

目录

前言.....	2
一. GuiTools 简介.....	2
二. 信息反馈.....	2
第一章 点阵字库.....	3
一. 生成字库.....	3
二. 字库预览与编辑	7
A. 如何预览一个字符的点阵信息	7
B. 如何编辑字符点阵	7
三. 如何 show 一个字符	8
1. MbcS.....	8
2. Unicode.....	8
四. 字库文件.....	8
➤ 文件结构.....	8
➤ 图文详解.....	10
五. 字符集	14
➤ 支持所有 windows 字符集.....	14
➤ 字符集顺序	14
第二章 多国语言.....	15
一. 多国语言文本管理.....	15
二. 内码 转 Unicode	17
附注.....	18

前言

一. GuiTools 简介

GuiTools 是一款集点阵字库生成、多国语言管理和图片转换等功能的软件，主要应用在所有需要点阵文字显示及图片资源显示的嵌入式系统中。

点阵字库生成 可快速转换任意字体、任意点阵大小及多种编码（MbcS / Unicode / Simple Unicode）选择，且支持多种输出文件格式（Bin / Bdf / Txt / Bmp）。

多国语言管理 支持读取 Excel 表格形式的文本资源，转换时可指定编码（MbcS / U16-LE / Utf8）格式，及输出文件格式（*.res / *.h）。

图片转换 支持多种图片文件格式（bmp / jpg / png 等），可转换成 RGB 和 YUV 等数据格式。

二. 信息反馈

为了能够更快更好的完善该工具，以便于大家在使用中达到事半功倍的效果。欢迎您的意见和反馈信息，我们都将慎重考虑、酌情处理，及时给以回复。谢谢！

联系人： 建国雄心

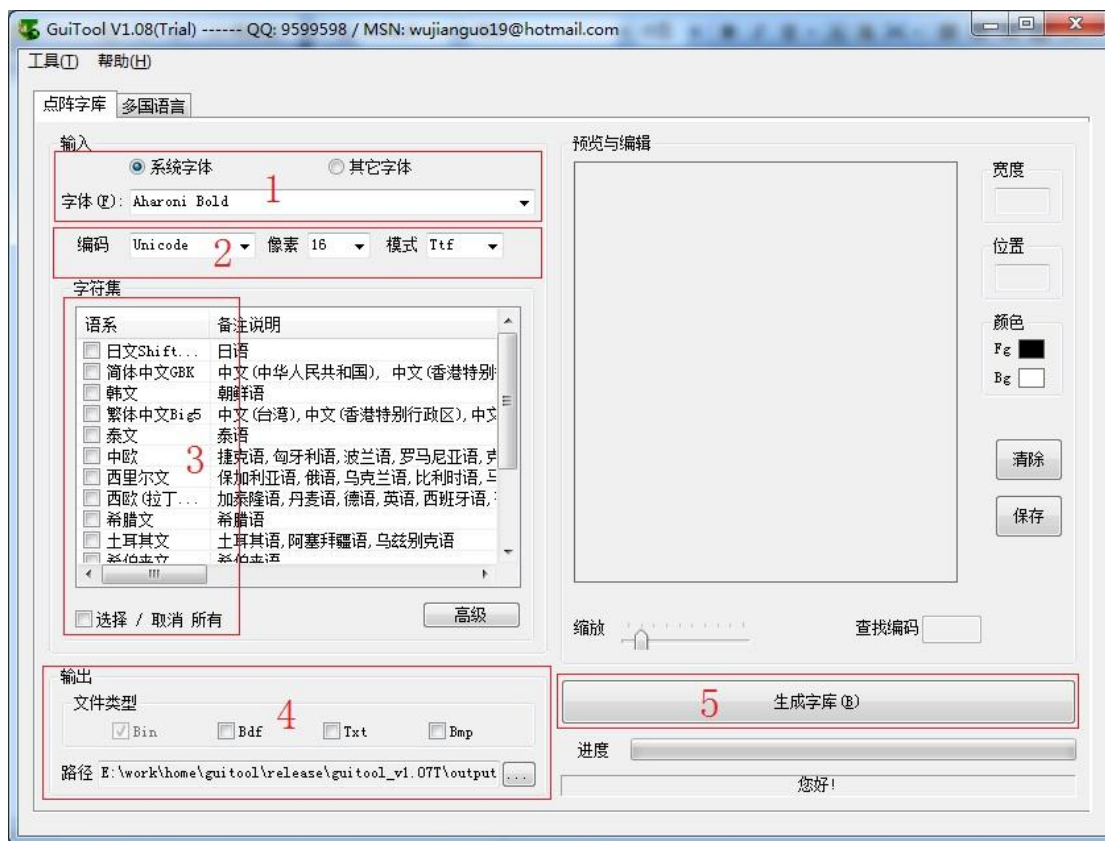
QQ: 9599598

MSN: wujianguo19@hotmail.com

非常感谢您的阅读!!!

