

TI StellarisWare 图形库中文显示使用指南

[V1.0]

[作者: Richard Ma]

[Email: mxschina@gmail.com]

1. 中文显示基本概念.....	1
2. 中文显示的实现.....	2
2.1. 设定目标效果	2
2.2. 为最终显示内容建立字库	2
2.2.1. 确定字库字符	2
2.2.2. 选择字体, 使用 ftrasterize 提取字体文件	3
2.3. 建立多语言字符串表	5
2.3.1. 设计多语言字符串表	6
2.3.2. 制作 csv 格式的多语言字符串表	6
2.3.3. 用 mkstringtable 工具生成字符表文件	8
2.4. 将字库和字符表加入程序实现中文显示	8
2.4.1. 在项目中引入字体库、字符串表	8
2.4.2. 加入图形库多语言支持的初始化代码	9
2.4.3. 选择语言环境	9
2.4.4. 使用中文显示及多语言切换	9
2.4.5. 修改示例程序实现所需效果	10
3. 示例代码.....	11

Texas Instruments 在最新的 StellarisWare 图形库中提供了中文支持。以前本人曾写过一篇介绍图形库大致使用方法的文章(TI Stellaris 图形库使用指南 – 发布在 21IC 论坛中 <http://bbs.21ic.com/viewthread.php?tid=297124>)，但该文章没有提及中文显示的相关内容。本文将对如何进行中文显示作一个简单介绍。同时，也会介绍如何让程序在多种语言间进行切换。

本文将在 TI LM3S9B96 开发板完成程序演示，以 StellarisWare 软件包中在 StellarisWare \boards\dk-lm3s9b96\hello_widget 目录下的例子为基础，实现中文显示及语言切换功能。

StellarisWare 图形库只有较新版本才支持中文，本文所使用的 StellarisWare 版本为 8555 版本，请要使用中文的朋友一定升级到 8555 以上的版本！

1. 中文显示基本概念

StellarisWare 图形库对文字的显示，即是将用户在程序中输入的文字，与对应文字的图形(字体)关联，在程序运行时，将该图形打印在屏幕上即显示了文字。

英文只有 26 个字母，算上大小写，加上数字符号空格等，不过百来个。这些元素是组成文本的全部要素。所以英文的显示并不困难，可以把全部字符的图形都存储在程序中，也不会消耗很多空间。

中文显示不同于英文。因为每个字的图形都不一样，而 MCU 程序空间有限，所以不可能把每个中文字图形都存储在程序中。最好的办法是：只保存要显示字符的图形，即建立自己定制的中文字库给程序使用。

显示中文的本质也就可以看成两步：

- 1) 提供中文字库(提供中文字图形)
- 2) 在程序中录入要显示的中文文本(提供要显示的字符串)

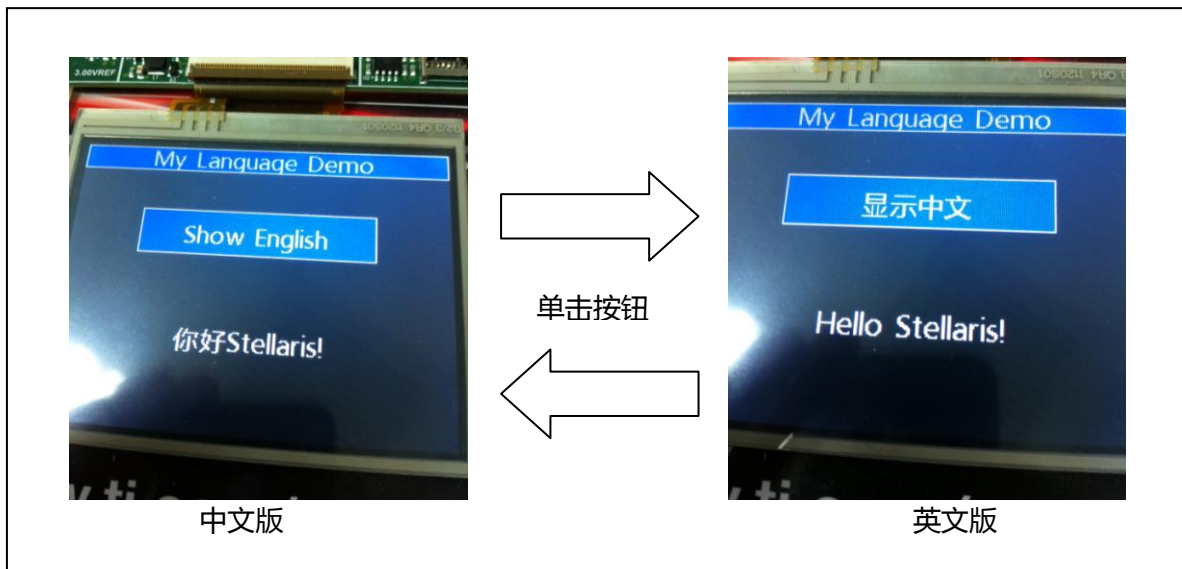
比方说要显示“中国”两个字，即首先提供一个含有“中国”二字图形的字库。然后在编写程序时候，录入的“中国”这个字符串就可以被显示了。这个过程和录入、显示“Hello World”没有本质区别。(这里如此解释是为了便于理解，实际使用时需要些额外步骤。)

