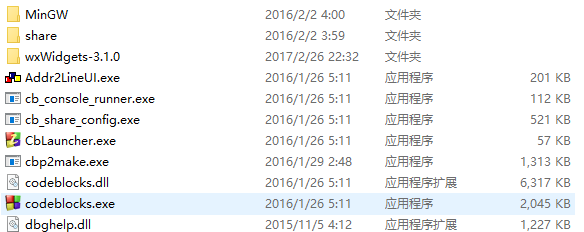
1. **环境搭建**

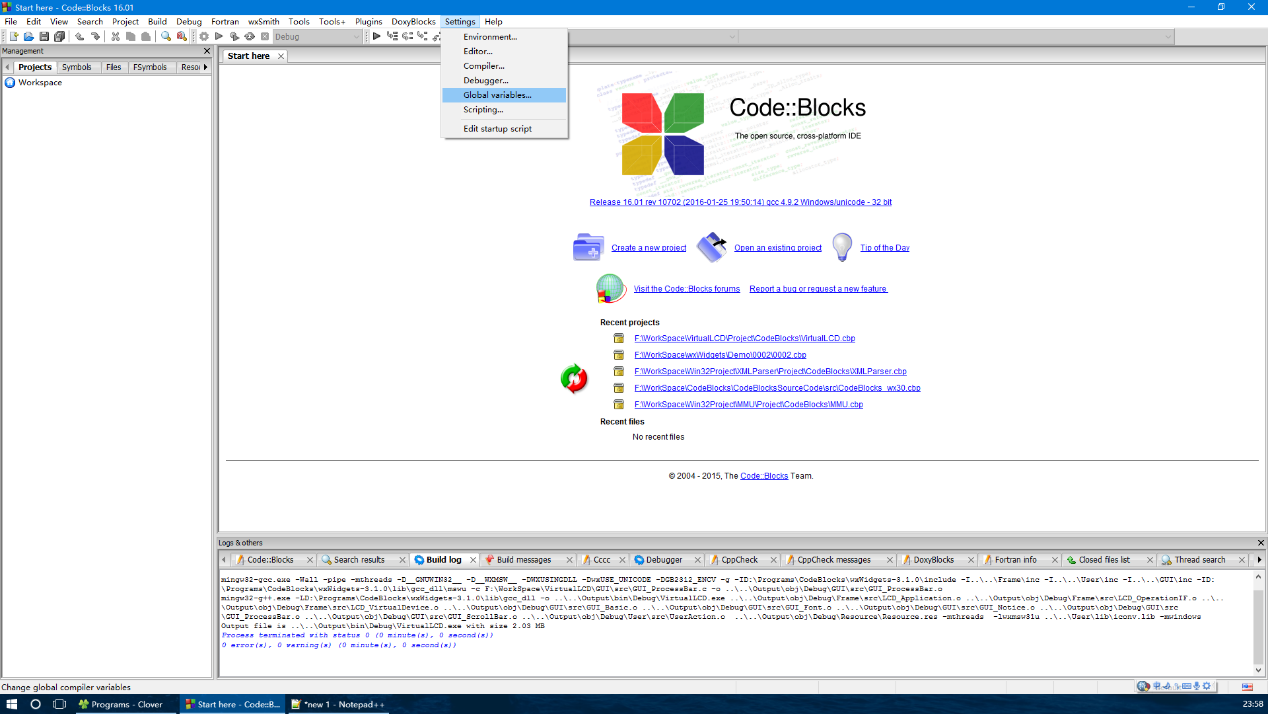
解压IDE压缩包后，打开文件夹。



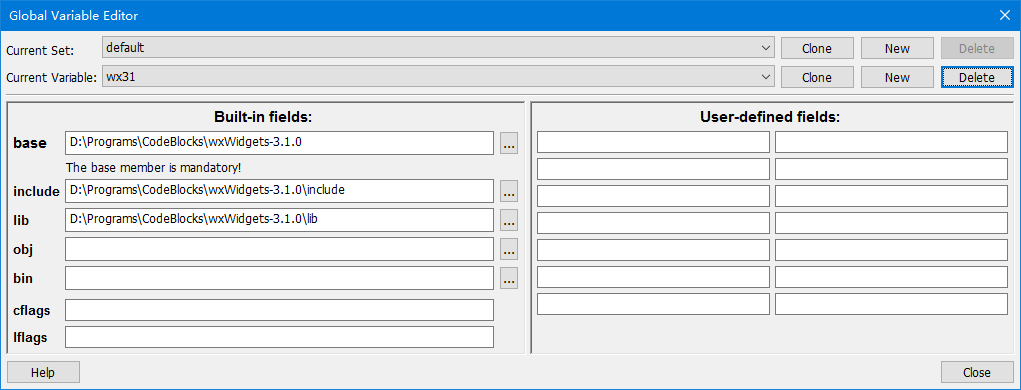
MinGW文件夹为TDM-GCC编译器，目前使用的是4.9.2版本。最新版为5.1.0，但使用最新版会导致wxWidgets库编译异常。

wxWidgets-3.1.0文件夹为wxWidgets库，已经编译完成。

点击codeblocks.exe启动IDE。



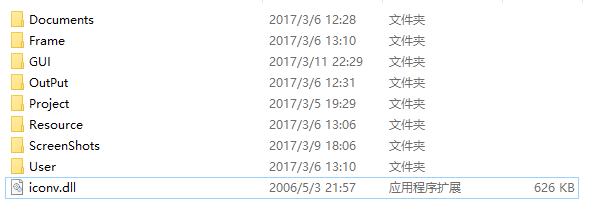
然后请点击Setting/Global variables，进入全局变量设置。



请参照上图，新建环境变量wx31。此处配置的是wxWidgets库的引用路径，不支持使用相对地址，所以请根据IDE的解压路径详细配置base、include和lib三项。