第一章 点阵字库

一. 生成字库

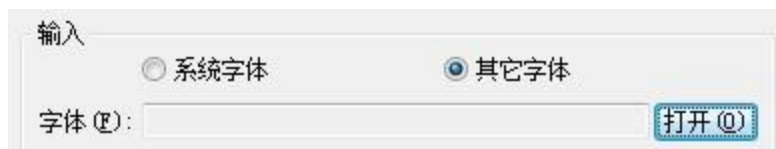


如上图红色框选部分所示，分 5 步。

1. 选择字体

在选择字体之前，请先选择字体类型。

- A. **系统字体**，即系统已安装字体 (*.ttc / *.ttf)。选择系统字体，会显示一个下拉框，所有系统字体都列举其中，选择你的目标字体即可。
- B. **其它字体**，即未安装字体 (*.ttc / *.ttf)，或其它格式字体 (*.bdf / *.bin)。选择其它字体，会显示一个静态编辑框和一个按钮(如下图所示)，点击“打开”按钮，会弹出一个对话框，选择你的目标字体文件即可。



补充说明： 建议去网上找一个 ArialUni.ttf 字库（也可找本人提供，字符相对较全面）。

下载地址：<http://ishare.iask.sina.com.cn/f/4942778.html>

2. 选择编码格式，字体大小，转换模式

➤ 编码格式

目前支持三种： MbcS、Unicode 和 Simple Unicode

1) MBCS， 它的点阵信息按内码的编码顺序存放，一个字符集生成一个字库文件，使用于较传统的字库方式。例如： ucdos 字库（HZK16， HZK24F）

2) Unicode 和 Simple Unicode ， 它们的点阵信息都是按 Unicode 的编码顺序存放，只是文件格式存在小小差异，多个字符集可集合生成一个字库文件，使用相对较普遍。

具体选择哪种编码格式视情况而定，如系统送给显示的编码是内码，则使用 MbcS 编码格式，否则使用 Unicode 编码格式。

例如： 字符 ‘€’ ， 它（如下图示）的内码编码是 0x80， Unicode 是 0x20AC， 如果传递给显示接口的编码是0x80， 表示它使用的是内码格式， 则在生成字库时选择 MBCS， 否则选择 Unicode 。



➤ 字体大小

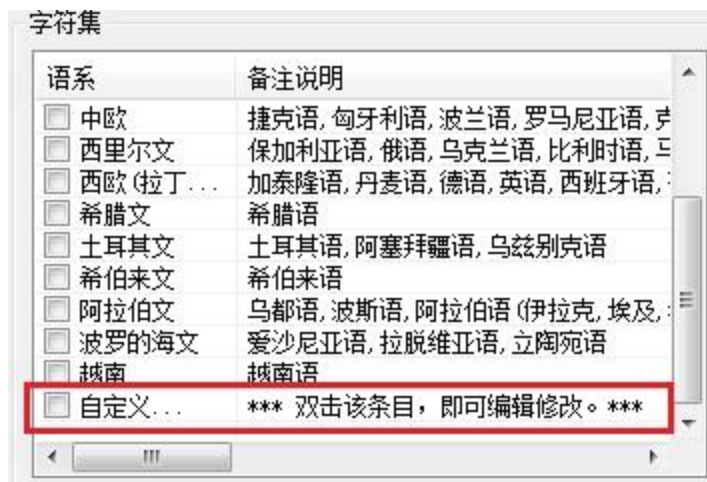
范围（ >= 8 && <= 63 ），支持手动编辑该参数。

➤ 转换模式（略）

Otf 生成的中文字符效果比 Ttf 更匀称，但英文效果却没有 Ttf 效果好。

3. 选择需要支持的字符集

- 1) 支持 windows 所有字符集。
- 2) 增加了一个自定义功能（如下图示），支持自定义编码段（仅 Unicode 编码有效），实现扩展字符的增加（**非常实用**）。