StellarisWare 图形库新增的中文显示功能，可以理解为增加了相应工具，用以建立自己中文的字体库，以及可以被程序识别的中文文本。同时图形库完成了很多接口，使用户可以通过简单的 API 实现中文显示和语言切换，而不必纠结于细节。

2. 中文显示的实现

2.1. 设定目标效果

打开本文的示例打开本文的示例 StellarisWare \boards\dk-lm3s9b96\hello_widget，该例子实现了一个 Hello World 文字显示/隐藏的示例。我们在此基础上修改，实现如图 2-1 所示的效果，当点击按钮时，程序可以在中文版、英文版显示间来回切换 (英文版本中显示中文的按钮，中文版本显示英文的按钮，版本指下方文字的语言)。



2.2. 为最终显示内容建立字库

2.2.1. 确定字库字符

观察分析要显示的文字，其中包括了如下的内容：

- 中文字符：你、好、显、示、中、文
- 英文字符
- 标点符号
- 空格

根据前文所说，要显示这些文字就应该为这些字符提供图形。即中文要提供“你”、“好”、“显”、“示”、“中”、“文”这几个字符的图形。因为英文字符和标点符号等占用空间很小，所以可以把整个 ASCII 码(可现实字符也就一百多个)表都包含进字体库，以方便其它英文及符号的显示。

2.2.2. 选择字体，使用 ftrasterize 提取字体文件

字体是不同的文字图形，同一个字符可以以不同字体显示。如宋体、黑体、幼圆等都是常见的中文字体。需要注意的是，一般 Windows 系统中常见的中文字体也包含了英文字符的图形。

当选定字符后，接下来需要确定需要的字体。在图 2-1 的例子中，选用了微软雅黑字体(个人觉得比较好看)。StellarisWare 图形库中提供了一个叫 ftrasterize 的工具，可以从 Windows 系统中常见的 TTF 字体文件中提取所需文字的字体图形。

下面是提取所需文字的步骤：

1) 获得中文字体 TTF 文件

在 C:\WINDOWS\Fonts 找到喜欢的字体文件，如微软雅黑为 Microsoft YaHei (True Type)。按住 Ctrl 按钮将其拖动到桌面上。这样就将字体的 TTF 文件复制出来了 (注意记得按 Ctrl 否则系统中这个字体就被删除了)。

