

SimpleGUI

一套针对单色显示屏的开源 GUI 接口

SimpleGUI 概述



开源，是一种态度。

写在前面

其实想把这个发布出来也已经很久了。

最早做这个东西，仅仅就是在做一个项目时的需求，由于在此之前，作者很少处理有关嵌入式 GUI 的东西，但这个客户要求终端可独立操作，而且有比较复杂的参数设置，不得已，必须使用 GUI 进行支持，所以，这套 GUI 接口也可以说是一个“逼上梁山”的产物。最初有考虑过使用 stEmWin，也有说这个东西就是 uCGUI，具体的作者也没有考证过，就不得而知了。但是事实上，很多时候，特别是对于单色屏来说，并不需要例如 stEmWin 这样花哨的 GUI，需要的仅仅就是简单的显示文字、按钮、列表、文本框等，而且单色显示屏往往分辨率比较低，一屏内并不能显示太多的内容，所以对于复杂的图层、窗口等更是没有什么硬性需要，而且在彩屏上实现的那种华丽复杂的 GUI 系统也需要诸如扩展运存、外部存储等硬件支持，在有些小型、低成本的设计上市不需要的。

综上种种，我决定将自己使用的 GUI 接口代码重新整理、重写并开源公布，本着简约而不简单的目标，取名 SimpleGUI，也藉此希望使用 SimpleGUI 的用户能够轻松，顺利的开发出自己想要的系统。

因为在开发时发现的种种不便，于是我又着手开发了这套可以用于简单模拟单色液晶显示屏的模拟系统，大家可以根据自己的喜好与实际需求，对模拟显示屏的颜色、分辨率、像素点大小等进行调整。同时模拟环境提供截图保存与复制功能，方便使用者在必要时编辑与制作文档。具体的使用方法将在说明正文中详细说明。

本文将只对开发环境与显示屏模拟环境进行简要说明，关于 SimpleGUI 的 API 使用方法，将会在另外的开发文档中逐步更新并详细说明。

由于目前为止，SimpleGUI 尚由本人独立开发和维护，本人尚有家庭和工作需要顾及，故此难以投入全身心的开发，所以，您的意见和反馈可能不一定能得到及时的答复，如有怠慢，还请谅解。

最后，希望使用者能够将在使用中发现的 Bug 和您觉得不足的地方告知于我，电子邮箱如下：

326684221@qq.com

为了方便对您的反馈进行筛选与存档，如果您有 BUG 或建议需要反馈，请按照以下规则书写您的反馈邮件标题：

【SimpleGUI BUG 反馈/建议】 邮件标题内容

您的每一条 BUG 反馈与建议都将是 SimpleGUI 的一次飞跃。SimpleGUI 感谢您的关注与支持。

最后，真诚的希望鄙人拙作能够方便您的开发与学习。

搭建开发环境

SimpleGUI 代码遵循 ANSI C 标准进行开发，经可能摆脱硬件平台依赖。

SimpleGUI 单色 LCD 模拟开发环境（以下简称“LCD 模拟环境”或“模拟环境”）使用 CodeBlocks 集成开发环境，基于 wxWidgets GUI 框架开发，使用 MinGW 编译器进行编译与调试。关于 CodeBlocks、wxWidgets 与 MinGW 的详细信息，请用户自行了解，在此不做过多说明。

首先，需要下载 wxWidgets 源码与 CodeBlocks 集成开发环境。

wxWidgets 主页：<https://www.wxwidgets.org/>

CodeBlocks 主页：<http://www.codeblocks.org/>

截至作者发布此文当时，wxWidgets 最新发布版本为 3.1.0，CodeBlocks 最新发布版本为 16.01。此 LCD 模拟环境使用 wxWidgets3.1.0、CodeBlocks16.01 与 MinGW4.9.2 开发与调试。

1、安装 CodeBlocks

进入 CodeBlocks 官方网站,参照下图，进入 CodeBlocks 的下载页面，下载包含编译器的 CodeBlocks 版本，安装版或绿色版请依照自己的需求或喜好自行决定。

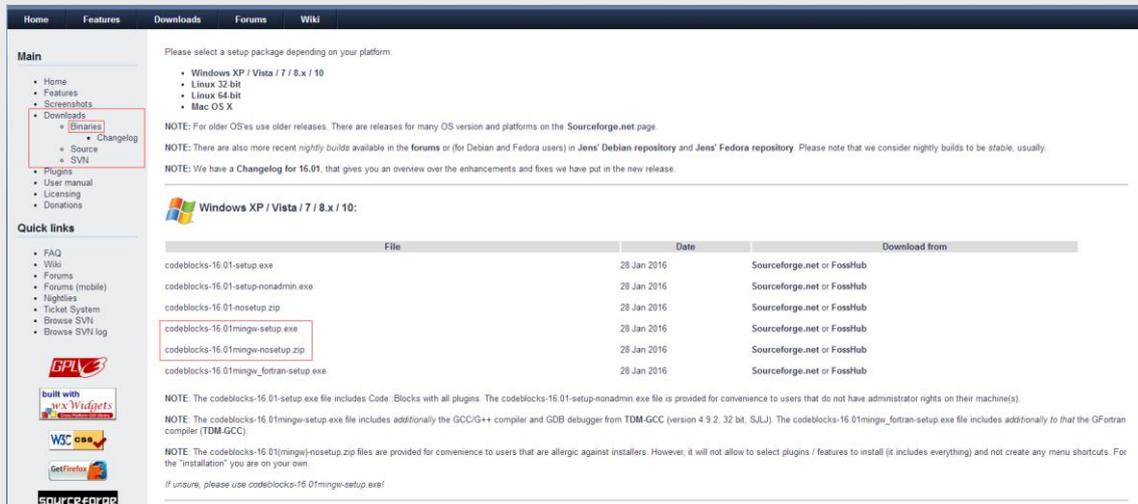


图 1 CodeBlocks 下载页面

下载后，安装或解压 CodeBlocks 即可，编译器默认已经配置为自带的 MinGW 编译器。CodeBlocks 16.01 版本自带 MinGW4.9.2 版本编译器，本 LCD 模拟环境均以此版本编译器便宜与调试，请勿换用其他版本，否则将导致不可预知的错误。

CodeBlocks 解压或安装完成后，需要设置编译器的环境变量，因为后面编译 wxWidgets 库时需要用到。为了方便说明，本文以下载 CodeBlocks 绿色版并解压至 D:\Software\CodeBlocks 路径下为例。其他操作类似，请悉知自己的 CodeBlocks 安装或解压的路径。

进入 CodeBlocks 根目录下后，可以看到一个名为 MinGW 的文件夹，此文件夹中便是我们要使用的编译器，如果没有此文件夹，则说明下载的 CodeBlocks 版本不正确或下载的文件损坏，请重新下载安装或解压。

明确 MinGW 的路径后，我们需要将 MinGW 的可执行文件路径添加至 Windows 环境变量中，以便在控制台下使用 MinGW。启动 Windows 命令行，然后执行以下命令：

```
set path=%path%;D:\Software\CodeBlocks\MinGW\bin
```

