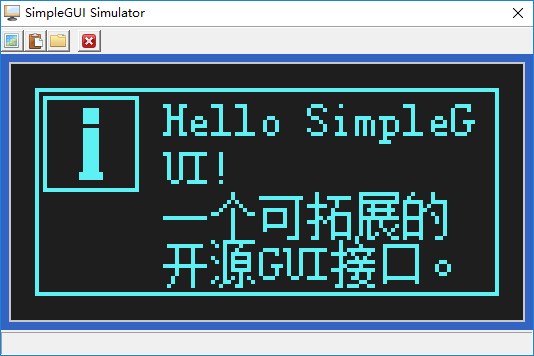
SimpleGUI

针对单色显示屏设计的开源GUI接口

**SimpleGUI API文档**



**开源，是一种态度。**

# 目录

[目录 1](#_Toc483838467)

[前言 2](#_Toc483838468)

[认识SimpleGUI模拟环境工程 7](#_Toc483838469)

[模拟器配置 10](#_Toc483838470)

[从Hello world开始。 12](#_Toc483838471)

[简单交互 15](#_Toc483838472)

[自定义拓展 18](#_Toc483838473)

[平台移植 19](#_Toc483838474)

# 前言

SimpleGUI是一款针对单色显示屏设计的一套GUI显示与控制接口，接口尽可能的做到与硬件无关、方便移植。

鉴于目前单色点阵显示屏刷新速度慢，分辨率普遍偏低的特点，SimpleGUI以尽可能提供简单人机交互方式为出发点，方便使用者在进行完平台的相关移植之后，可以尽快构建起一套自己需要的GUI系统。

本文档将以对SimpleGUI中提供的绝大多数接口函数提供尽可能详尽的功能说明与原理阐述。由于各人使用的硬件平台可能不尽相同，本文档中对API的讲解与演示将全部以模拟器环境为演示平台，效果基本与实际硬件平台相同。

# 文件结构

SimpleGUI的代码主要分为GUI和HMI两大部分，GUI部分负责绘图与现实，HMI负责交互。

在打开工程目录后，将会看到以下几个文件夹。

|  |  |
| --- | --- |
| Documents | 存放说明以及声明文档，包括您正在阅读的这篇文档。 |
| Frame | 存放模拟环境的框架源码，用于显示模拟LCD界面。 |
| GUI | 存放SimpleGUI的源码。 |
| HMI | 存放动作图(ActionMap)引擎的源码。 |
| Library | 用于存放一些第三方引用库。 |
| OutPut | 编译输出的临时文件夹。 |
| Project | 存放LCD模拟环境工程。 |
| Resource | 存放资源文件，包括程序图标与工具栏按钮图标等。 |
| ScreenShots | 存放截屏文件，如果不存在将会自动建立。 |
| User | 用户代码，用于调用SimpleGUI接口模拟显示或编写并处理一些简单逻辑事务。 |

我们日常进行模拟开发时，主要工作在User文件夹和HMI文件夹下。

进入SimpleGUI根目录后，可以在Project\CodeBlocks文件夹下找到模拟环境的工程文件SimpleGUI.cbp，需要注意的是，如果在配置环境时，使用的是绿色版的CodeBlocks，该文件类型不会被自动关联，需要用户自行创建文件关联或先启动CodeBlocks然后从程序打开工程。