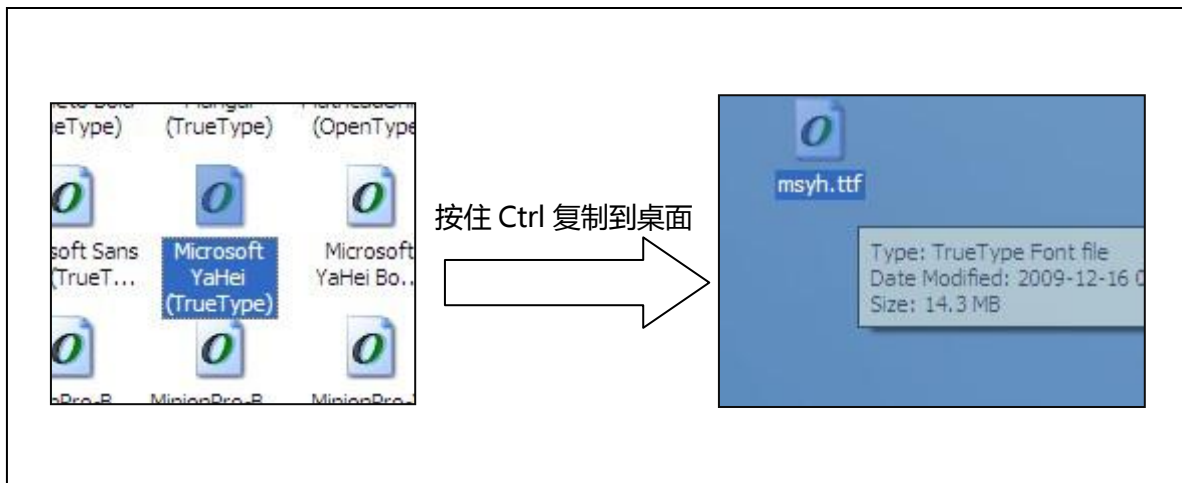


图 2-2 复制获得 TTF 字体文件

2) 找到 TI StellarisWare 软件包中字体提取软件 ftrasterize

在 StellarisWare\tools\bin 找到 ftrasterize.exe，将其和复制出来的字体 msyh.ttf 放到同一个文件夹去如 C:\myfont\ (这样方便稍后的操作)

3) 创建字符 MAP 文件

需要提取字体的字符，可以通过一个 MAP 文件进行指定。在 msyh.ttf 和 ftrasterize.exe 所在文件夹下新建一个 map.txt 文本。输入以下内容：

```
# ASCII characters  
0x00, 0xFF
```

```
# Chinese characters  
0x663E
```

0x793A
0x4E2D
0x6587
0x4F60
0x597D

该文件以行为单位。每行以“#”号开头的为注释，也可以是空，这些对指定字符无影响；每行也可以以 16 进制数字指定要提取的字符。单个数字如 0x793A 表示提取 UTF-8 码为 0x793A 的字符(即“示”字)的字体；两个数字由一个逗号与一个空格连接的，表示范围，如 0x00, 0xFF 即提取从编码 0x00 到编码 0xFF 字符的字体。

在上文中 0x00-0xFF 表示会编码所有的 ASCII 码。有了它们，所有的字母、英文标点、空格等都在里面了。

上文中下面的一串数字则是中文字符“显”、“示”、“中”、“文”、“你”、“好”的 UTF-8 码。文字的 UTF-8 码可以在网上找到转换工具。如下面的网址提供了转换：

<http://tool.chinaz.com/Tools/UTF-8.aspx>

进入网址后如图 2-3，输入“显示中文你好”这几个字符，单击“中文 转换 UTF-8”按钮，获得一串转换后的 UTF-8 码。“x”后面的 4 位字符即是每个字的 UTF-8 码。将其整理一下，按上文中例子所示，每个前面加上“0x”，整理在 map.txt 文本中。保存该文本。



图 2-3 中文字符与 UTF-8 码转换

4) 使用 ftrasterize 转换获得图形库支持的字体库文件

在开始菜单->运行中输入“cmd”进入命令控制台。输入“cd c:\myfont”回车进入 myfont 文件夹。接着输入如下命令，从 msyh.ttf 提取字符图形，建立大小为 20 像素的字体库 my_yh，如图 2-4 所示：

```
ftrasterize -f my_yh -s 20 -r -c map.txt msyh.ttf
```

命令中-c map.txt 参数即指定使用 map.txt 指定提取的字符图形。-f my_yh 指定了提出出来字库文件名字。因为 ASCII 码中有些字符是没有对应图形的，所以会出现警告(Warning)，无需理会。之后会生成 fontmy_yh20pt.c 文件，即 StellarisWare 图形库支持的含中文的字体库。打开该文件，里面的 g_pucMy_yh20pt 指针可以被强制转换为(cast)为 tFont *指针，作为图形库字体使用。