执行完毕后，MinGW 的路径将被系统加入环境变量。此时输入 mingw32-gcc -v，按回车执行后将会显示出 MinGW GCC 的版本，表明环境变量已注册成功。

```
C:\Users\xuyul>mingw32-gcc -v
Using built-in specs.
COLLECT_GCC=mingw32-gcc
COLLECT_LTO_WRAPPER=D:/Software/CodeBlocks/MinGW/bin/./libexec/gcc/mingw32/4.9.2/lto-wrapper.exe
Target: mingw32
Configured with: ../../src/gcc-4.9.2/configure --build=mingw32 --enable-languages=ada,c,c++,fortran,lto,objc,obj-c++
--enable-libgomp --enable-lto --enable-graphite --enable-libstdcxx-debug --enable-threads=posix --enable-version-specific
c-runtime-libs --enable-fully-dynamic-string --enable-libstdcxx-threads --enable-libstdcxx-time --with-gnu-ld --disable-
werror --disable-nls --disable-win32-registry --disable-symvers --enable-cxx-flags='-fno-function-sections -fno-data-sec
tions -DWINPTHREAD_STATIC' --prefix=/mingw32tdm --with-local-prefix=/mingw32tdm --with-pkgversion=tdm-1 --enable-sjlj-ex
ceptions --with-bugurl=http://tdm-gcc.tdragon.net/bugs
Thread model: posix
gcc version 4.9.2 (tdm-1)

C:\Users\xuyul>
```

图 2 MinGW 环境变量注册成功

2、 编译 wxWidgets 库

进入 wxWidgets 官方网站，参照下图进入 wxWidgets 的下载页面并下载 wxWidgets 源码。链接列表中的 Windows ZIP 和 Windows 7Z 中的内容是相同的，推荐下载 7Z，文件体积小，可以更快完成下载。

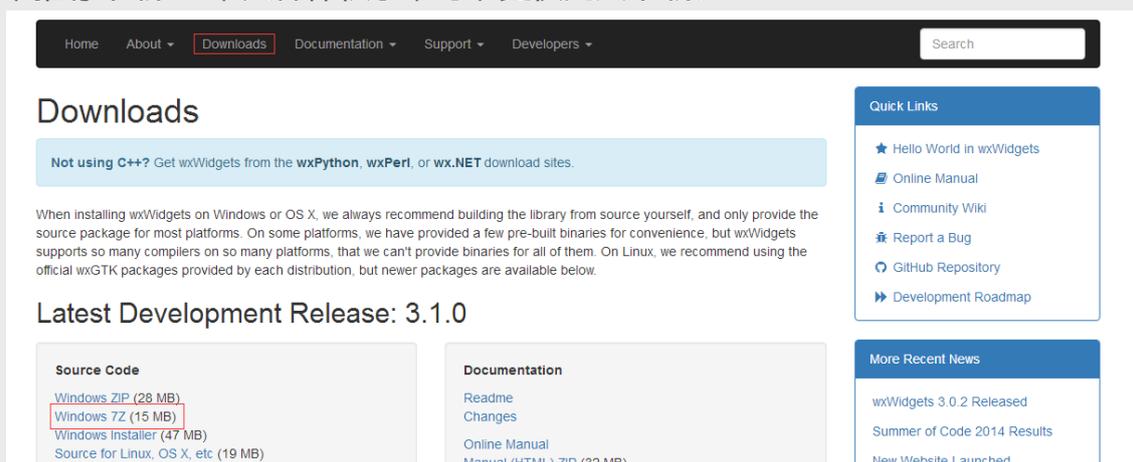


图 3 wxWidgets 下载页面

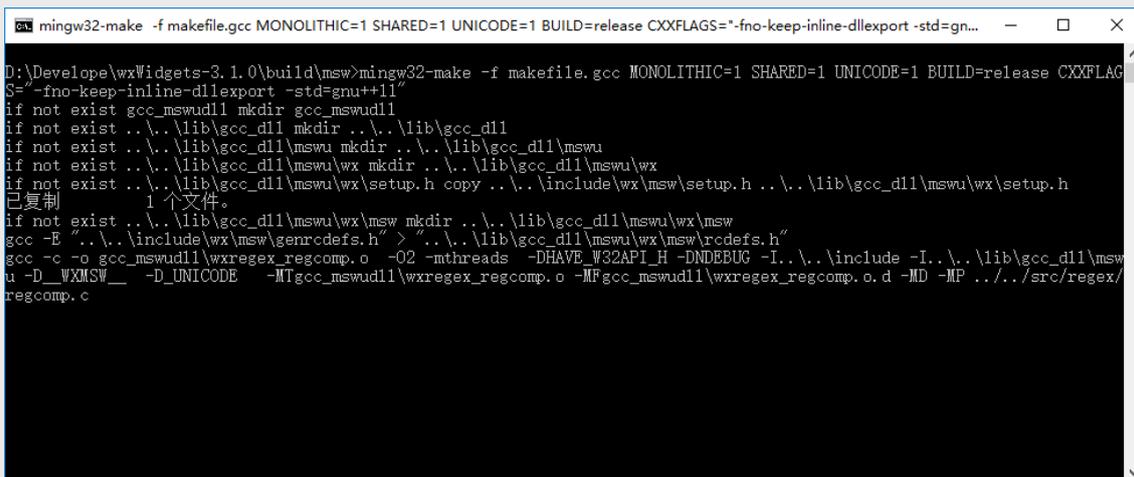
下载完成后，请将 wxWidgets 源码解压至一个固定路径，准备编译必要的库文件。需要注意的是，存放 wxWidgets 的路径，推荐为一个不常移动的路径，因为后文在配置 CodeBlocks 开发环境时，需要在 CodeBlocks 中设定 wxWidgets 的全局变量且不能使用相对路径。

为了方便说明，本文演示将 wxWidgets 解压至 D:\Develope\wxWidgets-3.1.0\路径下。

解压完成后，使用控制台，进入 wxWidgets 路径下的 build\msw 子目录，并执行以下命令，开始编译 wxWidgets，注意以下为一条命令，没有换行：

```
mingw32-make -f makefile.gcc MONOLITHIC=1 SHARED=1 UNICODE=1 BUILD=release CXXFLAGS="-fno-keep-inline-dllexport -std=gnu++11"
```

