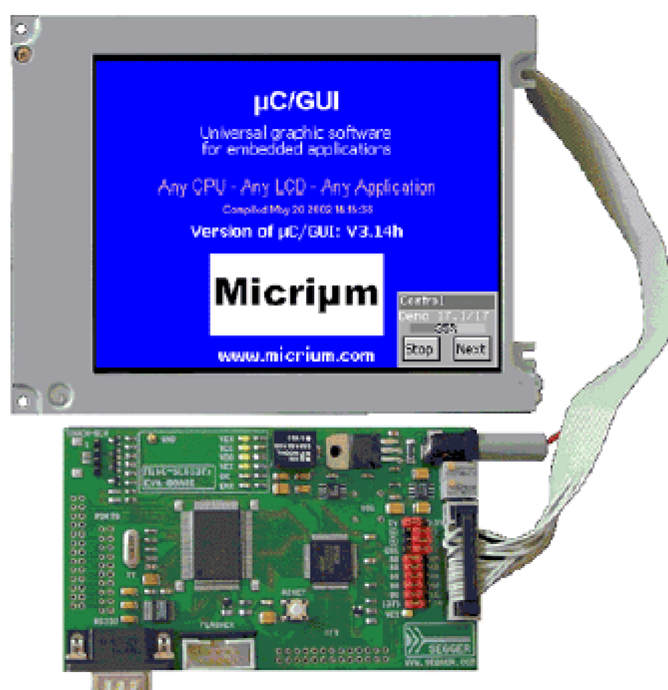
第3章 仿真器

μC/GUI的PC仿真器允许你在Windows下编译相同的“C”源程序。PC使用一个本地编译器（一般是微软所提供的）并建立一个用于你自己应用的可执行文件。这样做可能完成：

- 在你的PC上进行用户接口设计（不需要硬件支持）
- 调试你的用户接口程序
- 建立你的应用的演示，可以用于描述用户接口

最终的可执行文件能够很容易通过E-Mail传送。



3.1 理解仿真器

µC/GUI仿真器使用微软Visual C++（6.0或更高版本）及其所带的集成开发环境（IDE）。你能够在PC屏幕上看到你的LCD仿真效果，一旦正确配置你的LCD后仿真效果能提供与你的LCD在X轴和Y轴上相同的分辨率及同样精确的颜色。

仿真的整个图形库API和视窗管理API与你的目标系统是一样的；所有函数运行与在目标硬件上运行高度一致，因为仿真时使用了与目标系统同样的“C”源代码。唯一不同是在软件的底层：LCD驱动。PC仿真使用一个仿真的驱动写入一个位图，以代替实际的LCD驱动。在你的屏幕上显示的位图使用第二个仿真线程。第二个线程在实际应用并不存在，它只是在LCD程序被直接写屏时运行。