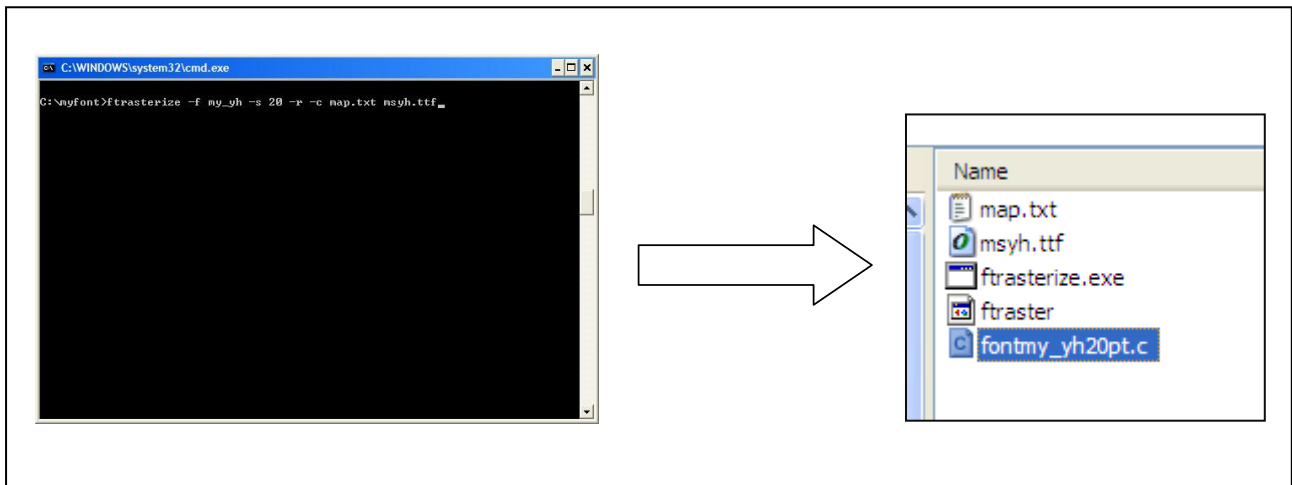


图 2-4 用 ftrasterize 工具提取字体库

2.3. 建立多语言字符串表

字库建立好后，原理上来说就可以如显示英文一样，在程序中用双引号包含的需要显示的字符串。如 `PushButtonTextSet(&g_sPushBtn, "Hello World")`。

但是如果在程序中使用 `PushButtonTextSet(&g_sPushBtn, "你好!")` 来更改按钮的文字时，即使字体的设置正确，也不能得到正确的显示结果。这是因为编码的原因。我们的 C 代码保存的文件往往以各种不同的字符格式保存(如 Unicode, UTF-8 或 ANSI 等)，所以在编译时，很难说清编译器会将某个字符以何种编码保存，所以无法正常显示。最保险的办法，是利用 StellarisWare 图新库提供的工具生成字符串表，用图形库兼容的编码格式，来储存文本。字符串表保存在单独的 C 文件及 H 头文件中，由图形库中的另一个工具 `mkstringtable` 生成的。

StellarisWare 图形库还提供了另一个重要的功能：语言切换。在字符串表里，可以保存同一文本不同语言的版本，通过在不同的语言环境下载入不同内容，图形库实现了易用的语言切换功能。

下文将对建立多语言字符串表的方法进行介绍。

2.3.1. 设计多语言字符串表

如图 2-1 所示，在两个语言版本切换时，屏幕上的文本将按表 2-1 显示。显示英文版本时，提示文字显示英文“Hello Stellaris!”；显示中文版本时，提示文字为中英文混合的“你好 Stellaris!”。在英文模式下，按钮为中文，以提示不懂英文的人；在中文模式下，按钮用英文，提示不懂中文的人。

显示的内容	英文版字符串	中文版字符串
按钮文字	显示中文	Show English
按钮下方提示文字	Hello Stellaris!	你好 Stellaris!

表 2-1 示例中中英文版本内容

2.3.2. 制作 csv 格式的多语言字符串表

由表 2-1 可以看出，一个二维的表格即可以存储多语言程序的全部字符串。StellarisWare 图形库的工具支持将 csv 格式的表格转换成兼容的字符串表代码。csv 格式是一种简单易用的表格格式，可以直接使用 Windows 的记事本(Notepad)创建和修改。建立 csv 所需步骤如下：

1) 输入表格内容

打开 Windows 记事本(Notepad)输入如下的表格内容：

```
Unused,GrLangEnUS,GrLangZhPRC
STR_SHOWLANGUAGE,显示中文,Show English
STR_HELLOSTELLARIS,Hello Stellaris!,你好 Stellaris!
```

这些内容由逗号分列，回车分行。故上文的内容相当于下表：

Unused	GrLangEnUS	GrLangZhPRC
STR_SHOWLANGUAGE	显示中文	Show English
STR_HELLOSTELLARIS	Hello Stellaris!	你好 Stellaris!