1. **文件结构**

压缩包解压后，会有如下子文件夹。



Documents：这个文件夹准备放一些说明文档。

Frame：此文件夹是模拟器的源代码。

GUI：此文件夹中就是GUI系统的API接口实现，向单片机移植时，此文件夹中的文件请按需移植。

OutPut：编译输出文件夹。

Project：工程文件夹，目前只有Codeblocks的工程文件，但是wxWidgets库也有VS版，如有需要请自行研究。

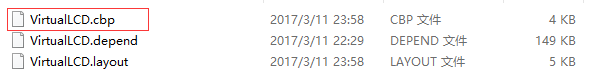
Resource：资源文件夹，包含系统图标文件等。

Screenshots：截图文件夹，保存模拟器截图用，如果没有会被自动创建。

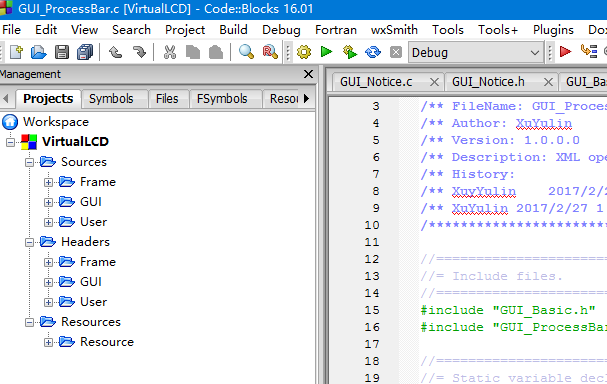
User：这里是交互接口，用于处理初始化以及用户键盘操作事件，可以用于一些简单的模拟动作。