3.2 在评估版µC/GUI中使用仿真器

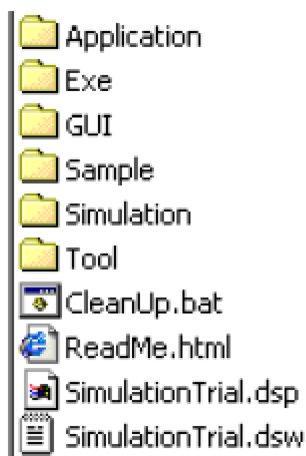
版本

µC/GUI 评估版包括一个完全的库，允许你评估µC/GUI的所有效特性。同时也包括µC/GUI观察器（用于调试应用程序）及字体转换器和位图转换器的演示版。

请记住这些，做为一个评估版本，你不能改变任何配置或者察看源代码，但是你仍然能够熟悉µC/GUI的使用。

目录结构

评估版仿真器的目录结构如下图所示。



目录“Application”包括演示程序的源代码。

目录“Exe”包括一个“ready-to-use”演示程序。

目录“GUI”包括库文件和库使用的包含文件。

目录“Sample”包括仿真范例及其源代码。

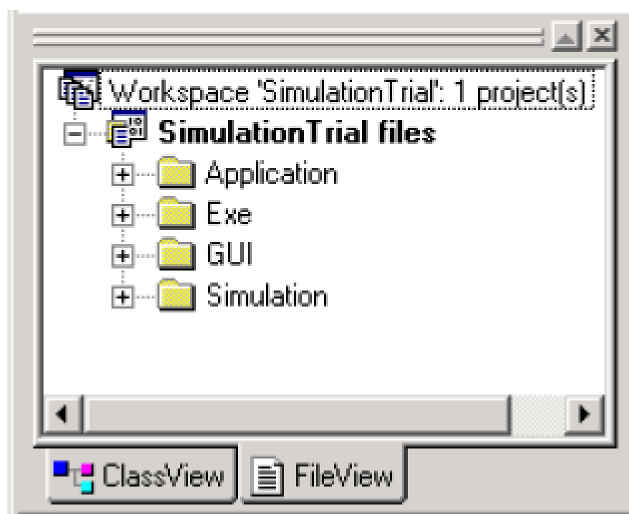
目录“Simulation”包括仿真所需的文件。

目录“Tool”包括μC/GUI观察器，一个演示版的位图转换器和一个演示版的字体转换器。

Visual C++工作区

上面所示的根目录包括微软Visual C++工作区（Simulation-Trial.dsw）及项目文件（Simulation-Trial.dsp）。双击工作区文件可以打开微软IDE。

Visual C++工作区的目录结构如下图所示。



编译演示程序

位于应用目录下的演示程序源文件是一个“ready-to-go”仿真，意思是你仅仅需要建立和启动它。请注意，如果需要建立可执行文件，你必须先安装微软Visual C++（6.0或以上的版本）。

- 第一步：双击Simulation-Trial.dsw 文件打开Visual C++工作区。
- 第二步：在菜单中选择“Build/Rebuild All” （或按“F7”键）重建项目。

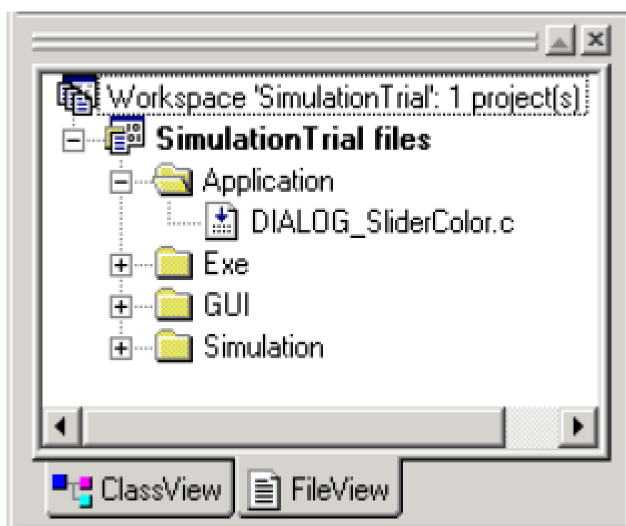
- 第三步：在菜单中选择“Build/Start Debug/Go” （或按“F5”键）开始仿真。

演示项目开始运行，在任意时候可能通过单击右键并选择“Exit”退出。

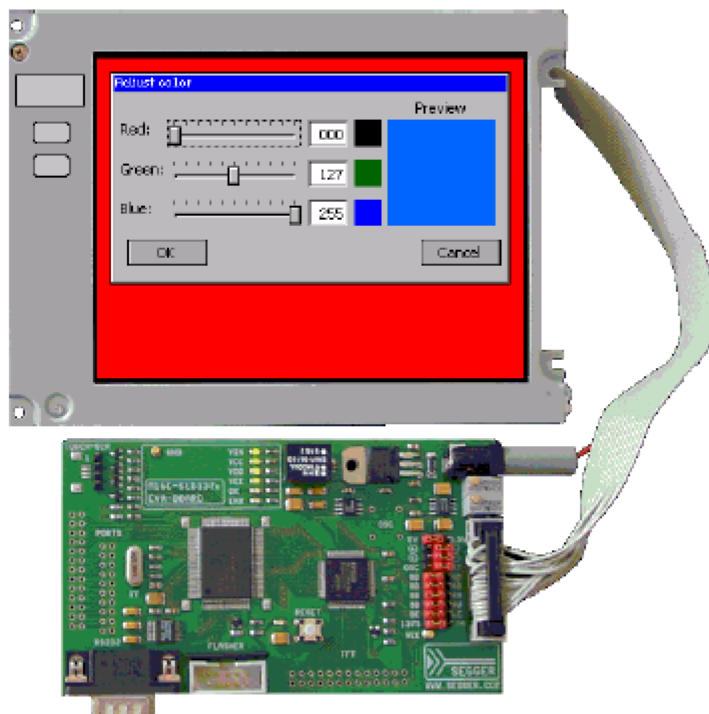
编译范例程序

目录“Sample”包括“ready-to-go”范例程序，可以示范µC/GUI的不同特性及提供它们的典型应用的例子。为了建立这些可执行文件，它们的C源代码必须加入项目中。通过下面的步骤很容易做到：