表 2-2 Notepad 中输入内容定义的表格

在表 2-2 中，第一列定义了字符串的名字，第一行定义了可以切换的语言，左上角的第一格无用。字符串名字是一个常量，由用户自己定义，用来代表某个文本，在程序中用这个名字代替字符串。可切换语言指的是可以选用的语言环境，不同语言环境下，同一文本对应的字符不同。语言环境是定义在 grlib.h 文件中的，可在下面一系列值中选择：

```
GrLangZhPRC    // Chinese (PRC) – 中国大陆简体中文
GrLangZhTW     // Chinese (Taiwan)
GrLangEnUS     // English (United States) – 美国英文
```

GrLangEnUK	// English (United Kingdom)
GrLangEnAUS	// English (Australia)
GrLangEnCA	// English (Canada)
GrLangEnNZ	// English (New Zealand)
GrLangFr	// French (Standard)
GrLangDe	// German (Standard)
GrLangHi	// Hindi
GrLangIt	// Italian (Standard)
GrLangJp	// Japanese
GrLangKo	// Korean
GrLangEsMX	// Spanish (Mexico)
GrLangEsSP	// Spanish (Spain)
GrLangSwKE	// Swahili (Kenya)
GrLangUrIN	// Urdu (India)
GrLangUrPK	// Urdu (Pakistan)

在程序中，可以调用图形库函数 `GrStringLanguageSet` 选择程序当前语言环境。当语言设置后，就可以用 `GrStringGet` 函数读出该语言环境的字符串，赋到缓冲区中作为要显示的字符串。具体用法后文的例子中会详细解释。在这里只需要明白，当语言环境为 `GrLangEnUS` 时，按表 2-2，从 `STR_SHOWLANGUAGE` 读出的字符串是“显示中文”；设置语言为 `GrLangZhPRC` 时，读出“Show English”，这样实现语言切换。

文本中有时会包含逗号，这时候需要用双引号将文字引起来。如“你好，Stellaris!”。

2) 将表格保存为 UTF-8 格式的 csv 文件

在 StellarisWare 图形库中，显示中文需要使用 UTF-8 编码。在记事本中点击另存为，弹出如图 2-5 所示的对话框。将保存类型选为 All Files，文件名为 xxxx.csv，选择 Encoding 为 UTF-8 模式。单击保存。

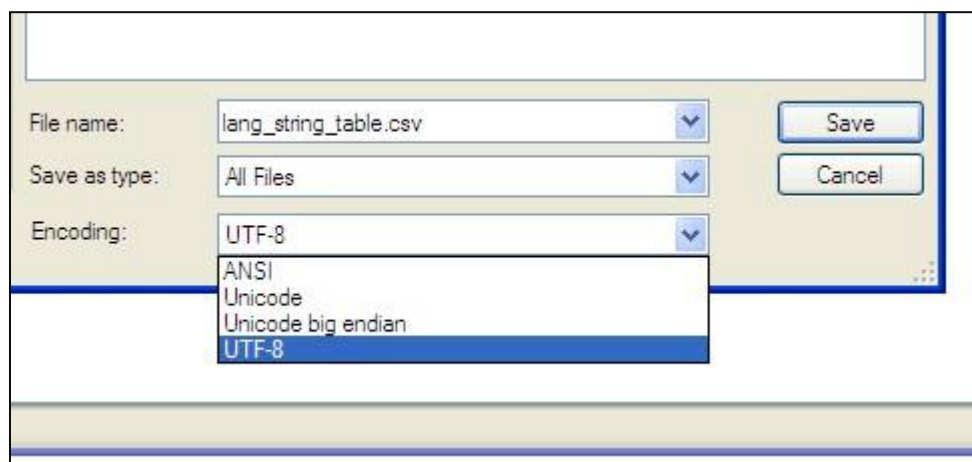
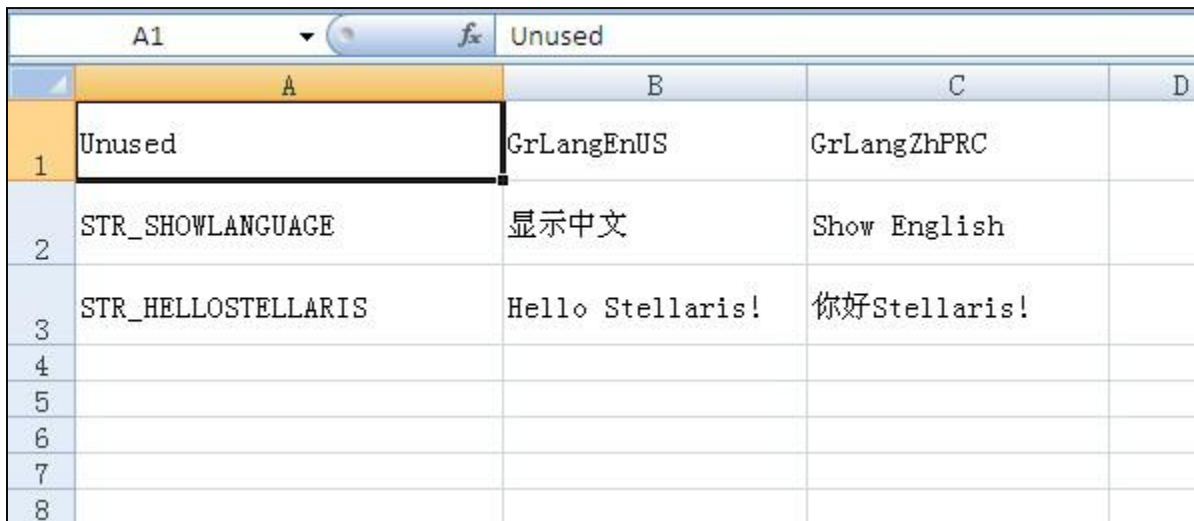


