每个屏幕尺寸对于的文件夹下都有一个 image.zip 压缩包中，系统中的图片资源就放在这个压缩包中，我们添加的图片也要放到压缩包对应的目录中，在编译资源的时候，系统会将其解压成 MainLCD 目录。CUST_IMG_PATH 是一个宏，定义在 plutommi\Customer\CustResource\PLUTO_MMI\ CustResDefPLUTO.h 文件中，它指向 image.zip 所在的目录，比如当前屏幕尺寸为 240X320（屏幕尺寸的配置在 FengKe2502C_11C_GPRS.mak 文件中查看 MAIN_LCD_SIZE 宏定义），则 CUST_IMG_PATH 定义指向 plutommi\Customer\Images\PLUTO240x320 目录，系统中定义的代码如下：

```
#elif defined(__MMI_MAINLCD_240X320__)
    #define CUST_IMG_PATH           "..\\\\..\\\\Customer\\\\Images\\\\PLUTO240X320"
#elif defined(__MMI_MAINLCD_320X240__)
    #define CUST_IMG_PATH           "..\\\\..\\\\Customer\\\\Images\\\\PLUTO320X240"
```

image_filepath 是相对于 MainLCD 的目录，image_filename 就是图片的文件名，包含后缀名。图片资源的格式可以是 bmp、png、jpg、gif 还有 MTK 系统特定的一些图片格式。系统中添加成功的图片资源，其资源 ID 都可以在文件 plutommi\Customer\ResGenerator\debug\ image_resource_usage.txt 中找到。如果资源加载失败，则资源 ID 会被添加到相同目录的 image_load_fail.txt 文件中，这种情况通常都是图片的目录错误，或者 image.zip 中没有这个图片文件。

接下来我们添加一个图片资源，在 plutommi\Customer\Images\PLUTO240x320\MainLCD\IdleScreen\Wallpaper 目录下，随便找一张图片，或者自己随便找一张图片放到 image.zip 对应的目录中。这里我就使用 WALL01.jpg 来做测试。在 idle.res 文件中添加一个资源 ID——IMG_ID_HELLO_MTK，并加载 WALL01.jpg 图片，代码如下：

```
#ifdef __MMI_SUBLCD__
    <IMAGE id="IMG_ID_IDLE_SUBLCD_LOGO">CUST_IMG_PATH"\\\\SubLCD\\\\Active\\\\SB_ON.bmp"</IMAGE>
#endif

    <IMAGE id="IMG_ID_HELLO_MTK">CUST_IMG_PATH"\\\\MainLCD\\\\IdleScreen\\\\Wallpaper\\\\WALL01.jpg"</IMAGE>

    <!----- Screen Resource Area ---------------------------------------------->
    <SCREEN id="GRP_ID_IDLE_MAIN"/>
```

然后使用 make resgen 编译，编译完成之后在 image_resource_usage.txt 看是否能找到 IMG_ID_HELLO_MTK，如果能找到，说明资源加载成功。接下来在 mmi_my_mtk_func 函数中调用 gdi_image_draw_id 接口把图片显示出来。因图片是黑色，所以我们把字符颜色改为白色（UI_COLOR_WHITE），代码如下：

```
void mmi_my_mtk_func_exit(void)
{
}

void mmi_my_mtk_func(void)
{
    mmi_frm_scrn_enter(GRP_ID_ROOT, SCR_ID_MY_MTK_FUNC, mmi_my_mtk_func_exit,mmi_my_mtk_func,MMI_FRM_FULL_SCRN);

    gui_set_text_color(UI_COLOR_WHITE);/*设置字符打印颜色*/

    gdi_image_draw_id(0, 0, IMG_ID_HELLO_MTK);/*显示图片*/

    gui_move_text_cursor(10, 15);/*设置字符打印坐标*/

    gui_set_font(&MMI_medium_font);/*设置字符显示的字体*/

    gui_print_text((UI_string_type)GetString(STR_ID_HELLO_MTK));/*打印字符*/

    /*刷新屏幕*/
    gui_BLT_double_buffer(0, 0, UI_DEVICE_WIDTH,UI_DEVICE_HEIGHT);

    /*注册右软键事件*/
    SetKeyHandler(mmi_frm_scrn_close_active_id, KEY_RSK, KEY_EVENT_UP);
} ? end mmi_my_mtk_func ?

/*****************************************************************************
 * FUNCTION
 *  mmi_idle_set_handler
```