1. **工程结构**

工程文件存放于Project\CodeBlocks下，使用CodeBlocks打开工程。



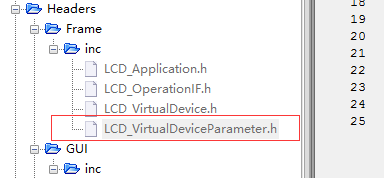
打开后，工程结构如下。

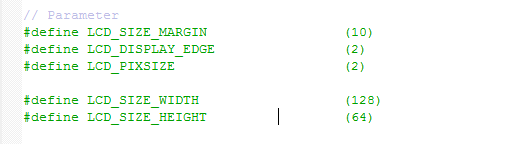


分组结构与上述所说的文件夹结构基本一致。

1. 参数配置

此模拟器何以模拟各种尺寸的单色LCD显示屏，屏幕的尺寸以及边框均可以通过调整文件LCD\_VirtualDeviceParameter.h中的参数进行配置。





LCD\_SIZE\_MARGIN参数为LCD外边缘的宽度。

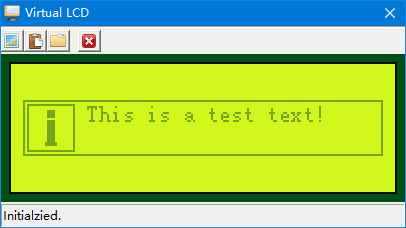
LCD\_DISPLAY\_EDGE参数为LCD显示区域边框的宽度。

LCD\_PIXSIZE参数为LCD模拟器像素点的大小。

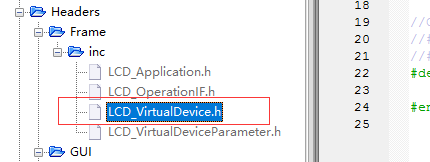
LCD\_SIZE\_WIDTH参数为显示区域尺寸的宽度。

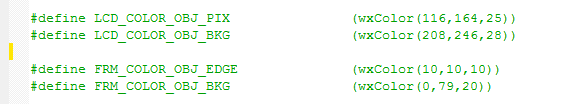
LCD\_SIZE\_HEIGHT参数为显示区域尺寸的高度。

例如，将参数LCD\_SIZE\_WIDTH设定为192，LCD\_SIZE\_HEIGHT设定为64，重新编译并运行后，我们就可以模拟19264液晶屏了。



如果想更改模拟器配色，请通过LCD\_VitualDevice.h文件中的参数进行修改。





LCD\_COLOR\_OBJ\_PIX参数为像素点有效时的颜色。

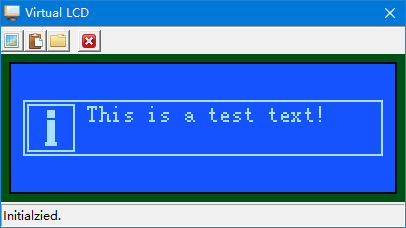
LCD\_COLOR\_OBJ\_BKG参数为像素点无效时的颜色，也就是LCD的背景色。

FRM\_COLOR\_OBJ\_EDGE参数为LCD显示区域边框的颜色。

FRM\_COLOR\_OBJ\_BKG参数为LCD外边缘的底色。

例如，模拟器默认的配色方案是黄黑屏，修改LCD\_COLOR\_OBJ\_PIX与LCD\_COLOR\_OBJ\_BKG，即可将模拟器的显示效果修改为蓝白屏。

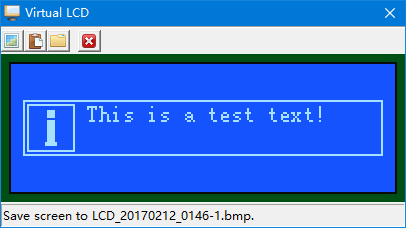


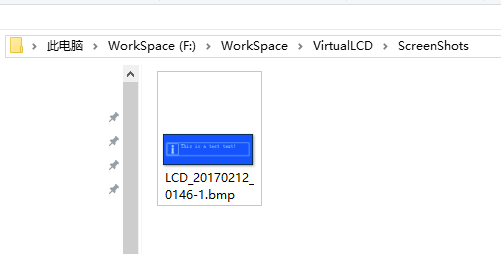


1. 附加功能

为了方便使用，模拟器提供两个附加功能。

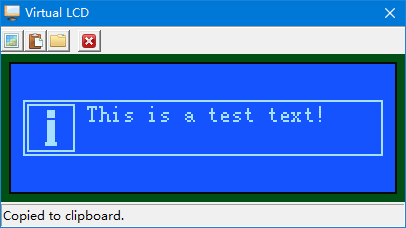
按下工具栏上第一个按钮或快捷键Ctrl+P，可以将当前画面的截图保存至Screenshots文件夹中。