此步骤需要用到 MinGW 编译器，所以请确保之前注册 MinGW 环境变量注册成功且正确。

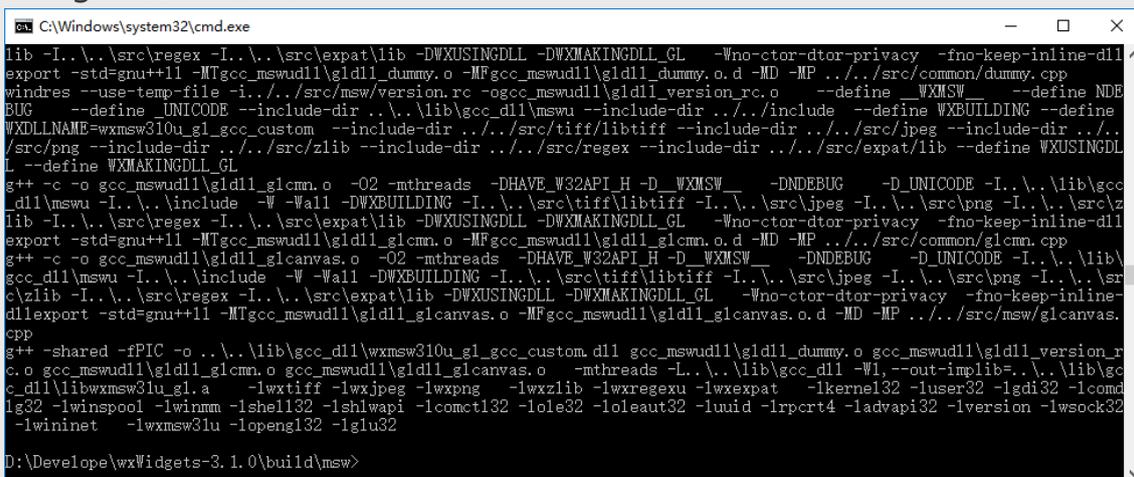


```
mingw32-make -f makefile.gcc MONOLITHIC=1 SHARED=1 UNICODE=1 BUILD=release CXXFLAGS="-fno-keep-inline-dllexport -std=gn...
D:\Develope\wxWidgets-3.1.0\build\msw>mingw32-make -f makefile.gcc MONOLITHIC=1 SHARED=1 UNICODE=1 BUILD=release CXXFLAG
S="-fno-keep-inline-dllexport -std=gnu++11"
if not exist gcc_mswudll1 mkdir gcc_mswudll1
if not exist ..\..\lib\gcc_dll mkdir ..\..\lib\gcc_dll
if not exist ..\..\lib\gcc_dll\mswu mkdir ..\..\lib\gcc_dll\mswu
if not exist ..\..\lib\gcc_dll\mswu\wx mkdir ..\..\lib\gcc_dll\mswu\wx
if not exist ..\..\lib\gcc_dll\mswu\wx\setup.h copy ..\..\include\wx\msw\setup.h ..\..\lib\gcc_dll\mswu\wx\setup.h
已复制
1 个文件。
if not exist ..\..\lib\gcc_dll\mswu\wx\msw mkdir ..\..\lib\gcc_dll\mswu\wx\msw
gcc -E ..\..\include\wx\msw\genrcdefs.h > ..\..\lib\gcc_dll\mswu\wx\msw\rcdefs.h"
gcc -c -o gcc_mswudll1\wxregex_regcomp.o -O2 -mthreads -DHAVE_W32API_H -DNDEBUG -I....\include -I....\lib\gcc_dll\msw
u -D_WXMSW__ -D_UNICODE -MTgcc_mswudll1\wxregex_regcomp.o -MFgcc_mswudll1\wxregex_regcomp.o.d -MD -MP ..\..\src\regex/
regcomp.c
```

图 4 开始编译 wxWidgets

由于 wxWidgets 的限制，只能使用特定版本的 MinGW 编译器编译，所以再次强调，请使用 CodeBlocks 自带的 MinGW4.9.2 编译器，否则将由于 wxWidgets 库的原因产生不可预知的错误。

编译大约需要 20-30 分钟，视电脑性能而定。至最后编译完成，没有提示错误，wxWidgets 库生成完毕。



```
C:\Windows\system32\cmd.exe
lib -I....\src\regex -I....\src\expat\lib -DWXUSINGDLL -DWXMAKINGDLL_GL -Wno-ctor-dtor-privacy -fno-keep-inline-dll
export -std=gnu++11 -MTgcc_mswudll1\gldll_dummy.o -MFgcc_mswudll1\gldll_dummy.o.d -MD -MP ..\..\src\common\dummy.cpp
windres --use-temp-file -i....\src\msw\version.rc -ogcc_mswudll1\gldll_version.rc.o --define WXMSW__ --define NDE
BUG --define UNICODE --include-dir ..\..\lib\gcc_dll\mswu --include-dir ..\..\include --define WXBUILDING --define
WXDLLNAME=wxmsw310u_g1_gcc_custom --include-dir ..\..\src\tiff\libtiff --include-dir ..\..\src\jpeg --include-dir ..\..
\src\png --include-dir ..\..\src\zlib --include-dir ..\..\src\regex --include-dir ..\..\src\expat\lib --define WXUSINGD
L --define WXMAKINGDLL_GL
g++ -c -o gcc_mswudll1\gldll_g1cmn.o -O2 -mthreads -DHAVE_W32API_H -D_WXMSW__ -DNDEBUG -D_UNICODE -I....\lib\gcc
_dll\mswu -I....\include -W -Wall -DWXBUILDING -I....\src\tiff\libtiff -I....\src\jpeg -I....\src\png -I....\src\z
lib -I....\src\regex -I....\src\expat\lib -DWXUSINGDLL -DWXMAKINGDLL_GL -Wno-ctor-dtor-privacy -fno-keep-inline-dll
export -std=gnu++11 -MTgcc_mswudll1\gldll_g1cmn.o -MFgcc_mswudll1\gldll_g1cmn.o.d -MD -MP ..\..\src\common\g1cmn.cpp
g++ -c -o gcc_mswudll1\gldll_g1canvas.o -O2 -mthreads -DHAVE_W32API_H -D_WXMSW__ -DNDEBUG -D_UNICODE -I....\lib\
gcc_dll\mswu -I....\include -W -Wall -DWXBUILDING -I....\src\tiff\libtiff -I....\src\jpeg -I....\src\png -I....\sr
c\zlib -I....\src\regex -I....\src\expat\lib -DWXUSINGDLL -DWXMAKINGDLL_GL -Wno-ctor-dtor-privacy -fno-keep-inline-
dllexport -std=gnu++11 -MTgcc_mswudll1\gldll_g1canvas.o -MFgcc_mswudll1\gldll_g1canvas.o.d -MD -MP ..\..\src\msw\g1canvas.
cpp
g++ -shared -fPIC -o ..\..\lib\gcc_dll\wxmsw310u_g1_gcc_custom.dll gcc_mswudll1\gldll_dummy.o gcc_mswudll1\gldll_version_r
c.o gcc_mswudll1\gldll_g1cmn.o gcc_mswudll1\gldll_g1canvas.o -mthreads -L....\lib\gcc_dll -Wl,--out-implib=..\lib\gc
c_dll\libwxmsw31u_g1.a -lwtiff -lwjpeg -lwpng -lwzlib -lwregex -lwexpat -lkernel32 -luser32 -lgdi32 -lcomd
lg32 -lwinpool -lwinmm -lshell32 -lshlwapi -lcomctl32 -lole32 -loleaut32 -luuid -lrpcrt4 -ladvapi32 -lversion -lwsock32
-lwininet -lwmsw31u -lopeng132 -lglu32
D:\Develope\wxWidgets-3.1.0\build\msw>
```