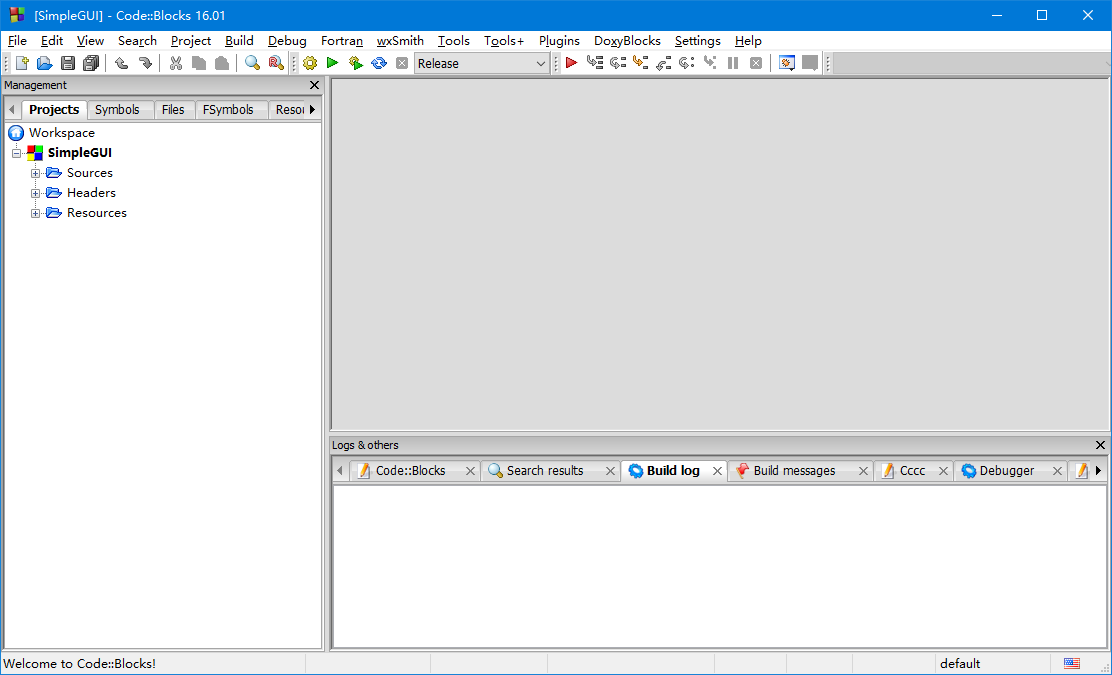


图 10 打开SimpleGUI工程

上方工具栏中，黄色齿轮图标即为编译按钮，自此向右的四个按钮分别为执行、编译并执行、完全重编译，具体操作请自行了解。

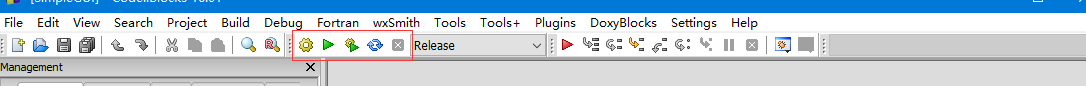


图 11 编译与执行

# 绘图接口

如果对当前颜色和显示尺寸有调整需求，可以按照以下方式对模拟器的一些参数进行设置。

打开工程的Header组下，Frame文件夹下的inc文件夹，双击并打开LCD\_VirtualDeviceParameter.h文件。

图 15 LCD模拟器参数配置文件。

文件中定义了一些模拟器的外观参数，可以通过修改这些参数对模拟器进行一些外观上的更改。

|  |  |
| --- | --- |
| LCD\_SIZE\_MARGIN | LCD显示区域与窗口边缘的间隔。 |
| LCD\_DISPLAY\_EDGE | LCD显示区域的边框。 |
| LCD\_PIXSIZE | 模拟LCD中每个像素点的尺寸。 |
| LCD\_SIZE\_WIDTH | 模拟LCD宽度的像素数。 |
| LCD\_SIZE\_HEIGHT | 模拟LCD高度的像素数。 |
| LCD\_COLOR\_OBJ\_PIX | LCD像素点的颜色。 |
| LCD\_COLOR\_OBJ\_BKG | LCD显示区域的背景色。 |
| LCD\_COLOR\_OBJ\_EDGE | LCD显示区域的边框颜色。 |
| FRM\_COLOR\_OBJ\_BKG | 模拟环境窗体的背景颜色。 |