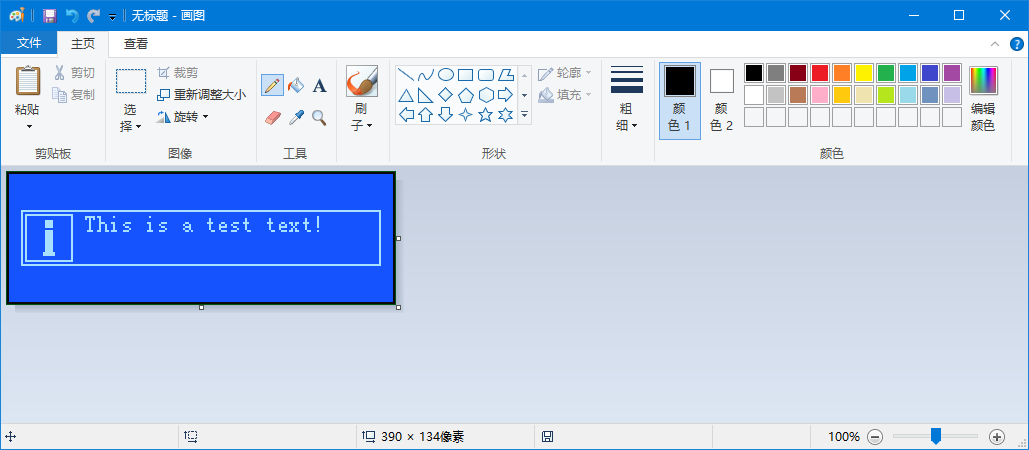






另一项功能是，按下工具栏上第二个按钮或快捷键Ctrl+C，可以将当前LCD画面复制到剪贴板，之后您可以选择粘贴在Word中，Windows画图中或作他用。



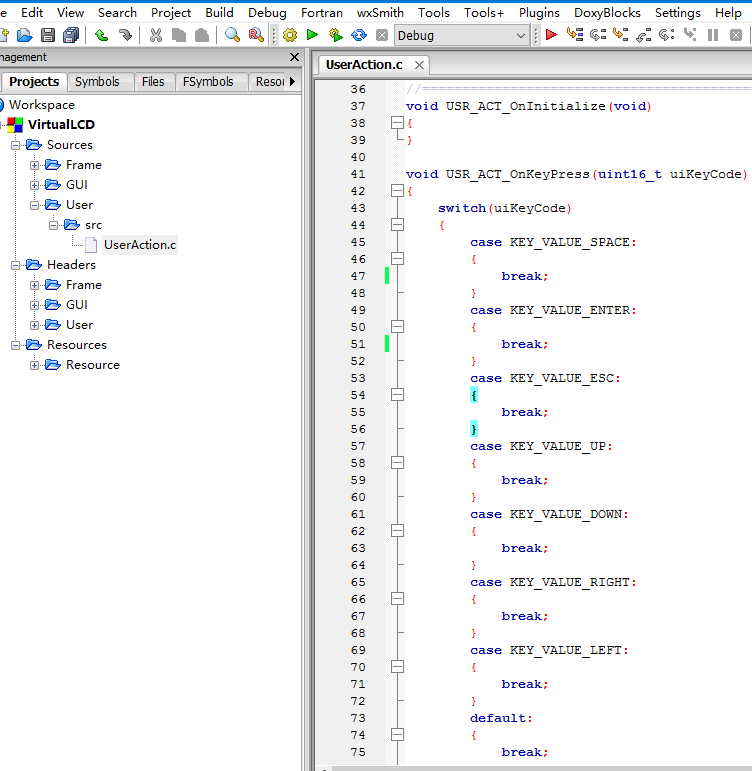


1. 使用范例

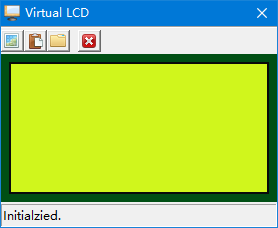
5-1、绘制图形。

本例中将简述如何在模拟器启动时，在屏幕上绘制图形。

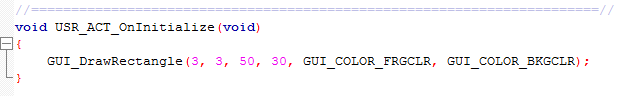
首先，打开Source/User组，双击打开User.c文件。



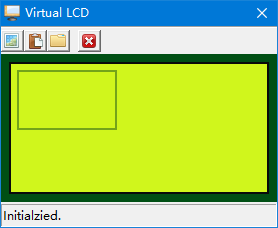
这里暂时提供了两个函数，USR\_ACT\_OnInitialize函数将在模拟器启动时调用，可以在这里添加启动时要绘制的图形，比如现在，启动模拟器，屏幕上是空白的。



