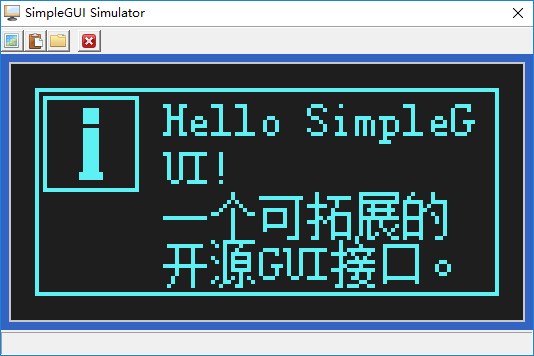
SimpleGUI

一套针对单色显示屏的开源GUI接口

**SimpleGUI API文档**



**开源，是一种态度。**

# 目录

[目录 2](#_Toc496102550)

[文件结构 3](#_Toc496102551)

[环境参数设置 4](#_Toc496102552)

[数据类型定义 5](#_Toc496102553)

[API 6](#_Toc496102554)

[模 13](#_Toc496102555)

[从Hello world开始。 14](#_Toc496102556)

[简单交互 17](#_Toc496102557)

[自定义拓展 20](#_Toc496102558)

[平台移植 21](#_Toc496102559)

# 文件结构

Todo

# 环境参数设置

为方便用户配置和移植，SimpleGUI在SGUI \_Config.h文件中定义了一些列的控制和开关宏，用户可以通过打开、关闭以及修改宏定义的值来对SimpleGUI的一些全局属性进行修改。

1. \_SIMPLE\_GUI\_ENABLE\_ICONV\_GB2312\_

此宏定义关联文字显示API对非ASCII文字的解码方式。设置值大于0时有效，文字相关的API接口将会视输入的文字为UTF-8格式并转换成GB2312格式进行解析，若此值为0，则视输入文字为GB2312格式进行解析。具体操作将会在“文字显示”章节详细阐述。

1. \_SIMPLE\_GUI\_VIRTUAL\_ENVIRONMENT\_SIMULATOR\_

此宏定义关联SimpleGUI的运行环境，设置值大于0时有效。当此宏定义有效，意味着SimpleGUI正运行于模拟环境中，用户在移植到目标平台后，请将此宏定义的值修改为0。

1. \_SIMPLE\_GUI\_ENABLE\_DYNAMIC\_MEMORY\_

此宏定义关联SimpleGUI移植目标平台的动态内存操作， 设置值大于0时有效。当此宏定义设置为有效时，意味着目标平台已实现了动态内存管理或支持动态内存管理，且用户已经做好相应的实现或移植，此时SimpleGUI的相关依赖内容也会被使能，如列表项目的动态增减。

1. \_SIMPLE\_GUI\_ENABLE\_BASIC\_FONT\_

此宏定义关联SimpleGUI的内置基础字体， 设置值大于0时有效。SimpleGUI内部存储了一组尺寸为6\*8像素的可见ASCII字符集数据，当此宏定义有效时，此套字体数据将被支持，且文字显示API将可以使用此文字数据显示基本ASCII字符内容。

此功能设计为，如果用户使用外部字库且外部字库出现损坏或数据异常时，可以使用此字库数据输出一些警告或调试信息。

# 数据类型定义

为避免因平台和编译器的差异造成的对基本数据类型的定义不同，进而导致代码产生不可预知的错误，SimpleGUI在SGUI\_Typedef.h文件中重新定义了包括整数、字符、字符指针在内的一系列基本数据类型，在SimpleGUI内部的代码实现中将一致性的使用重新定义过的数据类型，用户需要在使用前明确所在平台的数据类型定义并作出相应修改，以避免因溢出或其他异常导致的错误。

具体内容请参照SGUI\_Typedef.h文件。

# API

* + 1. **SGUI\_Common\_GetNowTime**

**功能描述**：获取当前特定系统时间。

**原型声明**：void SGUI\_Common\_GetNowTime(SGUI\_TIME\* pstTime)

**参数说明**：

pstTime：保存时间数据的结构体。

**返回值**：复制的字符串缓存头指针。

**注意事项**：此函数需绑定系统RTC的处理，需要用户更具目标平台自行实现，读取RTC时间并赋值到参数指定的RTC结构体中。

* + 1. **SGUI\_Common\_RefreshScreen**

**功能描述**：刷新屏幕显示。

**原型声明**：void SGUI\_Common\_RefreshScreen(void)

**参数说明**：无。

**返回值**：无。

**注意事项**：用于更新屏幕显示的接口，需要用户自行实现，通常用于使用了显示缓存的情况下。

* + 1. **SGUI\_Common\_ReadFlashROM**

**功能描述**：读取外部存储中的数据。

**原型声明**：void SGUI\_Common\_ReadFlashROM(SGUI\_ROM\_ADDRESS uiAddressHead, SGUI\_SIZE uiDataLength, SGUI\_BYTE\* pBuffer)

**参数说明**：

uiAddressHead：读取的首地址。

uiDataLength：读取数据的长度。

pBuffer：存放读取数据的缓冲区的首地址。

**返回值**：无。

**注意事项**：此函数需根据实际系统平台，实现从内部或外部Flash中读取数据的操作。在SimpleGUI中，此函数通常用于读取字模、图片或图标数据等。

* + 1. **SGUI\_Common\_Delay**

**功能描述**：延时等待。

**原型声明**：void SGUI\_Common\_Delay(SGUI\_UINT32 uiTimeMs)

**参数说明**：

uiTimeMs：等待的毫秒数。

**返回值**：无。

**注意事项**：根据实际系统平台实现的延时函数，用于HMI引擎中，绘图引擎不使用此函数。

* 1. **基础绘图**

基础绘图函数的实现位于SGUI\_Basic.c文件中，用于点、线、面和位图的绘制工作。

* + 1. **SGUI\_Basic\_ClearScreen**
    2. **SGUI\_Basic\_DrawPoint**
    3. **SGUI\_Basic\_DrawLine**
    4. **SGUI\_Basic\_DrawRectangle**
    5. **SGUI\_Basic\_DrawCircle**
    6. **SGUI\_Basic\_ReverseBlockColor**
    7. **SGUI\_Basic\_DrawBitMap**
  1. **文字显示**绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。
  2. **消息框**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