如果对模拟环境外观有特殊需求，请根据需要，对以上参数进行更改。对于屏幕的尺寸，建议宽度与高度均不小于64像素，不大于480像素。

例如，环境默认为模拟黑底蓝色的OLED12864显示屏，如果想模拟黄底黑字的19264显示屏，则对参数做出以下修改。

|  |  |
| --- | --- |
| LCD\_SIZE\_WIDTH | (192) |
| LCD\_SIZE\_HEIGHT | (64) |
| LCD\_COLOR\_OBJ\_PIX | (2) //像素大小可自行决定 |
| LCD\_COLOR\_OBJ\_BKG | (wxColor(116,164,25)) // RGB颜色 |
| LCD\_COLOR\_OBJ\_EDGE | (wxColor(208,246,28)) // RGB颜色 |

重新编译后运行，即可看到变化效果。

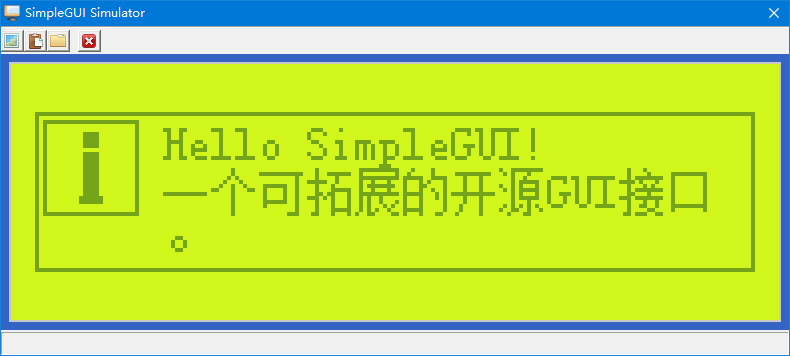


图 16 修改模拟器的配色方案

# 动作图接口

按照国际惯例，学编程从Hello world开始。

根据前文所述，用户的代码主要在User文件夹下编写，工程默认在此文件夹下只有一个UserAction.c文件，文件中有两个函数。USR\_ACT\_OnInitialize函数将在LCD模拟器启动时被调用，可以在其中编写一些初始化显示的内容。USR\_ACT\_OnKeyPress函数用于响应用户的键盘消息。

那么接下来，我们将使用绘制文字API，在屏幕上显示一些文字，比如“Hello world!”。

打开UserAction.c文件，在文件的头部添加#include "GUI\_Font.h"宏定义，然后找到USR\_ACT\_OnInitialize函数，并在其中添加如下代码：

RECTANGLE stTextDisplayArea = {5, 5, 30, 12};

RECTANGLE stTextDataArea = {0, 0, 0, 0};

GUI\_Text\_DrawSingleLineText("Hello world!", FONT\_SIZE\_H12, &stTextDisplayArea, &stTextDataArea, GUI\_DRAW\_NORMAL);