- 第一步：双击Visual C++工作区的“Application”文件夹。演示文件会出现在它下面。
- 第二步：选择“Application”文件夹下的所有文件，按下“Delete”键将它们删除。这些文件并不是真是被删除了，只是从项目中移走。
- 第三步：现在你有了一个空的“Application”文件夹。在其上面单击右键，选择需加入的文件加入到文件夹，出现一个对话框。
- 第四步：双击“Sample”文件夹，选择里面的一个范例文件。你的工作区目录应该如下图所示。当然，文件名可以不一样；在这里，很重要的一件事是“Application”文件夹只能包含你所想编译的范例的C文件，而不能是其它种类的文件。



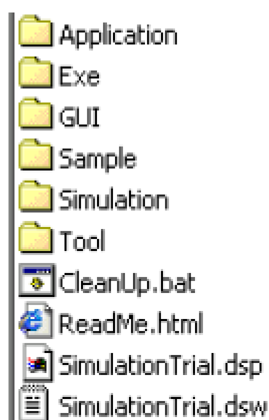
- 第五步：在菜单下选择“Build/Rebuild All”（或按“F7”键）重建范例文件。
- 第六步：在菜单中选择“Build/Start Debug/Go” （或按“F5”键）开始仿真。上面所选择范例的仿真结果如下图所示：



3.3 使用µC/GUI 源代码的仿真器

目录结构

仿真器的根目录可以在PC上的任意位置，例如：C:\work\GSCSim。目录结构如下图所示，该目录结构与我们推荐的用于目标应用的目录结构很相似（参阅第2章：“入门指南”以获得更多信息）。子目录GUI包括µC/GUI程序文件，用于目标（交叉）编译器的同名目录中也要有同名的文件。你不应对GUI子目录做任何改变，因为这样会使µC/GUI升级到新的版本变得困难。配置目录包括需要修改的配置文件，以反映你的目标硬件设置（主要是LCD尺寸和能够显示的颜色）。

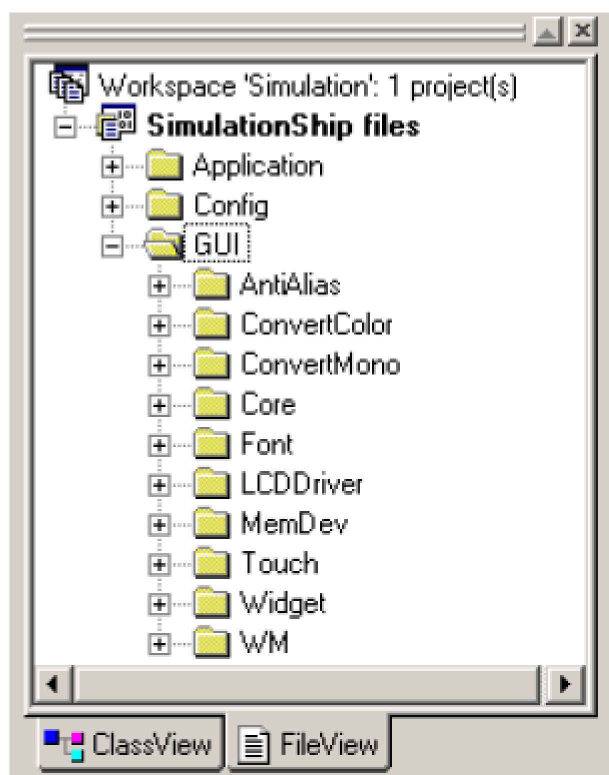


Visual C++工作区

上面所显示的根目录包括微软Visual C++工作区（Simulation.dsw）和项目文件（Simulation.dsp）。

工作区允许你在编译应用程序将其用于目标系统之前进行修改及调试。

Visual C++ 工作区的目录结构与下图所示的相似。在这，GUI文件夹是打开的，显示μC/GUI子目录。请注意，你的GUI目录可能与下图并不完全一样，这取决于你所拥有的μC/GUI的附加特性。文件夹“Core”，“Font”及“LCDDriver”是基本μC/GUI软件包的一部分，总在工作区目录中显示。