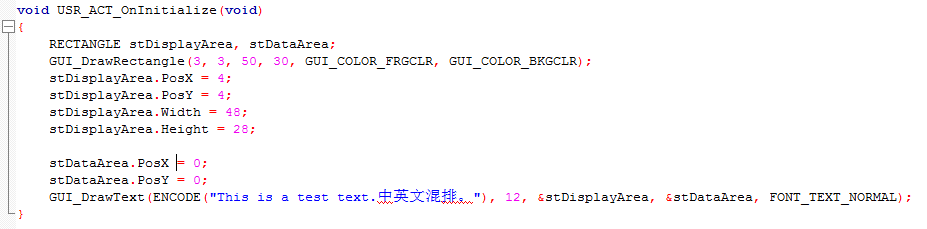
接下来我们在USR\_ACT\_OnInitialize函数中添加一些绘图指令，来使模拟器在启动时显示一些画面。



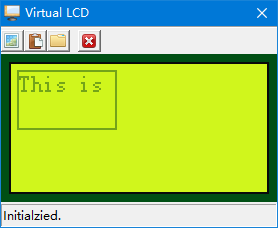
图中调用的API，会在左上角点(3,3)的位置，绘制一个长50，宽30的矩形。



接下来，尝试显示一段文字。



执行效果是这样的。

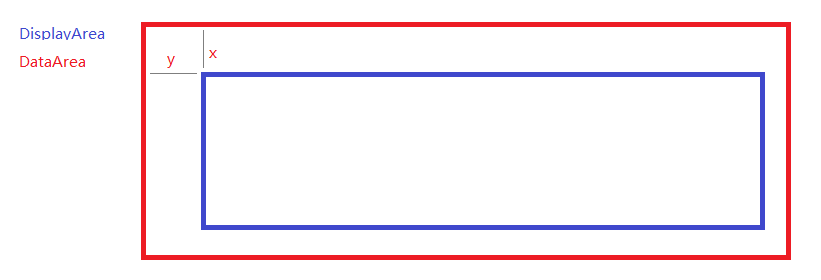


可以看到，文字被限定在一定区域内，超区域的部分不会被显示。这里要说一下这个叫RECTANGLE的类型定义，在文字、图片显示时，此类型会经常使用。

RECTANGLE类型，被定义在GUI\_Basic.h文件中，在文字显示函数中，有两个参数用到此类型，

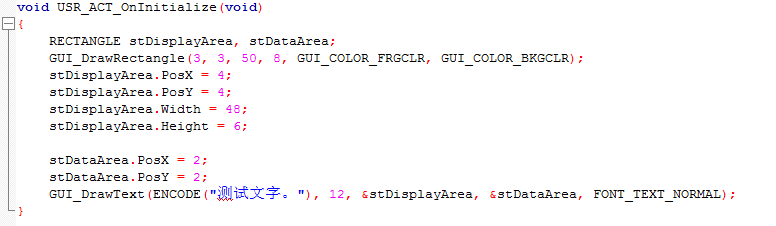


pstDisplayArea参数，限定了文字显示矩形区域的位置和大小，pstStringDataArea参数则使用到了PosX和PosY两项，用来指定文字区域外偏移，这里说的比较晦涩，看下面的图片说明应该比较好懂。

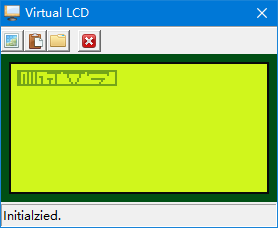


蓝色为显示区域，红色为实际的 文字或图形大小。需要注意的是，在显示文字时，DataArea项只需要指定偏移，而在调用GUI\_DrawBitMap函数进行绘图时，需要详细在DataArea参数中指定Width和Height，两项必须为图片的实际宽度与高度，否则将出现绘图不正确。