运行模拟器，效果如图：

显示图片的资源有：

1、 gdi_image_draw_id(OFFSET_X,OFFSET_Y,IMAGE_ID)

函数功能：这是一个宏函数，根据图片 ID 在屏幕上显示图片

OFFSET_X：图片的显示的 X 坐标。

OFFSET_Y：图片的显示的 Y 坐标。

IMAGE_ID：图片资源 ID

2、 gdi_image_draw_resized_id(OFFSET_X,OFFSET_Y,RESIZED_WIDTH,RESIZED_HEIGHT,IMAGE_ID)

函数功能：这个函数可以实现图片的缩放，不管图片的尺寸是多少，都显示成指定的大小。

OFFSET_X：图片的显示的 X 坐标。

OFFSET_Y：图片的显示的 Y 坐标。

RESIZED_WIDTH：图片显示的宽度，如果大于这个宽度则缩小，如果小于这个宽度则放大。

RESIZED_HEIGHT：图片显示的高度，同 RESIZED_WIDTH 一样缩放图片。

IMAGE_ID：图片资源 ID

3、 gdi_image_draw(OFFSET_X,OFFSET_Y,IMAGE_PTR)

函数功能：这个函数的执行效果跟 gdi_image_draw_id 函数是一样的，只不过参数需传入一个图片的数组

OFFSET_X：图片的显示的 X 坐标。

OFFSET_Y：图片的显示的 Y 坐标。

IMAGE_PTR ：图片二进制数组，通常可通过 GetImage(IMAGE_ID)获取。

另外，图片资源还有 gif 动画，动画资源的添加方式跟图片资源是一样的，只不过显示的函数接口有点区别，动画的接口包含在 gdi_include.h 文件中，通常是以 gdi_anim_ 开头的函数，比如 gdi_anim_draw_id，关于动画的使用本章暂时不做过多的讲解，有兴趣的读者可以自己看代码学习。

菜单资源

菜单的添加方法比图片和字符稍微复杂些，格式有如下三种：

1、 菜单中包含子菜单。

< MENU ……[item value]…… >

<MENUITEM_ID>MENU_ITEM_ID</MENUITEM_ID>或 <MENUITEM id=" MENU_ITEM_ID " ……/>

……

</ MENU >

如果不包含子菜单则可以写成：

< MENU ……[item value]…… /> 或< MENU ……[item value]…… ></ MENU >

2、菜单中不包含子菜单

<MENUITEM id=" MENU_ITEM_ID " ……/>

3、菜单作为主菜单

<MAINMENUITEM id="MENU_ID" ……/>

此处的 MENU_ID 必须是通过第一种方式已经加载成功的菜单资源 ID。

以上三种格式的省略号（……）省略了一些属性值，有些属性是可有可无的，在后面通过代码来讲解。在上一个例子中，我们直接把 mmi_my_mtk_func 放在 idle 界面的，通过注册按键的方式执行。在这一节中我们把 mmi_my_mtk_func 放到菜单中，作为菜单的功能入口函数。