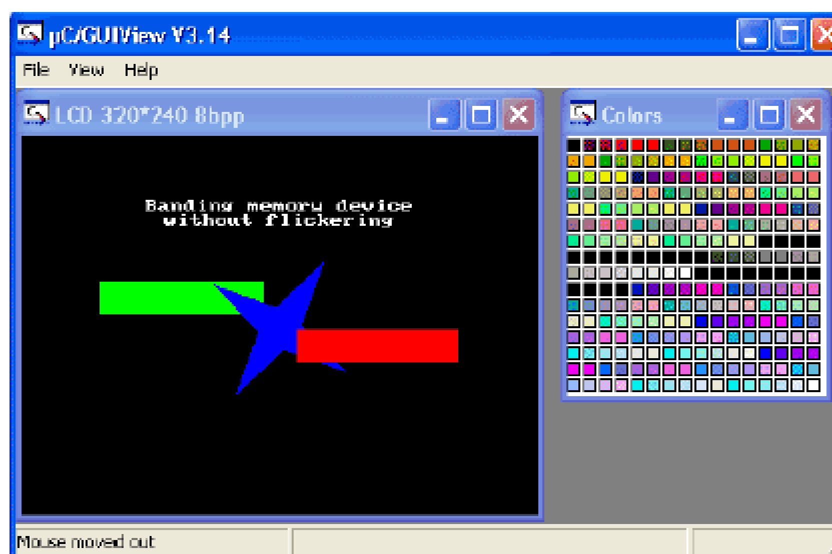
编译你的应用程序

仿真演示包括一个或多个可以修改的C文件（在“Application”目录下），你也可以在项目中增加或删除文件。最典型的是你至少应该把位图修改成为你公司的徽标或所选择的图片。你应该在Visual C++ 工作区中重建程序以进行测试及调试。一旦你获得一个满意的结果并打算在你的应用中使用该程序，你应该能够在目标系统中编译这些同样的文件，得目标显示上得到同样的结果。使用仿真器的通常的处理步骤如下所述：

- 第一步：双击Simulation.dsw 文件打开Visual C++工作区。
- 第二步：在菜单下选择“Build/Rebuild All”（或按“F7”键）编译项目。
- 第三步：在菜单中选择“Build/Start Debug/Go”（或按“F5”键）开始仿真。
- 第四步：使用你的徽标或图片代替原有的位图。
- 第五步：如果有需要，对应用程序进行更大的修改，这通过编辑源代码和增加/删除文件来完成。
- 第六步：在Visual C++中编译及运行应用程序测试结果，根据你的需要进行继续修改和调试。
- 第七步：在你的目标系统上编译及运行应用程序。

3.4 观察器

如果你使用仿真器调试你的应用程序，当你对源代码执行单步调试时无法看到LCD输出。观察器可能解决这个问题，它能显示仿真的LCD窗口和色彩窗口。观察器的执行文件是“Tool\µC-GUI-View.exe”。



使用仿真器和观察器

如果你在调试应用程序之前或正在调试时，想启动观察器，同时使用仿真器及观察器是你的选择。我们建议：

- 第一步：启动观察器。在仿真开始前，没有LCD或色彩窗口出现。
- 第二步：打开Visual C++ 工作区。
- 第三步：编译和运行应用程序。

- 第四步：如先前描述的一样调试应用程序。

优点是，当你使用单步操作时，能在LCD窗口中显示所有相应绘图操作。观察器窗口默认的位置总是在顶部，你可能通过在菜单中选择“View\Always on top”修改这个行为。

3.5 设备仿真及其它高级特性

警告：设备仿真和以其为基础的其它特性，是高级特性，可以对仿真器源代码提出要求以使其能工作于目标系统。通常这些源代码不随µC/GUI一起提供。请和我们联系得到更多的信息。