* 1. **输入框**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

* 1. **进度条**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

* 1. **滚动条**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

* 1. **列表框**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

* 1. **实时曲线**

绘图API的实现文件位于GUI文件夹中，主要负责显示屏硬件的控制与屏幕绘图。

1. **HMI API**
   1. 公共函数全部定义在SGUI\_Common.c文件中，主要负责SimpleGUI的全局共通处理和SimpleGUI与外部的数据交互。

。

。

# 模

# 从Hello world开始。

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。

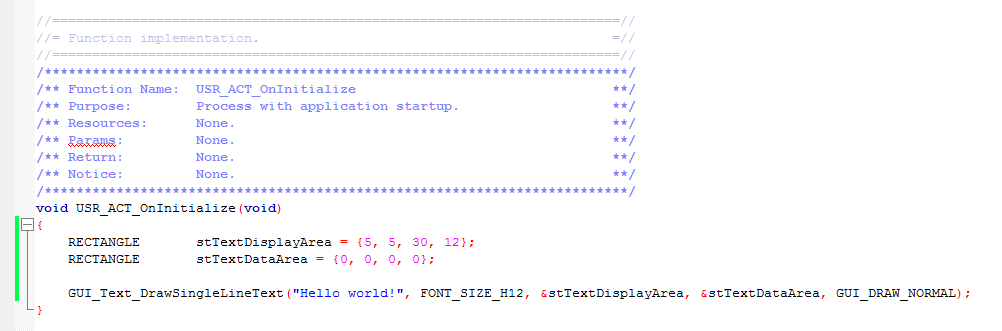


图 17 HelloWord代码

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。

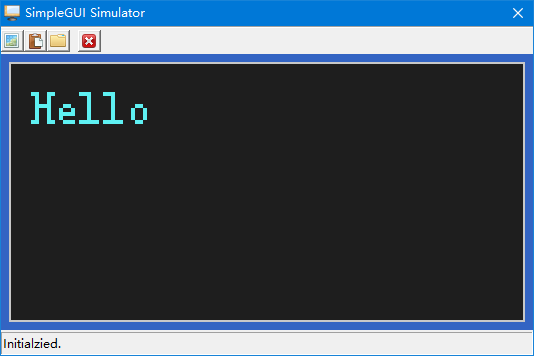


图 18 显示“Hello world”

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。

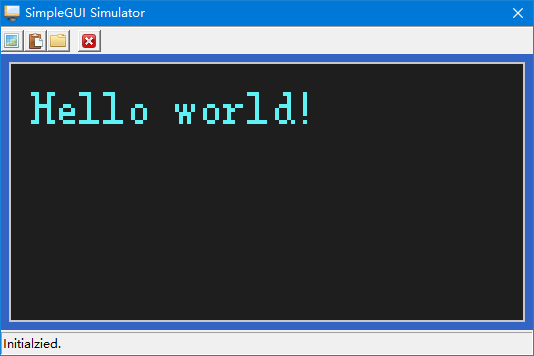


图 19 完整显示的“Hello world!”

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。

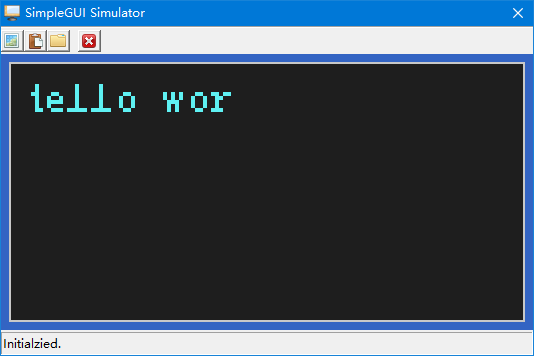


图 20 偏移数据显示

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。

# 简单交互

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。

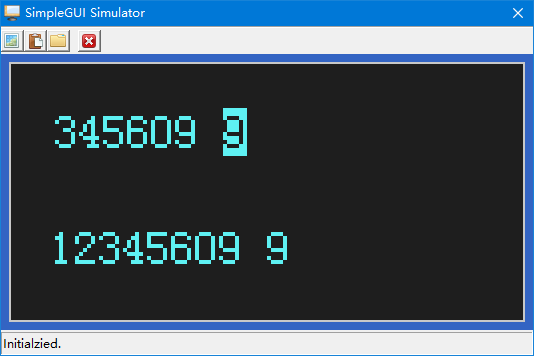


图 21 显示文字编辑框与编辑的文字

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。

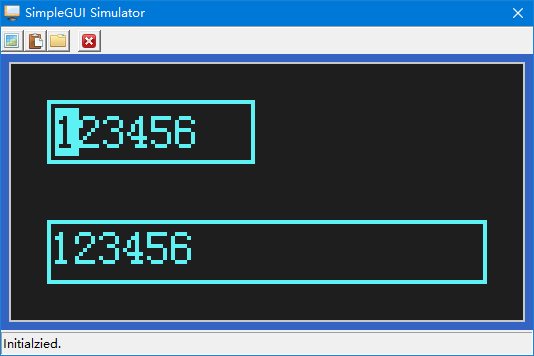


图 22 添加外框

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