首先我们把 idlecommon.c 文件中 mmi_idle_set_handler 函数里最后一行 SetKeyHandler(mmi_my_mtk_func, KEY_LSK, KEY_EVENT_UP);去掉。添加一个 mmi_highlight_my_mtk 函数，作为菜单的高亮处理函数，并注册左右软键功能，代码如下：

```
void mmi_my_mtk_func_exit(void)
{
}

void mmi_my_mtk_func(void)
{
    mmi_frm_scrn_enter(GRP_ID_ROOT, SCR_ID_MY_MTK_FUNC, mmi_my_mtk_func_exit,mmi_my_mtk_func,MMI_FRM_FULL_SCRN);

    gui_set_text_color(UI_COLOR_WHITE);/*设置字符打印颜色*/

    gdi_image_draw_id(0, 0, IMG_ID_HELLO_MTK);/*显示图片*/

    gui_move_text_cursor(10, 15);/*设置字符打印坐标*/

    gui_set_font(&MMI_medium_font);/*设置字符显示的字体*/

    gui_print_text((UI_string_type)GetString(STR_ID_HELLO_MTK));/*打印字符*/

    /*刷新屏幕*/
    gui_BLT_double_buffer(0, 0, UI_DEVICE_WIDTH,UI_DEVICE_HEIGHT);

    /*注册右软键事件*/
    SetKeyHandler(mmi_frm_scrn_close_active_id, KEY_RSK, KEY_EVENT_UP);
} ? end mmi_my_mtk_func ?

void mmi_highlight_my_mtk(void)
{
    SetLeftSoftkeyFunction(mmi_my_mtk_func, KEY_EVENT_UP);
    SetKeyHandler(mmi_frm_scrn_close_active_id, KEY_RSK, KEY_EVENT_UP);
}
```

然后在 MainMenuRes.res 文件中添加一个菜单 MENU_MY_MTK_ID ，并把它放到 MAIN_MENU_SETTINGS_MENUID 设置菜单下面，代码如下：

```
#ifdef __MMI_WEARABLE_DEVICE_SETTING__

    /*添加菜单ID*/
    <MENU id="MENU_MY_MTK_ID" type="APP_MAIN" str="STR_ID_HELLO_MTK" img="IMG_GLOBAL_OK" highlight="mmi_highlight_my_mtk"/>

/* Setting */
    <MENU id="MAIN_MENU_SETTINGS_MENUID" type="APP_MAIN" str="MAIN_MENU_SETTINGS_TEXT" img="MAIN_MENU_SETTINGS_ICON"
        highlight="highlight_mainmenu_settings" shortcut="ON" shortcut_img="MAIN_MENU_TITLE_ICON" launch="EntryScrSettingMenu">

                <MENUITEM_ID>MENU_MY_MTK_ID</MENUITEM_ID>/*添加菜单ID到设置菜单下*/

                <MENUITEM_ID>@OID:MENU_CONN_BT_MAIN</MENUITEM_ID>
#ifdef __HOTKNOT_SUPPORT__

                <MENUITEM_ID>@OID:MENU_HOTKNOT_SETTING</MENUITEM_ID>
#endif
                <MENUITEM_ID>@OID:MENU9102_INITIAL_SETUP</MENUITEM_ID>
                <MENUITEM_ID>@OID:MAIN_MENU_PROF_SOUND_SETTING</MENUITEM_ID>
```

在 idlecommon.c 文件中 mmi_idle_set_handler 函数最后一行，把 SetKeyHandler(mmi_my_mtk_func, KEY_LSK, KEY_EVENT_UP);改为 SetKeyHandler(EntryScrSettingMenu, KEY_LSK, KEY_EVENT_UP);强制进入设置菜单选项。执行 make resgen 再到 VS2008 中重新编译并运行模拟器，点击 LSK，进入“设置”菜单，在设置菜单界面就可以看到我们自己添加的菜单了“你好，MTK！”，再点击确定，就执行了我们之前添加的函数 mmi_my_mtk_func。如下图所示：

1. <MENU id="MENU_MY_MTK_ID" type="APP_MAIN" str="STR_ID_HELLO_MTK" img="IMG_GLOBAL_OK"
   highlight="mmi_highlight_my_mtk"/>
   type="APP_MAIN"：描述菜单的类型。取值可选 APP_SUB，Option。
   str="STR_ID_HELLO_MTK"：菜单名称的字符串资源 ID，如上图设置菜单列表界面显示的“你好，MTK !”。
   img="IMG_GLOBAL_OK"：菜单的图标资源 ID，显示在菜单的最坐标，但这里没有显示，而是用编号代替，如上图编号 1。如果显示菜单自己的图标，需要修改代码，打开相关功能宏。
   highlight="mmi_highlight_my_mtk"：菜单被选中的时候，会执行该处理函数。