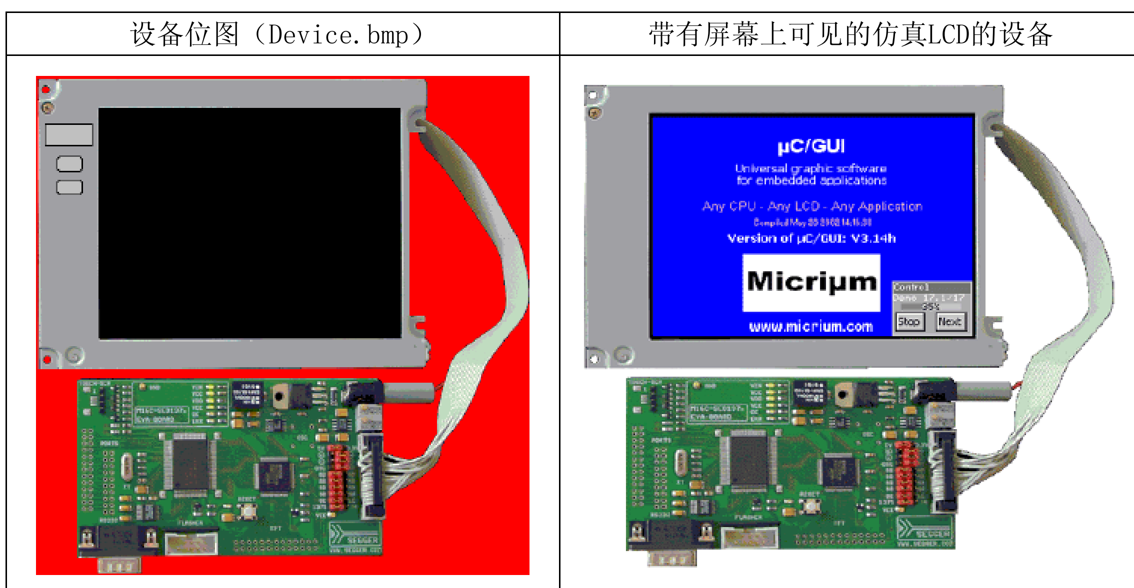
仿真器能够在你所选择的一张位图中显示仿真的LCD，例如你的目标设备的图片。该位图能够在屏幕上任意拖动，在某些应用中，可以用于仿真整个目标设备的行为。

为了仿真设备的外观，需要一幅位图。该位图通常是设备的照片（顶视图），必须命名为Device.bmp。它可以是一个独立文件（在同一目录下作为一个可执行文件），或是做为一个资源包括进应用程序当中，下面一行表示该位图文件包括在资源文件（extension.rc）中：

```
145 BITMAP DISCARDABLE "Device.bmp"
```

更多的信息，请参考Win32文档资料。

位图的尺寸应该是这样的，位图中LCD区域在屏幕上显示的尺寸应该等于仿真LCD的分辨率。在下面的例子中这是最好的显示效果：

设备位图（Device.bmp）	带有屏幕上可见的仿真LCD的设备
	

红色区域自动成为透明区。透明区没有必要是矩形；它们可以是任意的形状（甚至可以

是你的操作系统所限制的复杂形状，但是一般的就足够了）。亮红色（0xFF0000）是默认的透明区域的颜，主要因为在大多数位图中很少用到这种颜色。如果位图中含有亮红色，你可以在函数SIM_SetTransColor中改变默认的透明色。

Hardkey 仿真

Hardkey也可以作为设备的一部分进行仿真，它可以由鼠标指针选择。意思是能区别在仿真设备中的一个键或一个按钮是否被按下或没有按下。当鼠标指针位于一个Hardkey上方，并且鼠标按键保持按下状态，则该Hardkey被认为是按下。当鼠标按键释放或指针移开Hardkey，表示该Hardkey“没有按下”。在“按下”和“非按下”之间的切换行为也可以在程序SIM_HARDKEY_SETmode中说明。