图 5 wxWidgets 编译完成

3、配置开发环境

启动 CodeBlocks，点击 CodeBlocks 的 Setting 菜单，选择 Global varibal 项目，进入全局变量设定窗口。

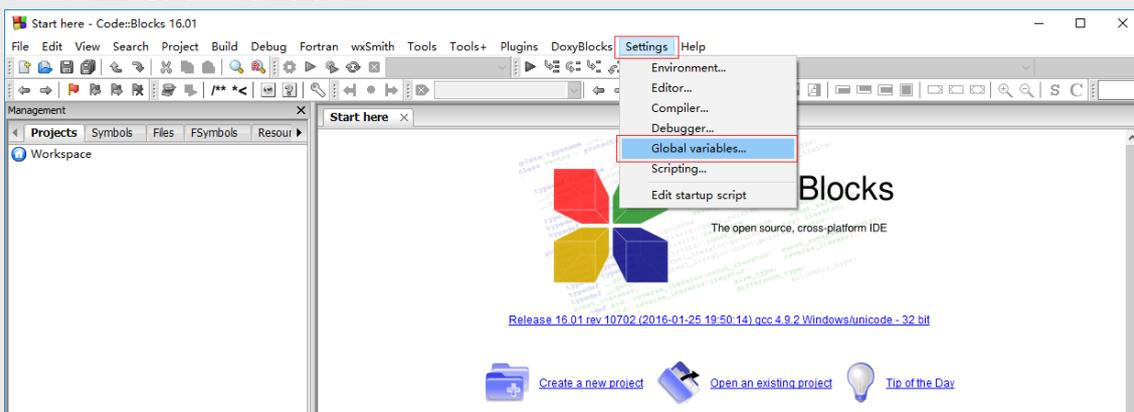


图 6 进入 Global variable 设定-1

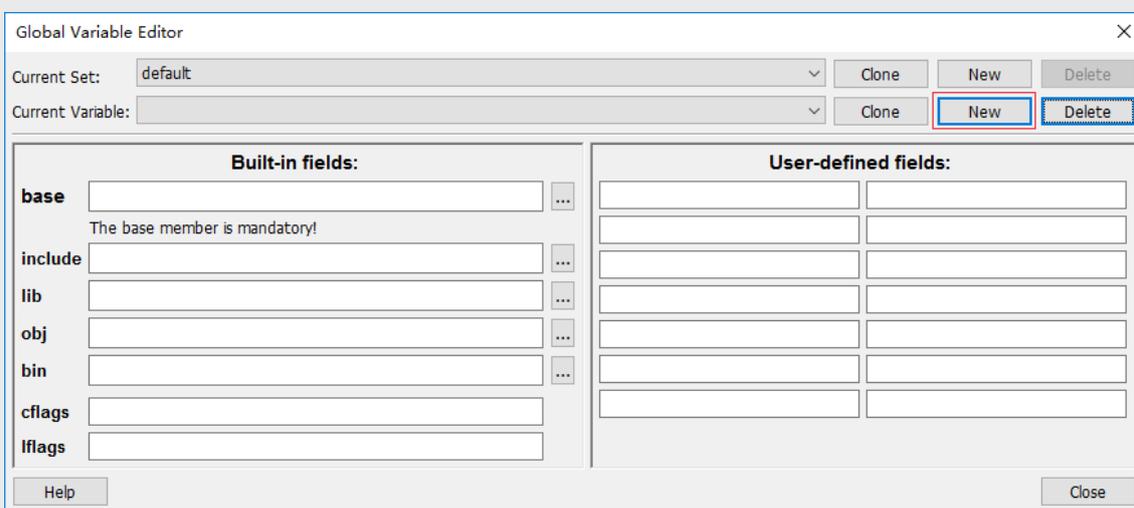


图 7 进入 Global variable 设定-2

进入上图窗口，点击右上角 Current Variable 项目后的 New 按钮，新建一个名为 wx31 的全局变量。

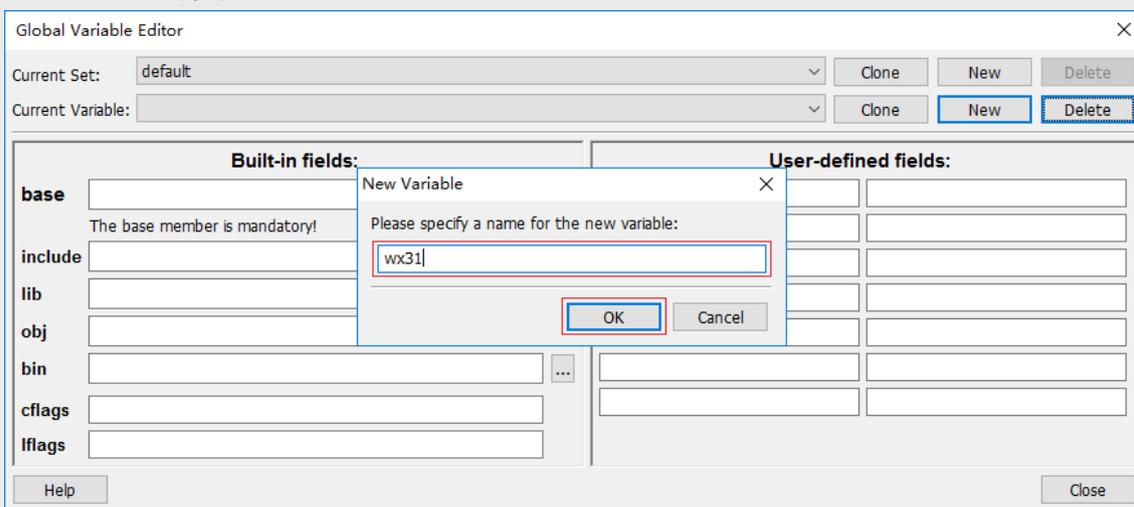


图 8 图 7 设定全局变量-1

然后根据 wxWidgets 库的具体存放位置，设置全局变量的 base、include 和 lib 三项的具体路径，base 为 wxWidgets 的根目录，include 为 wxWidgets 根目录下的 include 文件夹，lib 为 wxWidgets 根目录下的 lib 文件夹。

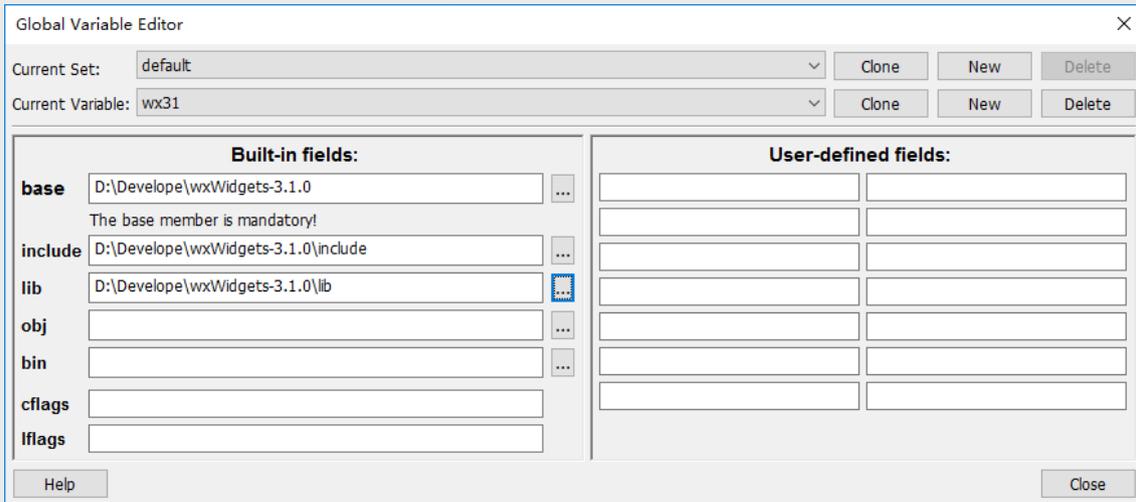


图 9 设定全局变量-2

设定完成后，点击 Close 按钮，退出全局变量窗口。
至此，LCD 模拟环境的开发环境构建完毕。

认识 SimpleGUI 模拟环境工程

在使用 LCD 模拟环境之前，首先熟悉一下工程的目录结构。

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。
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。