2. <MENUITEM_ID>MENU_MY_MTK_ID</MENUITEM_ID>
   在其他菜单下面添加子菜单。此处还有另一种写法，可用下面的语句代替，并删除上面第 1 条语句。最终执行的结果是一样的，请读者自己尝试。
   <MENUITEM id="MENU_MY_MTK_ID" str="STR_ID_HELLO_MTK" img="IMG_GLOBAL_OK"
   highlight="mmi_highlight_my_mtk"/>

菜单添加成功可以在 plutommi\Customer\CustResource\CustMenuTree_Out.c 文件中找到对应的 ID，这个文件包含系统中所有菜单 ID 的依赖关系，菜单跟屏幕一样也是呈树形结构，IDLE_SCREEN_MENU_ID 是树的根，除此之外，所有的菜单 ID 在该文件中至少出现两次。

如果添加菜单时有 highlight 属性，则在 plutommi\Framework\EventHandling\EventsInc\mmi_menu_handlers.h 文件中可以查看到 ID 与 highlight 函数的对应关系。

铃声资源

铃声资源的添加方式跟图片资源相似，系统中所有的铃声文件都包含在 plutommi\Customer\Audio\PLUTO\audio.zip 压缩包中，编译资源时会被解压，所以我们添加的铃声文件也要放到这个压缩包中。添加铃声资源的格式为：

<AUDIO id="AUD_ID_NAME" flag="MULTIBIN">CUST_ADO_PATH"\\\\file_path \\\\file_name"</AUDIO>

说明：

id="AUD_ID_NAME"：资源 ID 的名称。

flag="MULTIBIN"：描述资源的属性，这个属性可有可无。

CUST_ADO_PATH：宏定义，指向 plutommi\Customer\Audio\PLUTO 目录

file_path：audio.zip 压缩包中，铃声文件所在的目录。

file_name：铃声文件名。格式可为 mp3、wav、mid、imy 等。

系统中大多数铃声资源都在情景模式中，见 ProfilesSrv.res 文件。铃声资源编译成功后，可以在 plutommi\Customer\ResGenerator\debug\audio_resource_usage.txt 文件中找到 ID 以及与其对应的铃声文件。铃声资源的播放，可以用 get_audio 获取铃声资源的数组，然后再用 mdi_audio_play_string_with_vol_path 函数播放。此处不做代码测试，留给读者自己完成。

nvram 资源

MTK 设备有些数据在关机之后也要保存。实现这种功能的方法有两种，一种方法是存储到磁盘文件中，另一种方法就是我们本节要讲的 nvram（简称 NV）。而 nvram 也分为两种，一种可以存储复合数据类型，比如结构体；而另一种只能存储数值类型，比如 byte、short、double，我们用的最多的是 byte 和 short。存储数值类型的 NV 就是通过资源的方式添加，添加 NV 资源的格式为：

```
<CACHEDATA type="type" id="NVRAM_NAME" restore_flag="TRUE/FALSE">
    <DEFAULT_VALUE> [value] </DEFAULT_VALUE>
    <DESCRIPTION> description </DESCRIPTION>
    <FIELD min="min_number" max="max_number"></FIELD>
</CACHEDATA>
```

说明：

type=" type"：type 的取值有 byte、short、double。byte 只能存一个字节，short 能存两个字节，double 可以存 8 个字节，一般用的很少，如果存储的数据量较大，则作为复合数据类型存储。

id="NVRAM_NAME" : NVRAM 的资源 ID，NV 的读写都是通过 ID 进行。

restore_flag="TRUE/FALSE"：TRUE 恢复出厂设置时重置该 NV 的值为默认值

<DEFAULT_VALUE> [value] </DEFAULT_VALUE>：NV 的默认值，type 为 byte，取值最大为[0xFF]；short 最大为[0xFF,0xFF]，左侧为高为，右侧为低位；double 最大为[0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]，同样是左侧高位，右侧低位。比如 type 为 short，存储 value 为 300，转换成十六进制为[0x01,0x2C]。