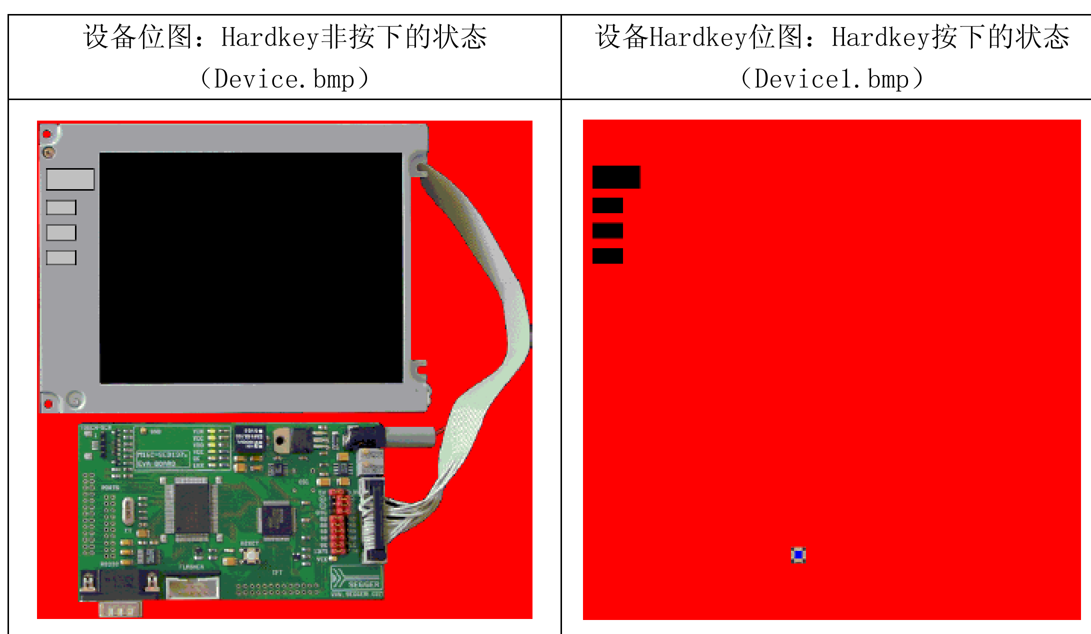
为了仿真Hardkey，你需要第二幅设备位图，除了按键自己（按下状态）以外，都是透明的。该位图也作为目录里的一个独立文件或包括在可执行文件中做为一个资源。

文件名需要定为“Device1.bmp”，典型的，下面两行表示两们位图文件包括在资源文件（extension.rc）中：

```
145 BITMAP DISCARDABLE "Device.bmp"
146 BITMAP DISCARDABLE "Device1.bmp"
```

尽管Hardkey可以是任意的形状，但有一点很重要，就是两幅位图尺寸必须是一样的，即在像素上一样，这样在Device1.bmp上的Hardkey就能够正确地覆盖在Device.bmp相应的位置上。下面的例子说明了这种情况：

设备位图：Hardkey非按下的状态（Device.bmp）	设备Hardkey位图：Hardkey按下的状态（Device1.bmp）



当一个键被鼠标“按下”，hardkey 位图（Device1.bmp）的相应部分将覆盖设备位图以显示该键处于其按下状态。

键可以周期性的轮询以确定它们的状态（按下/非按下）是否已经改变以及它们是否需要更新。二者选一，当一个hardkey的状态改变时，一个回调函数会被设置以触发一个自带的特殊动作。

3.6 仿真器API

所有仿真器的API函数在设置阶段必须被调用。调用应该从函数SIM_X_Init()内部被完美执行，该程序位于文件SIM_X.c中。下面的例子表示在设置中调用SIM_SetLCDPos()：

```
*/
#include <windows.h>
#include <stdio.h>
#include "SIM.h"
void SIM_X_Init()
{
    SIM_SetLCDPos(0,0);       // 定义LCD在位图中的位置
}
```

下表列出了与仿真相关有用的函数，在各自的类型中按字母进行排列。函数的详细描述在后面列出。

函 数	说 明
设备仿真	
SIM_SetLCDPos()	在目标设备位图中设置仿真LCD的位置
SIM_SetTransColor()	设置用于透明区域的颜色
Hardkey仿真	
SIM_HARDKEY_GetNum()	返回一个有效的Hardkey的号码
SIM_HARDKEY_GetState()	返回一个指定的Hardkey的状态（0＝非按下，1＝按下）
SIM_HARDKEY_GetCallback()	设置当指定Hardkey状态改变时要执行的回调函数
SIM_HARDKEY_SetMode()	设置一个指定Hardkey的行为（默认＝0，不切换）
SIM_HARDKEY_SetState()	设置一个指定Hardkey的状态

SIM_SetLCDPos()

描述

在目标设备位图中设置仿真LCD的位置。

函数原型

```
void SIM_SetLCDPos(int x, int y);
```

参　数	含　意
x	仿真LCD左上角（单位：像素）的 X 轴坐标
y	仿真LCD左上角（单位：像素）的 Y 轴坐标

附加信息

X和Y坐标相对于目标设备位图，因此坐标（0,0）表示位图左上角（原点）而非你的实际LCD。只有仿真屏幕的原点需要指定；你的显示器的分辨率应该已经反映在配置目录下的配置文件中。

SIM_SetTransColor()