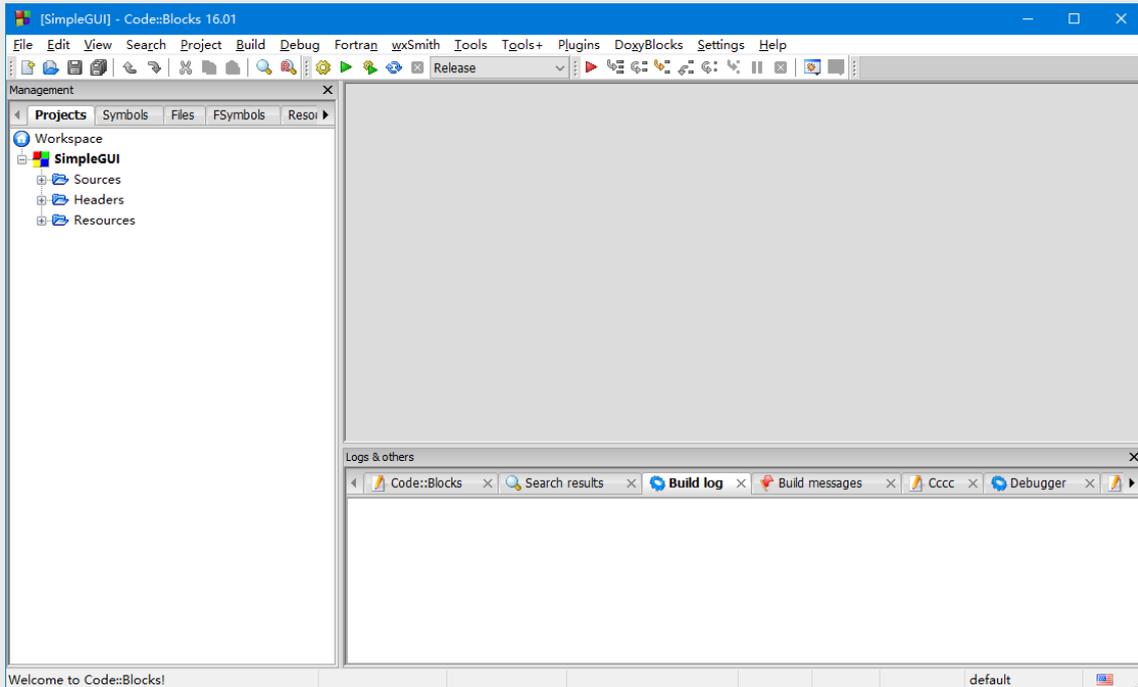


图 10 打开 SimpleGUI 工程

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。

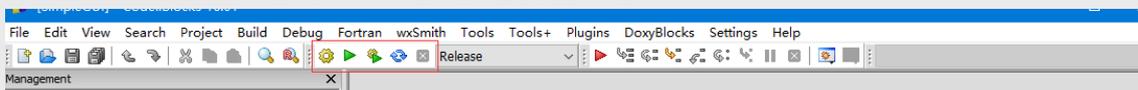


图 11 编译与执行

点击“编译”按钮开始编译，下方的消息窗口会即时显示编译的情况。

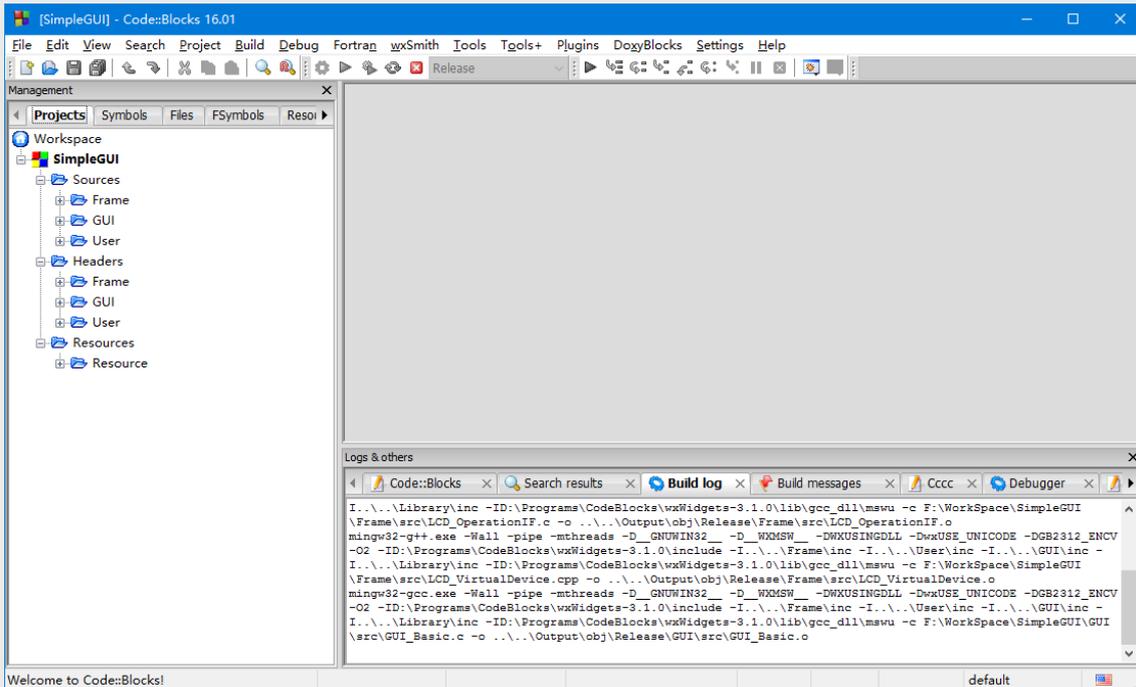


图 12 编译

编译完成后，下方显示 0 错误，0 警告，表示编译正确完成。

本文将只对开发环境与显示屏模拟环境进行简要说明，关于 SimpleGUI 的 API 使用方法，将会在另外的开发文档中逐步更新并详细说明。

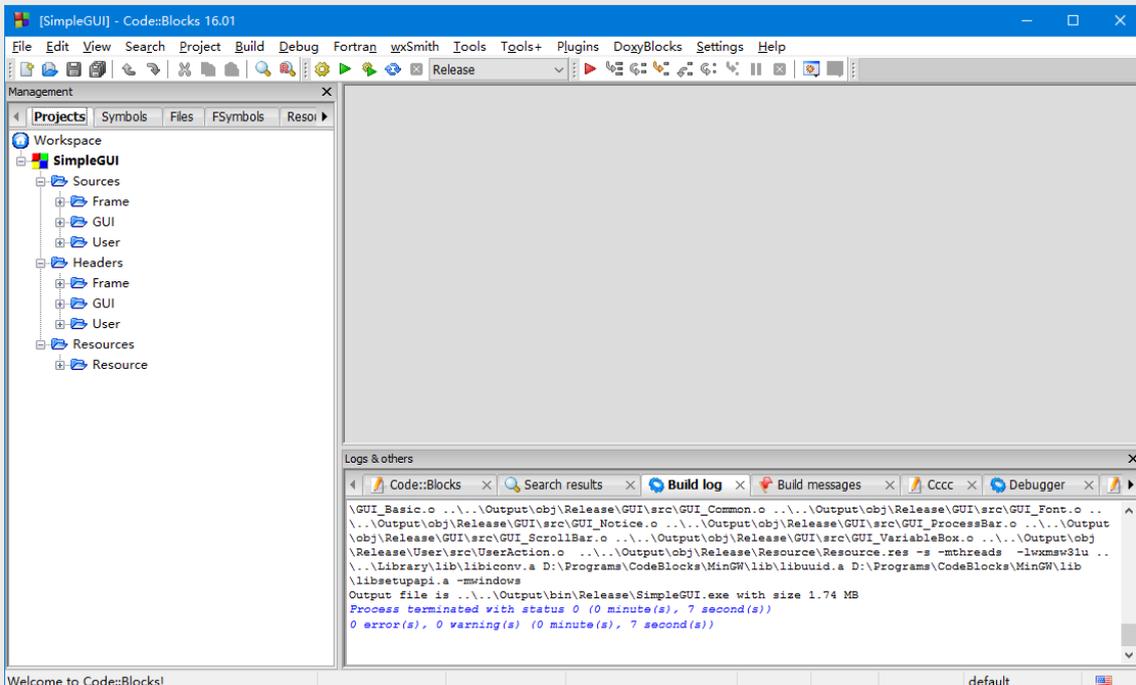


图 13 完成编译

编译完成后，按下“运行”（绿色向右箭头）按钮，即可运行 LCD 模拟环境。

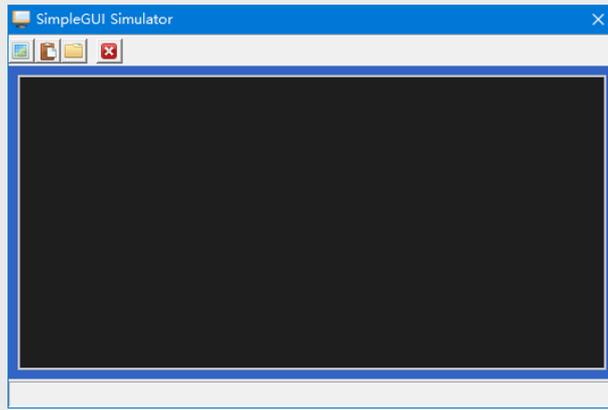


图 14 LCD 模拟器运行

由于现在还没有添加其他功能性代码，屏幕显示尚为空白。

模拟器配置

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。

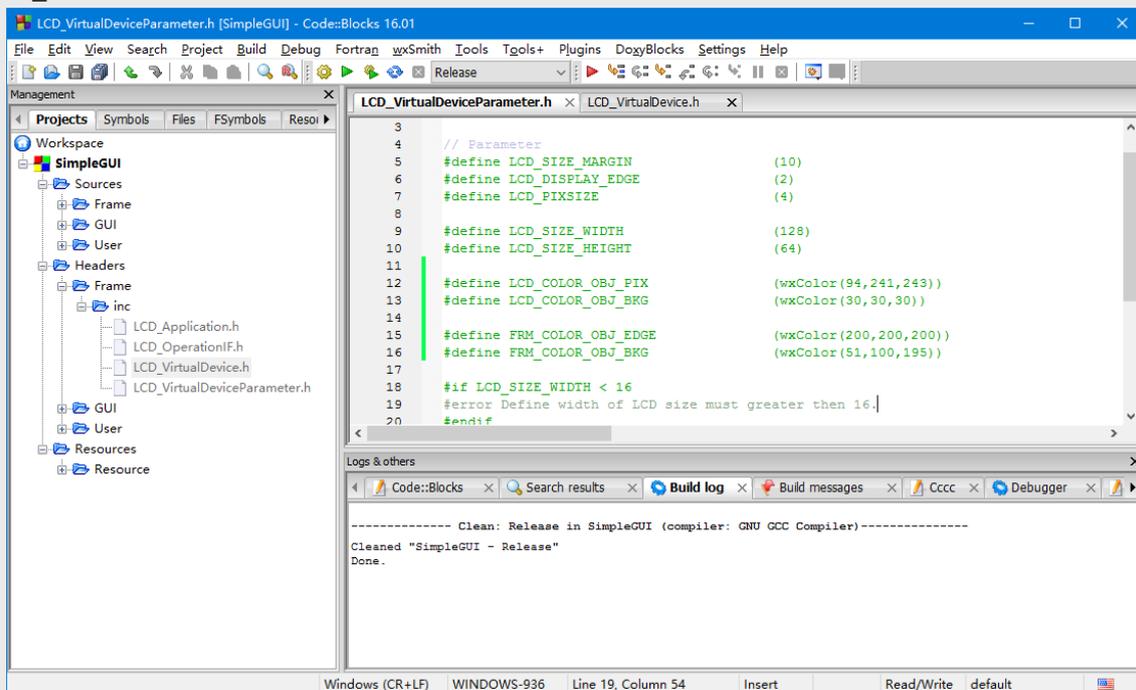


图 15 LCD 模拟器参数配置文件。

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_PIXEL_SIZE	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	(wxColor(88,120,20)) // 像素颜色(RGB)