例如，我想显示一个字符串在一个小空间里，就可以这样做。



实际运行效果。



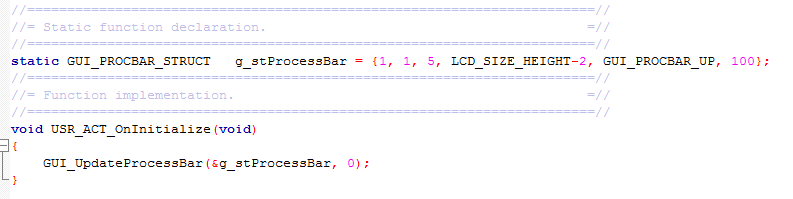
5-2、交互操作

本例中将通过ProcessBar举例说明如何建立简单的交互模拟。

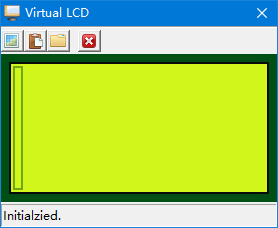
首先，添加ProcessBar的头文件引用。



然后全局声名一个用于构建ProcessBar的结构体，并在屏幕初始化函数中进行显示。



编译后运行。



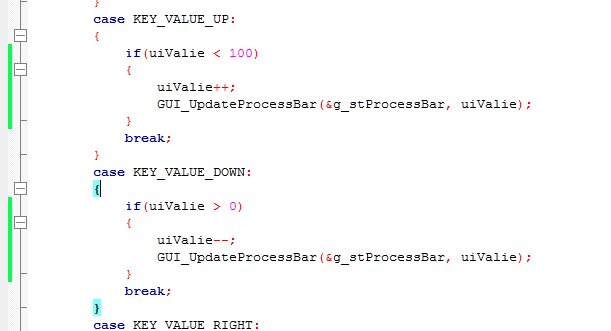
可以看到，屏幕左侧我们指定的位置，出现了一个方框。

接下来我们添加一些交互。用户的按键动作可以通过USR\_ACT\_OnKeyPress函数接收，所有按键的动作都会被投送到这里，函数中默认会有一些常用功能按键的空白分支，可以直接在这里添加代码。

我们声明一个局部静态变量，用来记录ProcessBar的值。

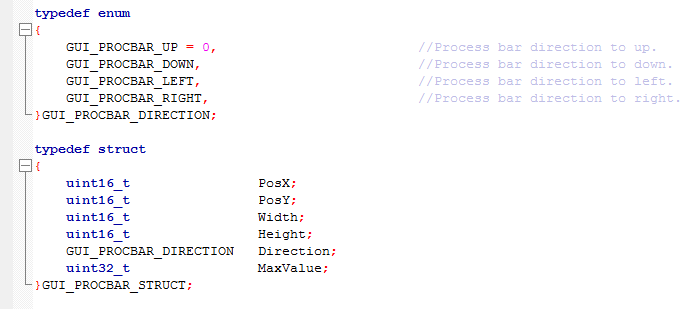


然后我们在在向上和向下按键中，添加对应的事件处理。由于在初始化的时候我们将ProcessBar的上限值设定为了100，所以此处我们最好也做出一些限定。



编译后运行，然后我们可以尝试按下键盘上的向上和向下箭头，可以看到，ProcessBar的进度已经被改变了。

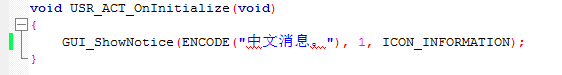
ProcessBar可以分别朝向上下左右四个方向生长，在创建结构体时，可以通过以下枚举参数决定。