描述

设置用于设备或Hardkey位图的透明区域的颜色

函数原型

```
I32 SIM_SetTransColor(I32 Color);
```

参　数	含　意
Color	颜色的RGB数值

附加信息

透明色的默认设置为亮红（0xFF0000）。如果你的位图包括有同样色调的红色时，你才需要这个典型的设置。

SIM_HARDKEY_GetNum()

描述

返回一个有效的Hardkey的号码

函数原型

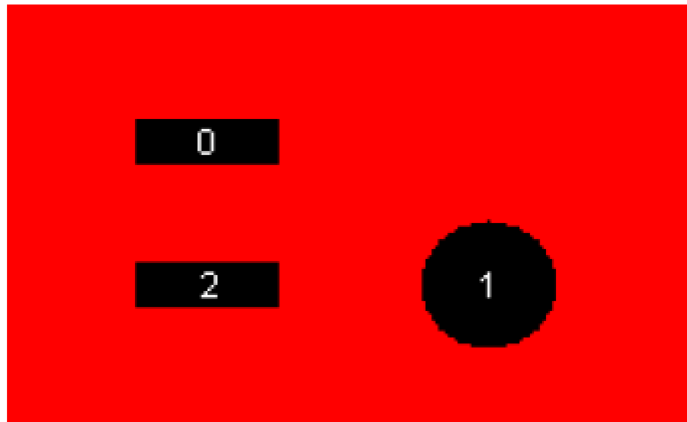
```
int SIM_HARDKEY_GetNum(void);
```

返回值

在位图中找到的有效的Hardkey的号码。

附加信息

Hardkey的序号遵循阅读顺序（从左到右，从上到下）。因此一个Hardkey最顶端的像素首先被发现，而不管它的水平位置如果。例如，在下面的位图中， Hardkey被标号，而在其它的函数中它们通过KeyIndex参数进行引用：



推荐调用该函数对一幅位图是否被完全截入进行校验。

SIM_HARDKEY_GetState()

描述

返回指定的Hardkey的状态

函数原型

```
int SIM_HARDKEY_GetState(unsigned int KeyIndex);
```

参　数	含　意
KeyIndex	Hardkey的标签（0＝第一个键的标签）

返回值

指定Hardkey的数值：

0： 非按下
1： 按下

SIM_HARDKEY_SetCallback()

描述

设置当指定Hardkey状态改变时要执行的回调函数

函数原型

```
SIM_HARDKEY_CB* SIM_HARDKEY_SetCallback (   unsigned int KeyIndex,
                                            SIM_HARDKEY_CB* pfCallback);
```

参　数	含　意
KeyIndex	Hardkey的标签（0＝第一个键的标签）
pfCallback	回调函数的指针

返回值

先前的回调函数的指针。

附加信息

回调函数原型必须如下所示：

函数原型

```
typedef void SIM_HARDKEY_CB(int KeyIndex, int State);
```

参　数	含　意
KeyIndex	Hardkey的标签（0＝第一个键的标签）
State	指定Hardkey的状态（如下所示）

参数State允许的数值：

0： 非按下

1：　　　按下

SIM_HARDKEY_SetMode()

描述

设置指定Hardkey的行为

函数原型

```
int SIM_HARDKEY_SetMode(unsigned int KeyIndex, int Mode);
```

参　数	含　意
KeyIndex	Hardkey的标签（0＝第一个键的标签）
Mode	行为模式（如下表所示）

参数Mode允许的数值

0：　　　正常行为（默认）
1：　　　切换行为

附加信息

正常（默认）Hardkey行为意思指当鼠标指针位于一个键上方，并且鼠标按键保持按下状态，则该键被认为是按下。当鼠标按键释放或指针移开Hardkey，该键被认为是非按下。

而切换行为，每一次单击鼠标对一个Hardkey进行一次“按下”或“非按下”状态的切换。这意思是如果你在一个Hardkey上方单击一下鼠标，它变成“按下”，这个状态一直保持到你再次单击鼠标。

SIM_HARDKEY_SetState()

描述

设置指定Hardkey的状态。

函数原型

```
int SIM_HARDKEY_SetState(unsigned int KeyIndex, int State);
```

参　数	含　意

KeyIndex	Hardkey的标签（0＝第一个键的标签）
State	指定Hardkey的状态（如下所示）

参数State允许的数值：

0：　　　非按下

1：　　　按下

附加信息

该函数只有在 SIM_HARDKEY_SetMode 被设为 1（切换模式）时才有效。