LCD_COLOR_OBJ_BKG	(wxColor(116,164,25)) // 屏幕背景色(RGB)
LCD_COLOR_OBJ_EDGE	(wxColor(208,246,28)) // 边缘背景色(RGB)

重新编译后运行，即可看到变化效果。

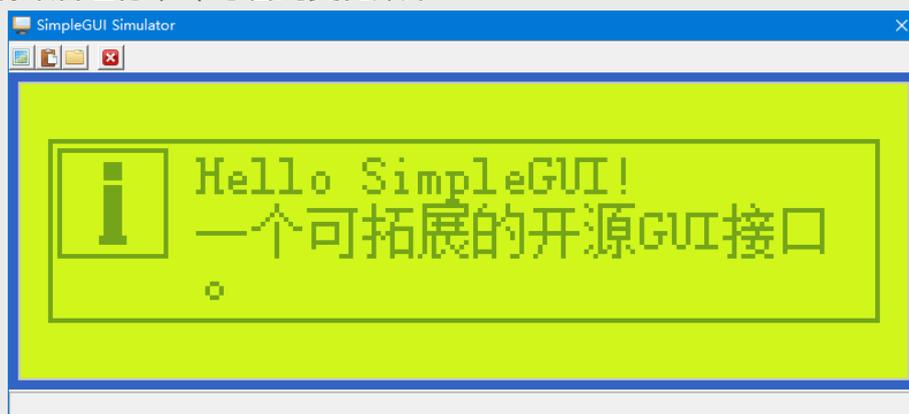


图 16 修改模拟器的配色方案

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。

```
=====
// Function implementation.
=====
/** Function Name:  USR_ACT_OnInitialize
** Purpose:        Process with application startup.
** Resources:      None.
** Params:         None.
** Return:         None.
** Notice:         None.
**/
void USR_ACT_OnInitialize(void)
{
    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);
}
```

图 17 HelloWorld 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 world!”文字。

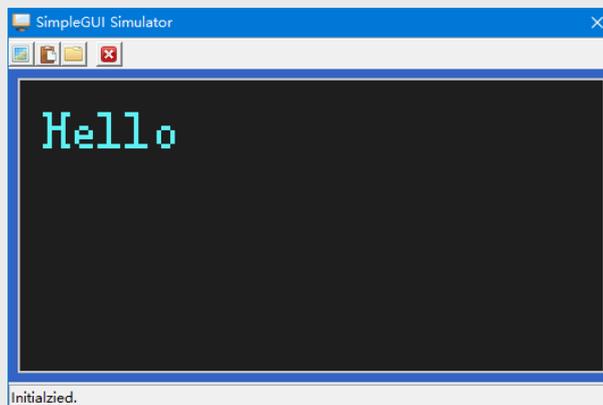


图 18 显示 “Hello world”

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。

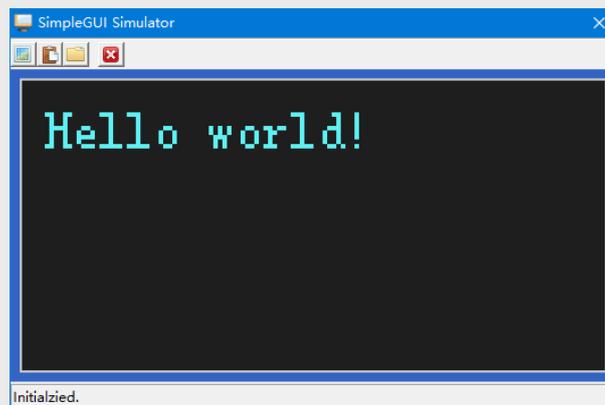


图 19 完整显示的“Hello world!”