图 2-5 保存 UTF-8 编码的 csv 表格文件

文件保存后，csv 文件可以用 Excel 查看(仅仅查看，不要用 Excel 编辑，否则可能影响格式)。如果输入正确，可以看到如图 2-6 的列表。



	A	B	C	D
1	Unused	GrLangEnUS	GrLangZhPRC	
2	STR_SHOWLANGUAGE	显示中文	Show English	
3	STR_HELLOSTELLARIS	Hello Stellaris!	你好Stellaris!	
4				
5				
6				
7				
8				

图 2-6 在 Excel 中观察 csv 数据

2.3.3. 用 mkstringtable 工具生成字符表文件

在 StellarisWare\tools\bin 目录下有 mkstringtable.exe 工具，可以用之前生成的 csv 文件生成所需的 C 代码字符表。为方便起见，将生成的 csv 和 mkstringtable.exe 一同复制到之前的 C:\myfont\ 文件夹中。

在 cmd 命令行，C:\myfont\目录下，输入如下命令建立含字符表的 C 文件和 H 文件：

```
mkstringtable -u -s utf8 -f lang_string_table.csv -b my_lang
```

命令中-u 和-s utf8 指定输入的 csv 文件为 UTF-8 编码。-b my_lang 指定输出的 C 代码为 my_lang.c 和 my_lang.h。

2.4. 将字库和字符表加入程序实现中文显示

2.4.1. 在项目中引入字体库、字符串表

在完成 2.2、2.3 章后，我们得到了三个文件，将其加入项目，及项目所在文件夹：

```
fontmy_yh20pt.c
my_lang.c
my_lang.h
```

首先通过包含字符表头文件引入字符表：

```
#include "my_lang.h"
```

接下来在 `hello_widget.c` 加入如下代码，引入字体库。之后就可以在程序中用“MY_FONT”代替该字体了。

```
extern unsigned char g_pucMy_yh20pt[];
#define MY_FONT (tFont *)g_pucMy_yh20pt
```

2.4.2. 加入图形库多语言支持的初始化代码

在主函数初始化屏幕显示后(`Kitronix320x240x16_SSD2119Init`)，加入如下代码对多语言支持进行初始化：

```
GrLibInit(&g_GrLibDefaultmy_lang);
GrStringTableSet(g_pucTablemy_lang);
```

`GrLibInit` 用于初始化图形库的文字编码配置，如果要使用 UTF-8 编码的中文，这个函数必须被调用，否则将使用默认的文字编码 ISO8859-1 (不支持中文)。其参数类型为 `const tGrLibDefaults *pInit`，配置值 `g_GrLibDefaultmy_lang` 在建立字符表串表时已自动生成，在 `my_lang.h` 里有声明。

设置使用非默认文字编码后，要指定系统使用的字符串表。使用 `GrStringTableSet` 函数指定字符串表为之前生成的字符串表 `g_pucTablemy_lang` (在 `my_lang.h` 中有声明，`const unsigned char` 型)。