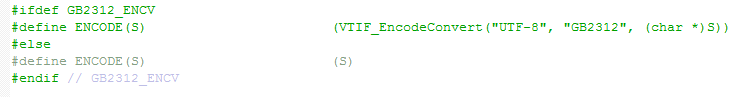
这种通过结构体数据进行显示和更新的机制将应用于所有高级控件。

5-3，编码转换

在进行文字显示的时候，示例代码中会在字符串外追加一个宏定义ENCODE。

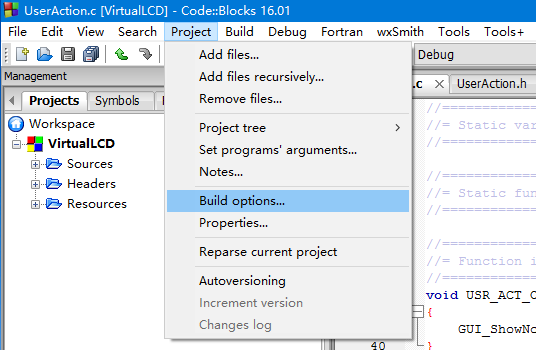


是由于为了方便防止乱码，示例代码写的时候是是以UTF8格式保存的，而程序在解析字符串时，是按照GB2312格式解析的，所以需要将字符串编码进行转换，ENCODE宏中引用了一个利用iconv库编写的转码函数。

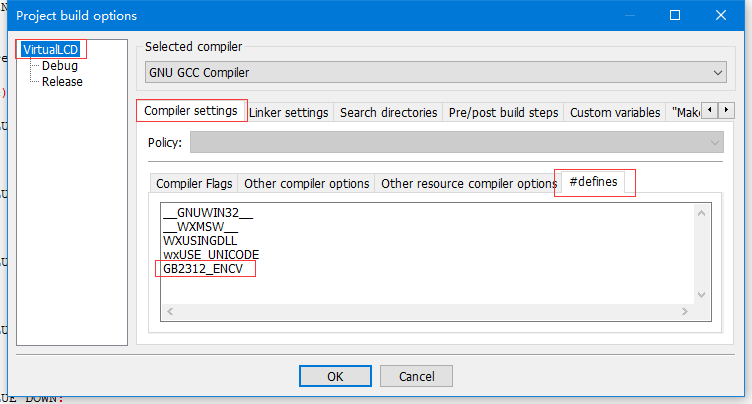




此函数用宏开关进行控制，全局宏定义请进入Project/Build options中进行设置。

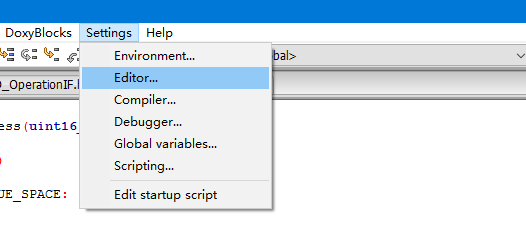


进入编译选项后，在全局宏定义中可以找到宏开关的定义。



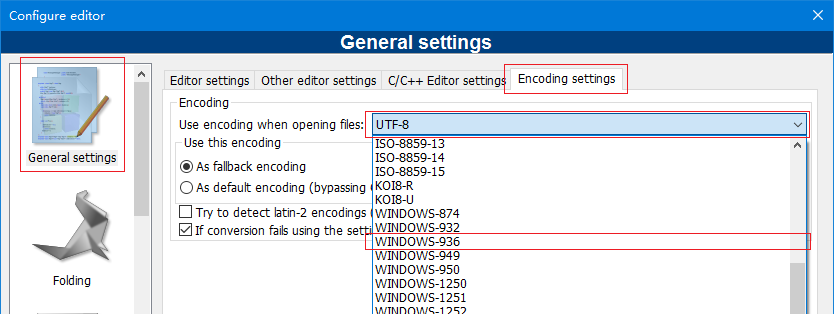
如果想直接在GB2312码下进行编译，请删除此条宏定义，并将代码编辑器改成GB2312格式。

在菜单中选择Setting/Editer，进入编辑器设置。

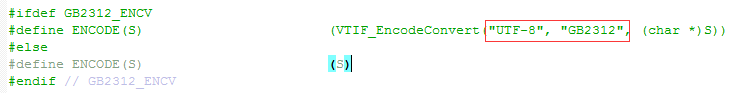


在Encoding setting选项中，可以选择编辑器的编码。

默认使用的是UTF-8格式编码，GB2312编码对应Windows-936编码。或者可以在这里选择要使用的其他编码。



修改编码后，如果使用的不是GB2312编码，需要在ENCODE宏定义中，修改编码转换的参数。



如果有任何问题和意见，可以联系QQ326684221或电子邮件xuyulin91@163.com