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。

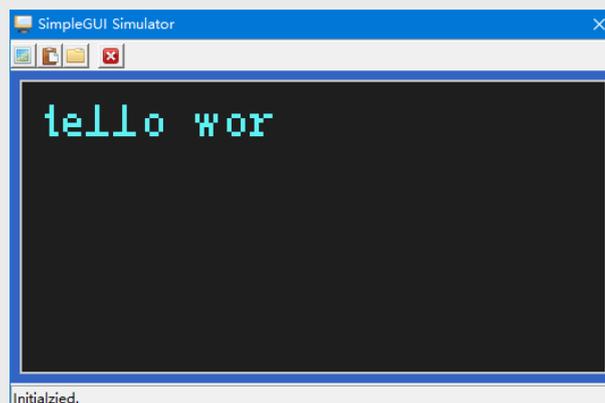


图 20 偏移数据显示

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。

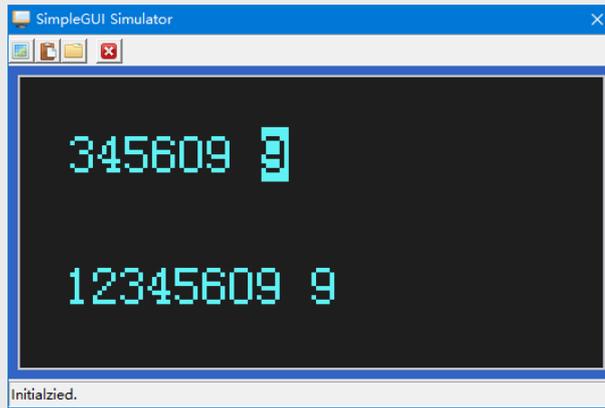


图 21 显示文字编辑框与编辑的文字

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。

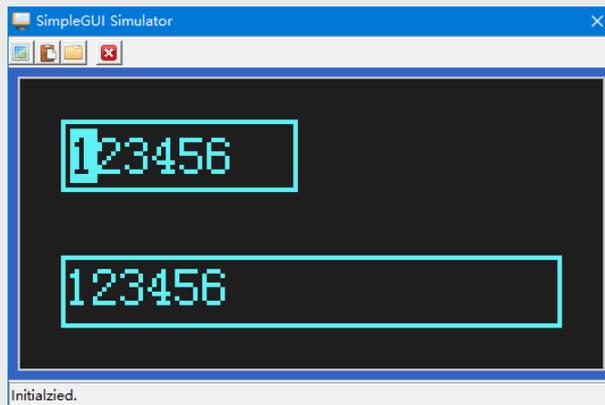


图 22 添加外框

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

SimpleGUI 尽可能的将实现层与驱动层隔离开来，用以使移植工作最简化。本例将以 STM32F103ZET6 与 KS0108 19264 LCD 显示屏为例，演示如何将 SimpleGUI 移植到一个 Keil MDK 5 的 STM32 工程中。

作为最简化移植，用户只需要实现“读像素”和“写像素”两个函数，就可以使用 SimpleGUI 了，当然，您如果需要对显示效果及用户体验有进一步的改善，还可以针对使用的芯片、系统以及显示控制器等对程序进行深层优化。

在对 SimpleGUI 进行移植与配置之前，用户需要提前做一些准备工作，比如建立一个目标平台的工程然后编写您想使用的屏幕的驱动程序，并按照上述所说，实现显示屏上像素的读函数与写函数。

由于各人所用的平台与显示屏千差万别，此过程请根据实际需求，参照其他资料实现，此处不做赘述。

上述准备工作完成后，进入 SimpleGUI 的根目录，找到 GUI 文件夹，并复制到您的工程目录下。

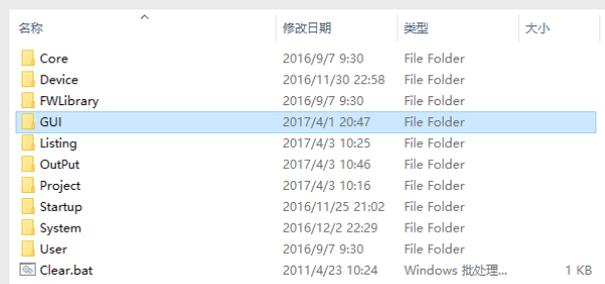


图 23 复制 GUI 文件夹到工程目录下。

然后打开建立好的 MDK5 工程，新建一个组(Group)，名称任意，此例命名为 GUI，然后将 GUI 文件夹下 src 子文件夹中的所有.c 文件添加到组中。