2.4.3. 选择语言环境

由于我们希望系统启动时默认使用中文，接下来设置语言环境为中文：

```
GrStringLanguageSet(GrLangZhPRC);
```

`GrStringLanguageSet` 函数中的参数可以是前文列举过的语言。不过由于例子中只提供了中英文支持，本例中参数只能为 `GrLangZhPRC` 或 `GrLangEnUS`。

2.4.4. 使用中文显示及多语言切换

要显示某个内容需要在内存中开辟一块空间存储字符串，为按钮和显示的文字声明下面数组：

```
char g_pcTittle[20];
char g_pcShow[20];
```

有了缓冲区，就可以用 `GrStringGet` 将字符串表中的字符读入缓冲区，示例代码如下：

```
GrStringGet(STR_SHOWLANGUAGE, g_pcTittle, sizeof(g_pcTittle));
```

看到这里，许多反应快的朋友立刻就明白了，只需要将图形库控件的文字指向缓冲区，字体设为字体库字体，那么就可以显示中文了。下面的示例为在按钮上使用中文字体的示例。高亮标注出的内容为 `hello_widget` 例子中更改的部分。

```
RectangularButton(
    g_sPushBtn, &g_sHeading, 0, 0, &g_sKitronix320x240x16_SSD2119, 60, 60, 200, 40,
    (PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT | PB_STYLE_FILL |
     PB_STYLE_RELEASE_NOTIFY),
```

```
ClrDarkBlue, ClrBlue, ClrWhite, ClrWhite,
MY_FONT, g_pcTittle,
0, 0, 0, 0, OnButtonPress);
```

所以切换语言的方法也就比较容易解释了。即让控件的文字始终指向缓冲区，在切换语言 (GrStringLanguageSet) 后，使用 GrStringGet 将对应语言的文本载入缓冲区，再调用 WidgetPaint 重绘控件，就可以看到文字的改变。

2.4.5. 修改示例程序实现所需效果

本节只阐述在之前基础上仍须完成的工作，具体代码请参考第 3 章。

- 1) 初始化时候就应该使用 GrStringGet 载入对应语言的文本:

```
GrStringGet(STR_SHOWLANGUAGE, g_pcTittle, sizeof(g_pcTittle));
GrStringGet(STR_HELLOSTELLARIS, g_pcShow, sizeof(g_pcShow));
```

- 2) 代码不再隐藏屏幕下方的文字，所以需要在初始化时直接将下方文字控件加入控件树:

```
WidgetAdd((tWidget *) &g_sPushBtn, (tWidget *) &g_sHello);
```

- 3) 修改单击按钮处理函数 OnButtonPress，实现在点击按钮时的文字切换:

```
void OnButtonPress(tWidget *pWidget)
{
    g_bHelloVisible = !g_bHelloVisible;

    if (g_bHelloVisible)
        GrStringLanguageSet(GrLangEnUS); // 切换为英文
    else
        GrStringLanguageSet(GrLangZhPRC); // 切换为中文

    // 载入文字
    GrStringGet(STR_SHOWLANGUAGE, g_pcTittle, sizeof(g_pcTittle));
    GrStringGet(STR_HELLOSTELLARIS, g_pcShow, sizeof(g_pcShow));

    // 重绘控件
    WidgetPaint((tWidget *)&g_sPushBtn);
}
```