完成后如下图。

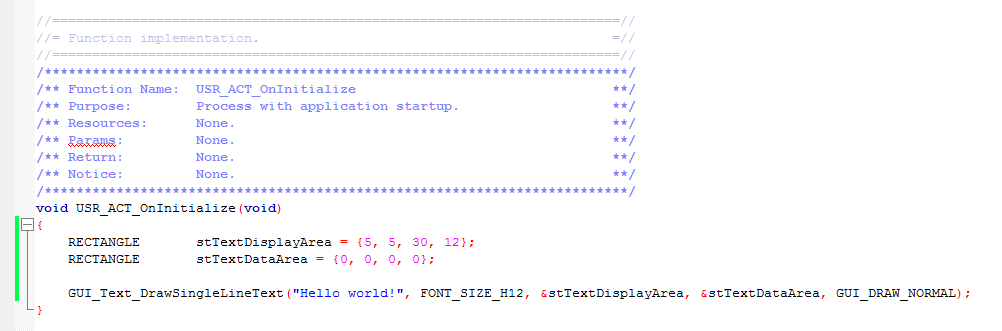


图 17 HelloWord代码

上图添加的代码，意为使用12像素字体，在屏幕上坐标(5,5)的位置，30像素宽20像素高的区域内显示“Hello word!”文字。

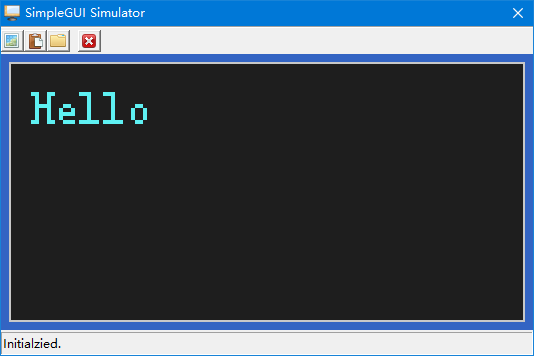


图 18 显示“Hello world”

但此时只显示了“Hello”，后面的文字却不知道哪里去了，现在再看代码，发现我们指定的显示区域只有30像素宽，超过区域的内容均没有被显示，而示例中使用了12像素的文字，宽度为高度的一半，为6像素，规定的显示区域刚好显示五个半角字符。找到了原因，现在将代码做如下修改。

RECTANGLE stTextDisplayArea = {5, 5, 50, 12};

RECTANGLE stTextDataArea = {2, 2, 0, 0};

GUI\_Text\_DrawSingleLineText("Hello world!", FONT\_SIZE\_H12, &stTextDisplayArea, &stTextDataArea, GUI\_DRAW\_NORMAL);

将文字显示区域的宽度修改为72后，重新编译执行，可以看到，完整的“Hello world!”已经显示出来了。

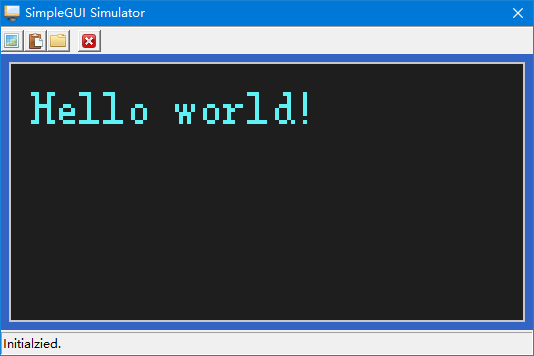


图 19 完整显示的“Hello world!”

示例代码中，定义了两个RECTANGLE类型的结构体，此结构体定义于GUI\_Basic.h文件中，用于表示一个矩形区域。显示文字API中，使用两个此类型的参数，第一个规定可用显示范围，第二个规定显示内容的尺寸与偏移。

将示例代码做如下修改，编译并运行。

RECTANGLE stTextDisplayArea = {5, 5, 50, 12};

RECTANGLE stTextDataArea = {2, 2, 0, 0};

GUI\_Text\_DrawSingleLineText("Hello world!", FONT\_SIZE\_H12, &stTextDisplayArea, &stTextDataArea, GUI\_DRAW\_NORMAL);

显示效果如下。

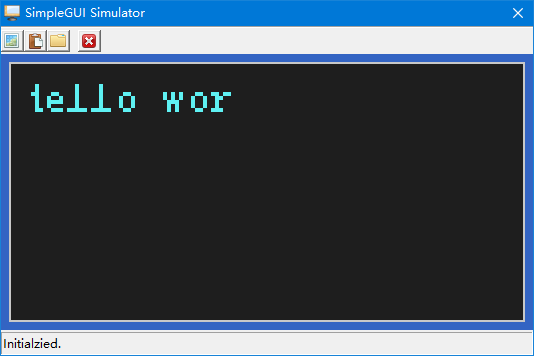


图 20 偏移数据显示

可以看到，文字在显示区域内，向左和向上偏移并忽略了3像素。

在文字显示时，由于字模的尺寸已经固定，所以显示时，文字的显示尺寸将自动计算，用户只需要指定X和Y方向上的偏移即可，但显示点阵图像时，需要明确指定图像的尺寸。

关于SimpleGUI各API的具体使用方法，后续将有专门的文档进行说明。

# 交互

UserAction.c文件中的USR\_ACT\_OnKeyPress函数可以在LCD模拟器启动后，在窗口获得焦点的情况下捕捉键盘事件。函数的参数uiKeyCode为响应的按键键值。