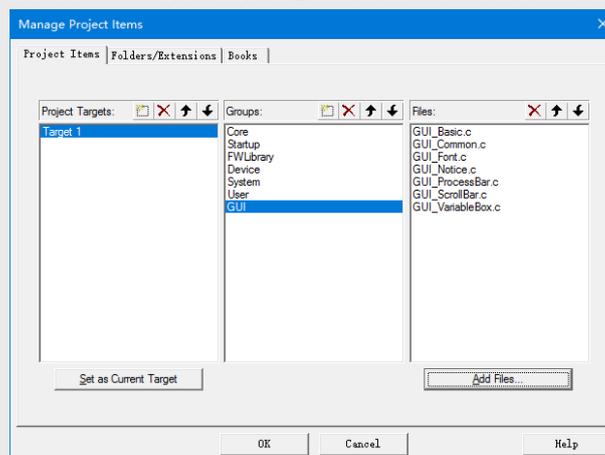


图 24 添加 SimpleGUI 文件到工程中

建立好的工程结构如下：

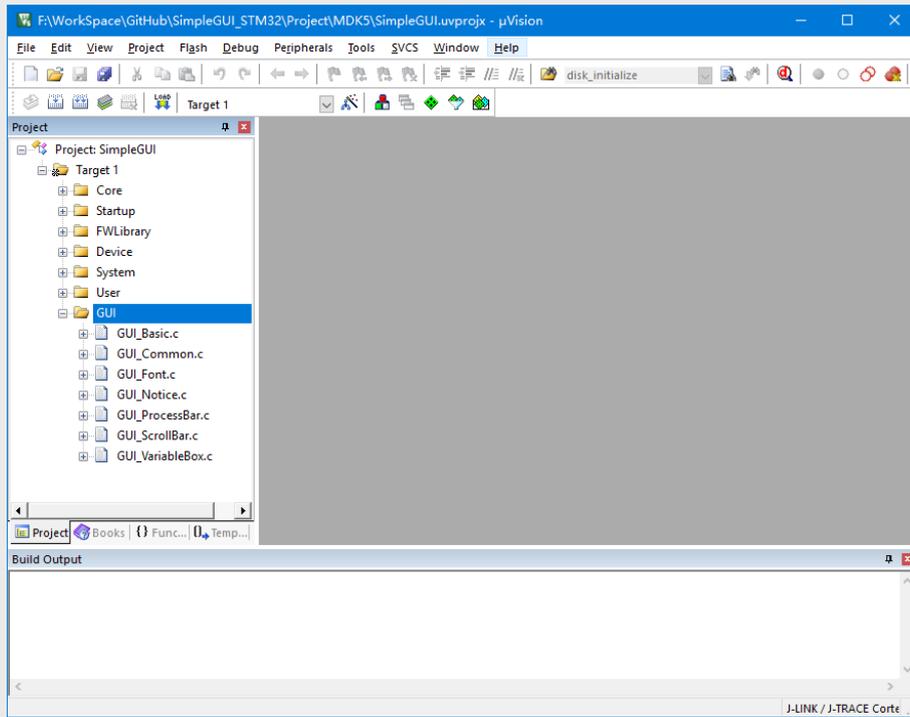


图 25 建立好的 MDK5 工程

添加文件后，还需要添加文件包含路径，打开工程配置对话框，然后在文件包含路径中添加 GUI 文件夹下的 inc 文件夹路径。

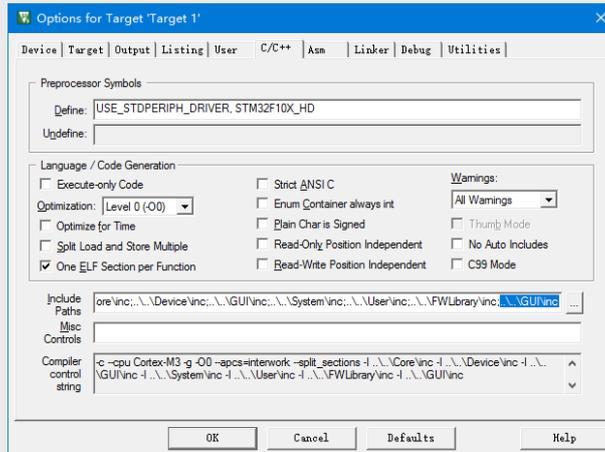


图 26 添加文件包含路径

至此，关于工程的配置已经结束，接下来需要将 SimpleGUI 与用户的屏幕驱动程序进行整合。

首先，确保您的显示屏驱动程序与屏幕像素的读写函数已经实现且没有错误，本例使用 KS0108 主控制器的 19264 显示屏，实现文件如下。

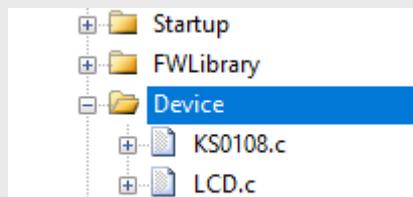


图 27 STM32 的 KS0108 驱动程序举例

需要注意的是，用户编写驱动程序时，需要在头文件中定义以下两个宏：

```
#define LCD_SIZE_WIDTH      192
#define LCD_SIZE_HEIGHT     64
```

这两个宏定义为 LCD 显示屏的宽度与高度，单位为像素，由于 SimpleGUI 中，对这两个宏定义值有多处引用，所以建议用户不要对这两个宏定义的名称进行修改，否则将需要对 SimpleGUI 中所有应用处进行修改，改动量大且容易引起错误。

接下来，打开 GUI_Basic.h 文件，在文件头处，包含外部引用文件处，找到以下代码；

```
#ifndef __SIMULATOR__
#include "LCD_OperationF.h"
#else
// Insert screen driver head file here.
#endif
```

在“Insert screen driver head file here.”处，添加对用户显示屏驱动程序头文件的引用，本例中为 LCD.h，修改完成后如下。

```
#ifndef __SIMULATOR__
#include "LCD_OperationF.h"
#else
#include "LCD.h"
#endif
```

然后打开 GUI_Basic.c 文件，找到 GUI_Basic_DrawPoint、GUI_Basic_GetPoint 与 GUI_ClearScreen 三个函数。

GUI_Basic_DrawPoint 中，找到以下代码

```
if(GUI_COLOR_FRGCLR == eColor)
{
#ifdef __SIMULATOR__
    VTIF_SetPoint(uiPosX, uiPosY, 1);
#else
    // Call draw pix interface here.
#endif
}
else if(GUI_COLOR_BKGCLR == eColor)
{
#ifdef __SIMULATOR__
    VTIF_SetPoint(uiPosX, uiPosY, 0);
#else
    // Call draw pix interface here.
#endif
}
```

在绿色注释处，添加对用户编写的写像素接口的函数调用，两处分别为写入像素（亮点）和擦除像素（灭点）的调用。移植后的函数代码如下。

```

if(GUI_COLOR_FRGCLR == eColor)
{
#ifdef __SIMULATOR__
    VTIF_SetPoint(uiPosX, uiPosY, 1);
#else
    LCD_SetPixel(uiPosX, uiPosY, LCD_COLOR_BLACK);
#endif
}
else if(GUI_COLOR_BKGCLR == eColor)
{
#ifdef __SIMULATOR__
    VTIF_SetPoint(uiPosX, uiPosY, 0);
#else
    LCD_SetPixel(uiPosX, uiPosY, LCD_COLOR_WHITE);
#endif
}

```

同理，在 GUI_Basic_GetPoint 与 GUI_ClearScreen 函数中，添加于相应位置添加用户的读像素函数与清屏幕函数。

上述修改处的代码，使用“__SIMULATOR__”宏定义判断，在 SimpleGUI 模拟环境工程中，由于在工程中有此宏定义声明，所以代码调用的是模拟环境下模拟显示屏的操作函数。而在移植后，用户的工程中没有对此宏定义进行声明，所以将会编译另外的分支，调用用户编写的驱动程序函数。

至此，SimpleGUI 完成了在 STM32+KS0108 在 MDK5 工程上的移植，接下来，在主入口函数（一般为 main 函数）中添加一小段测试代码：

```

int main(void)
{
    Initialize_HSEClocks(RCC_PLLMul_9);
    Initialize_NVIC(NVIC_PriorityGroup_2);
    Initialize_DebugPort(DEBUG_SWD);

    // Initialize LCD device.
    LCD_Initialize();
    // Clear screen.
    GUI_ClearScreen();
    // Display a notice box.
    GUI_Notice_ShowNotice("Hello SimpleGUI!\n 可拓展开源嵌入式 GUI 接口。", 0,
    ICON_INFORMATION);
    // Refresh LCD.
    LCD_RefreshScreen();

    while(1)
    {
    }
}

```

点击编译按钮或按快捷键“F7”开始编译，直至消息框中显示编译完成，没有错误和警告。

```
linking...
Program Size: Code=5872 RO-data=230048 RW-data=36 ZI-data=6844
"..\\..\\OutPut\\SimpleGUI.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:07
```

图 28 编译完成

接下来，将程序下载到目标板，如果驱动程序和线路连接都没有错误，您将会看到，屏幕上出现了一个消息框，消息框中就是调用 GUI_Notice_ShowNotice 函数时在函数参数表中传入的文字。

```
Device: STM32F103ZE
VTarget = 3.300V
State of Pins:
TCK: 0, TDI: 0, TDO: 1, TMS: 0, TRES: 1, TRST: 1
Hardware-Breakpoints: 6
Software-Breakpoints: 8192
Watchpoints: 4
JTAG speed: 4000 kHz

Erase Done.
Programming Done.
Verify OK.
Application running ...
Flash Load finished at 12:40:26
```

图 29 下载程序到目标板



图 30 显示效果实拍

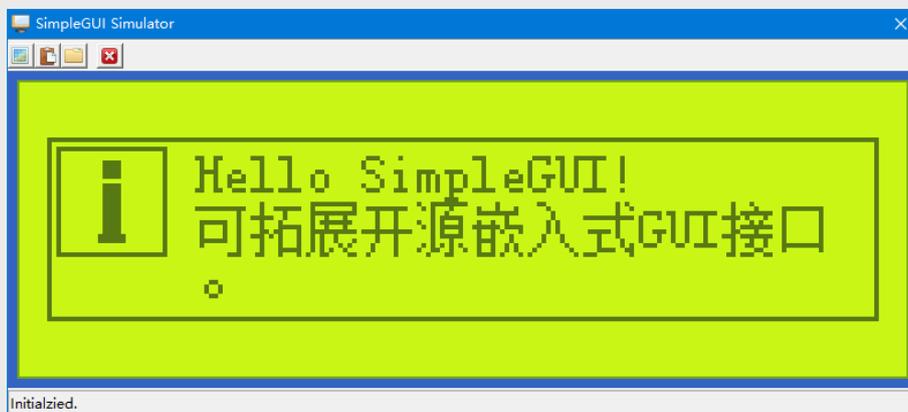


图 31 相同代码在模拟环境中的效果