3. 示例代码

以下为 hello_widget.c 中代码。高亮部分为在原有例子上修改后代码。

```
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "grlib/grlib.h"
#include "grlib/widget.h"
#include "grlib/canvas.h"
#include "grlib/pushbutton.h"
#include "drivers/kitronix320x240x16_ssd2119_8bit.h"
#include "drivers/touch.h"
#include "drivers/set_pinout.h"

// 引入自定字符串列表
#include "my_lang.h"

// 引入自定字体库
extern unsigned char g_pucMy_yh20pt[];
#define MY_FONT (tFont *)g_pucMy_yh20pt

// 定义缓冲区
char g_pcTittle[20];
char g_pcShow[20];

//*****
//
// Forward reference to various widget structures.
//
//*****
extern tCanvasWidget g_sBackground;
extern tPushButtonWidget g_sPushBtn;

//*****
//
// Forward reference to the button press handler.
//
//*****
void OnButtonPress(tWidget *pWidget);

//*****
//
// The heading containing the application title.
//
```

```

//*****
Canvas(g_sHeading, &g_sBackground, 0, &g_sPushBtn,
      &g_sKitronix320x240x16_SSD2119, 0, 0, 320, 23,
      (CANVAS_STYLE_FILL | CANVAS_STYLE_OUTLINE |
      CANVAS_STYLE_TEXT),
      ClrDarkBlue, ClrWhite, ClrWhite, MY_FONT,
      "My Language Demo", 0, 0);

//*****
//
// The canvas widget acting as the background to the display.
//
//*****
Canvas(g_sBackground, WIDGET_ROOT, 0, &g_sHeading,
      &g_sKitronix320x240x16_SSD2119, 0, 23, 320, (240 - 23),
      CANVAS_STYLE_FILL, ClrBlack, 0, 0, 0, 0, 0, 0);

//*****
//
// The button used to hide or display the "Hello World" message.
//
//*****
RectangularButton(g_sPushBtn, &g_sHeading, 0, 0,
      &g_sKitronix320x240x16_SSD2119,
      60, 60, 200, 40,
      (PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT |
      PB_STYLE_FILL | PB_STYLE_RELEASE_NOTIFY),
      ClrDarkBlue, ClrBlue, ClrWhite, ClrWhite,
      MY_FONT, g_pcTittle, 0, 0, 0, 0, OnButtonPress);

//*****
//
// The canvas widget used to display the "Hello!" string. Note that
// this is NOT hooked into the active widget tree (by making it a
// child of the g_sPushBtn widget above) yet since we do not want the
// widget to be displayed until the button is pressed.
//
//*****
Canvas(g_sHello, &g_sPushBtn, 0, 0,
      &g_sKitronix320x240x16_SSD2119, 0, 150, 320, 40,
      (CANVAS_STYLE_FILL | CANVAS_STYLE_TEXT),
      ClrBlack, 0, ClrWhite,
      MY_FONT, g_pcShow, 0, 0);

//*****
//
// A global we use to keep track of whether or not the "Hello" widget

```

```

// is currently visible.
//
//*****
tBoolean g_bHelloVisible = false;

// 按键处理函数
void OnButtonPress(tWidget *pWidget)
{
    g_bHelloVisible = !g_bHelloVisible;

    // 切换语言
    if (g_bHelloVisible)
        GrStringLanguageSet(GrLangEnUS);
    else
        GrStringLanguageSet(GrLangZhPRC);

    // 载入对应语言下的字符串
    GrStringGet(STR_SHOWLANGUAGE, g_pcTittle, sizeof(g_pcTittle));
    GrStringGet(STR_HELLOSTELLARIS, g_pcShow, sizeof(g_pcShow));

    // 重绘控件
    WidgetPaint((tWidget *) &g_sPushBtn);
}

int main(void)
{
    //
    // Set the system clock to run at 50MHz from the PLL
    //
    ROM_SysCtlClockSet( SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL |
                        SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    //
    // Set the device pinout appropriately for this board.
    //
    PinoutSet();

    //
    // Enable Interrupts
    //
    ROM_IntMasterEnable();

    //
    // Initialize the display driver.
    //
    Kitronix320x240x16_SSD2119Init();
}

```

```
//
// 初始化字符编码和字符串表.
//
GrLibInit(&g_GrLibDefaultmy_lang);
GrStringTableSet(g_pucTablemy_lang);

//
// 设置默认语言并载入字符串
//
GrStringLanguageSet(GrLangZhPRC);
GrStringGet(STR_SHOWLANGUAGE, g_pcTittle, sizeof(g_pcTittle));
GrStringGet(STR_HELLOSTELLARIS, g_pcShow, sizeof(g_pcShow));

//
// Initialize the touch screen driver.
//
TouchScreenInit();

//
// Set the touch screen event handler.
//
TouchScreenCallbackSet(WidgetPointerMessage);

//
// Add the compile-time defined widgets to the widget tree.
//
WidgetAdd(WIDGET_ROOT, (tWidget *) &g_sBackground);
WidgetAdd((tWidget *) &g_sPushBtn, (tWidget *) &g_sHello);

//
// Paint the widget tree to make sure they all appear on the
// display.
//
WidgetPaint(WIDGET_ROOT);

//
// Loop forever, processing widget messages.
//
while(1)
{
    //
    // Process any messages from or for the widgets.
    //
    WidgetMessageQueueProcess();
}
}
```