<DESCRIPTION> description </DESCRIPTION>：NV 的描述，相当于注释。这个属性可以缺省。

<FIELD min="min_number" max="max_number"></FIELD>：描述 NV 的取值范围。这个属性也可以缺省。

比如，系统中保存背光亮度和背光时间的 NV，在 gpiosrv.res 文件中定义如下：

```
<CACHEDATA type="byte" id="NVRAM_BYTE_BL_SETTING_LEVEL" restore_flag="TRUE">
    <DEFAULT_VALUE> [0xFF] </DEFAULT_VALUE>
    <DESCRIPTION> Backlight Level </DESCRIPTION>
    <FIELD min="1" max="20"></FIELD>
</CACHEDATA>

<CACHEDATA type="short" id="NVRAM_BYTE_BL_SETTING_HFTIME" restore_flag="TRUE">
    <DEFAULT_VALUE> [0xFF, 0xFF] </DEFAULT_VALUE>
    <DESCRIPTION> Backlight timer </DESCRIPTION>
</CACHEDATA>
```

NV 资源的读写通常用以下两个函数：

1、S32 ReadValue(U16 nDataItemId, void *pBuffer, U8 nDataType, S16 *pError)

函数功能：读取 NV 中存储的值。

nDataItemId：NV 的资源 ID。

pBuffer：保存 NV 中读取到的数值。

nDataType：NV 的类型，取值有 DS_BYTE、DS_SHORT、DS_DOUBLE，分别对应资源中的 byte、short、double 类型

pError；这个参数在函数实现中暂时并没有实际的作用。

例句：ReadValue(NVRAM_BYTE_BL_SETTING_HFTIME, &hftime, DS_SHORT, &error);

2、S32 WriteValue(U16 nDataItemId, void *pBuffer, U8 nDataType, S16 *pError)

函数功能：把 pBuffer 中的数值写入 NV 中。

nDataItemId：NV 的资源 ID。

pBuffer：需要写入 NV 中的数值。

nDataType：NV 的类型，取值有 DS_BYTE、DS_SHORT、DS_DOUBLE，分别对应资源中的 byte、short、double 类型

pError；这个参数在函数实现中暂时并没有实际的作用。

例句：WriteValue(NVRAM_BYTE_BL_SETTING_HFTIME, &hftime, DS_SHORT, &error);

本节只简单的介绍 NV 资源的添加，以及读写方式，在后面的章节中会有单独一章详细讲解 NVRAM，包括存储简单数据类型和复合数据类型。

定时器资源

定时器的作用就是延后执行某一个操作，也可以每隔一段时间就执行一次。定时器 ID 的添加方式有两种，一种是在 TimerEvents.h 文件中的 MMI_TIMER_IDS 枚举中添加，只要添加在 KEY_TIMER_ID_NONE 和 MAX_TIMERS 之间就可以了，这种添加方式比较简单，通常驱动里面要用的定时器都添加在这个文件中。另一种方式是通过资源 ID 方式添加，这种方式添加的定时器跟第一种方式添加的定时器没有本质的区别，在使用上也是完全一模一样。MMI 层用到的定时器两种添加方式都没有问题，但是强烈建议使用第二种，方便代码的模块化。

定时器资源的添加方式为：

<TIMER id="TIMER_ID_NAME"/>

TIMER_ID_NAME 为自定义的定时器 ID 名称。定时器启动和停止的常用函数接口如下：

1、 void StartTimer(U16 timerid, U32 delay, FuncPtr funcPtr)

函数功能：启动定时器。

timerid：定时器的资源 ID。

delay：定时器执行的时间间隔。

funcPtr：定时器执行的函数。

2、 void StopTimer(U16 timerid)

函数功能：停止定时器。

timerid：定时器的资源 ID。

消息接收器

这个资源在实际开发中用的比较少，一般都是系统自带的。我们只需要了解一下就可以了，消息接收器的定义方式如下：

<RECEIVER id=" RECEIVER _ID_NAME" proc="reveiver_func"/>

说明：

id=" RECEIVER _ID_NAME"：为消息 ID

proc="reveiver_func"：消息 ID 的处理函数。这个函数一般由系统调用。