SimpleGUI模拟器中已经定义了一些常用的功能键，例如F1-F12功能键，空格键，回车键等，用户可以在UserAction.h文件中查看。

SimpleGUI在USR\_ACT\_OnKeyPress函数中已经写好了一些常用按键的判断，但没有添加任何内容，接下来的示例将以一个TextVariableBox为例，简述用户交互的使用。

首先在文件的头部添加#include "GUI\_Font.h"、“#include "GUI\_VariableBox.h"”宏定义，然后在全局声名部分添加如下代码。

根据前文所述，用户的代码主要在User文件夹下编写，工程默认在此文件夹下只有一个UserAction.c文件，文件中有两个函数。USR\_ACT\_OnInitialize函数将在LCD模拟器启动时被调用，可以在其中编写一些初始化显示的内容。USR\_ACT\_OnKeyPress函数用于响应用户的键盘消息。

那么接下来，我们将使用绘制文字API，在屏幕上显示一些文字，比如“Hello world!”。

打开UserAction.c文件，在文件的头部添加#include "GUI\_Font.h"宏定义，然后找到USR\_ACT\_OnInitialize函数，并在其中添加如下代码：

RECTANGLE stTextDisplayArea = {10, 40, 108, 14};

RECTANGLE tTextDataArea = {0, 0};

char szTextVarBuffer[21] = {"123456"};

GUI\_TXT\_VARBOX\_STRUCT stTextVar = {10, 10, 50, 0, 20, szTextVarBuffer};

然后在USR\_ACT\_OnInitialize函数中添加如下代码：

GUI\_TextVariableBox\_Refresh(&stTextVar, GUI\_DRAW\_NORMAL);

在USR\_ACT\_OnKeyPress函数的KEY\_VALUE\_UP分支下，添加如下代码：

case KEY\_VALUE\_UP:

{

GUI\_TextVariableBox\_ChangeCharacter(&stTextVar, GUI\_DRAW\_NORMAL, GUI\_TEXT\_ASCII, GUI\_TXT\_VARBOX\_OPT\_PREV);

break;

}

在KEY\_VALUE\_DOWN分支下，添加如下代码：

case KEY\_VALUE\_DOWN:

{

GUI\_TextVariableBox\_ChangeCharacter(&stTextVar, GUI\_DRAW\_NORMAL, GUI\_TEXT\_ASCII, GUI\_TXT\_VARBOX\_OPT\_NEXT);

break;

}

在KEY\_VALUE\_RIGHT分支下，添加如下代码：

case KEY\_VALUE\_DOWN:

{

if(stTextVar.FocusIndex < stTextVar.MaxTextLength)

{

stTextVar.FocusIndex++;

GUI\_TextVariableBox\_Refresh(&stTextVar, GUI\_DRAW\_NORMAL);

}

break;

}

在KEY\_VALUE\_LEFT分支下，添加如下代码：

case KEY\_VALUE\_DOWN:

{

if(stTextVar.FocusIndex > 0)

{

stTextVar.FocusIndex--;

GUI\_TextVariableBox\_Refresh(&stTextVar, GUI\_DRAW\_NORMAL);

}

break;

}

在KEY\_VALUE\_ENTER分支下，添加如下代码：

case KEY\_VALUE\_DOWN:

{

GUI\_Text\_DrawSingleLineText(stTextVar.Value, FONT\_SIZE\_H12, &stTextDisplayArea, &stTextDataArea, GUI\_DRAW\_NORMAL);

break;

}

代码的意义为，在坐标(10,10)处，显示一个宽度为50，初始内容为“123456”，最大可编辑程度为20的字符串值编辑框。允许编辑的字符为可见的基本ASCII字符。

点击“编译”按钮，重新编译，编译完成后运行，LCD模拟器上将显示一个文本编辑框。通过键盘上的方向箭头，可以选定和修改某一个文字，按下回车键，可以在位置(10,40)宽度108，高度14的区域内显示当前编辑框中的文字。

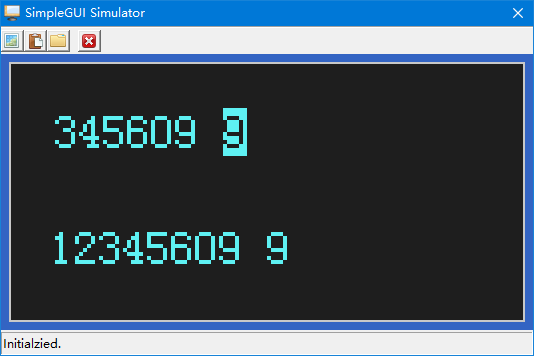


图 21 显示文字编辑框与编辑的文字

现在的界面看上去还是有点儿怪，看不出编辑框的具体范围，可以在编辑框和文本显示区域的外部绘制一个矩形，让GUI更美观一些。在文件的头部添加#include "GUI\_Font.h"宏定义，然后在USR\_ACT\_OnInitialize函数中追加以下代码：

GUI\_Basic\_DrawRectangle(9, 9, 52, 16, GUI\_COLOR\_FRGCLR, GUI\_COLOR\_TRANS);

GUI\_Basic\_DrawRectangle(9, 39, 110, 16, GUI\_COLOR\_FRGCLR, GUI\_COLOR\_TRANS);

再次编译后运行，可以看见，编辑框和文本均已经有了外框。

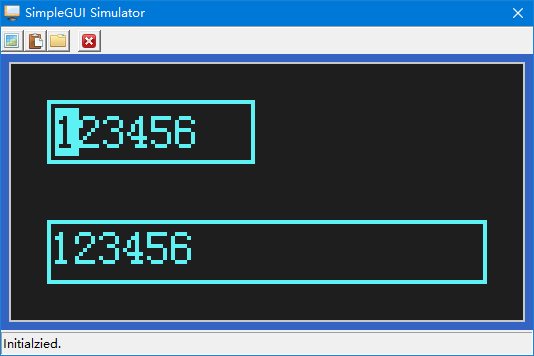


图 22 添加外框

至此，一个简单的SimpleGUI交互模拟已经完成，用户可以参照此原理，进行其他交互的开发、模拟与验证。

# 自定义拓展

也许根据现有的API，难以满足用户在开发过程中的需求，那么您也可以自己编写符合您需求的接口。

在SimpleGUI在GUI\_Basic.c文件和GUI\_Font文件中提供了基本的绘图与文字接口，用户可以根据这些API，在屏幕上绘制自己需要的内容。事实上，SimpleGUI的所有图形接口也都是根据这些基本API组合而来。

GUI\_Basic文件提供基本的绘图功能，包括以下API：

|  |  |
| --- | --- |
| GUI\_ClearScreen | 清空屏幕。 |
| GUI\_Basic\_DrawPoint | 绘制点。 |
| GUI\_Basic\_DrawLine | 绘制直线。 |
| GUI\_Basic\_DrawRectangle | 绘制矩形，可填充。 |
| GUI\_Basic\_DrawCircle | 绘制正圆，可填充。 |
| GUI\_Basic\_ReverseBlockColor | 反色矩形区域。 |
| GUI\_Basic\_DrawBitMap | 绘制位图。 |

GUI\_Font.c文件提供文字功能，包括以下API：

|  |  |
| --- | --- |
| GUI\_Text\_DrawSingleLineText | 在指定区域内绘制单行文本。 |
| GUI\_Text\_DrawMultipleLinesText | 在指定区域内绘制多行文本。 |

上述为截至目前为止所有的基本绘图API，以后还可能根据实际需求或用户反馈增加新的API。或者如果您只是想改变或增加一些接口的功能，也可以根据实际情况选择既有的API进行拓展绘制，而不用全部重绘。

# 平台移植