4. 设置输出文件类型，及输出路径

默认输出至当前工作目录中的output目录

5. 点击生成字库（ Build ）按钮，转换生成字库

◆ 补充说明：

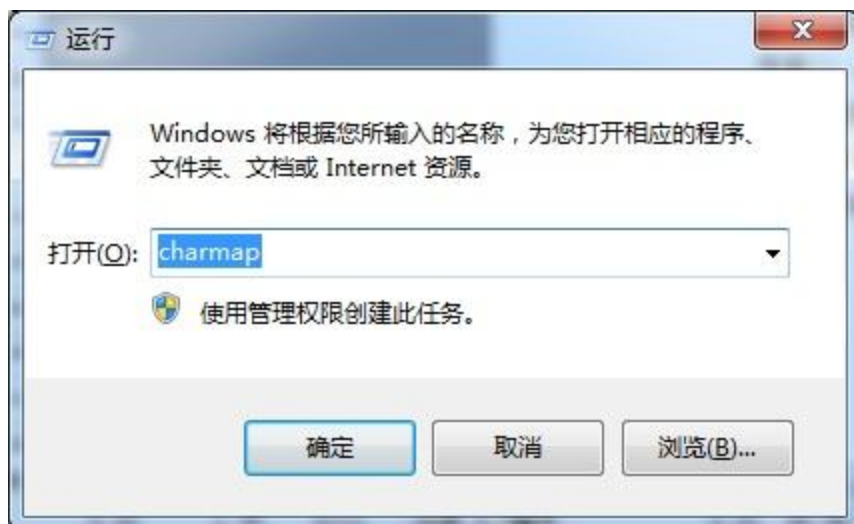
- A. 若选择的是 MBCS 编码格式，则会根据选择的字符集数产生相应份数的上述文件。
- B. 若选择的是 Unicode 编码格式，不论选多少个字符集，都只会输出一份上述文件。

C. 若生成字库失败，则有可能选择的字体文件(*.ttf)原本就不包含该字符集的字符信息。如：宋体中不存在韩文字符，即用宋体生成的字库无法支持韩文显示。

参考办法：用系统自带的字符映射表进行参照，其字符映射表中可选择不同的字体、字符集（点击“高级查看”）。

开启 字符映射表 的方法：

1) 以命令方式运行开启，点击开始菜单-->选择运行-->键入"**charmap**"回车即可。 如下图示：



2) 以菜单方式开启，点击开始菜单-->所有程序-->附件-->系统工具-->字符映射表

二. 字库预览与编辑

A. 如何预览一个字符的点阵信息



如上图示红色框选部分，分 3 步。

1. 选择“其它字体”。
2. 打开一个点阵字库文件 (*.bin)
将会自动打开并分析出其编码类型、点阵大小，包含的字符集等信息。
3. 根据编码格式 (MbcS, Unicode) 输入相应编码 (十六进制)。
例如：
 - a. 字库编码格式为 MbcS，则输入内码编码。以“建”字为例，在 3 处输入“BDA8”回车确认即可。
 - b. 字库编码格式为 Unicode，则输入 Unicode 码。同以“建”字为例，在 3 处输入“5EFA”回车确认即可。

补充说明：1. 如某字符不能预览，则表示该字库中不包含此字符。

2. 上图中黄色框选部分表示当前打开字库的相关信息。

B. 如何编辑字符点阵

如上图示蓝色框选部分，分 3 步。目前仅支持第一种扫描模式（横向 b7~b0）。

1. 编辑修改显示像素
将光标移到字符预览区域，单击鼠标左键描点，右键清点。
2. 清除与保存当前字符点阵信息
清除（Clean）与保存（Save）按钮，分别表示清除和更新保存当前字符的点阵信息。

3. 修改预览颜色

Fg : 前景色 (bit 为 1), Bg : 背景色 (bit 为 0)

三. 如何 show 一个字符

具体见 [Example](#) 子目录.

1. MbcS

A. 等宽 (CJK)

- a. 先读出 FL_Header 信息;
- b. 计算出 code 在当前字符集中的索引值 (index), 然后根据这个 $\text{sizeof}(\text{FL_Header}) + \text{index} * (\text{FL_Header.Ysize}/8 * \text{FL_Header.Ysize})$ 找到 code 的点阵数据;
- c. 然后根据 FL_Header.Ysize 与点阵数据即可 show 当前字符。

具体见: [MBCS 编码中的简中、繁中、日文和韩文索引值计算方法](#)

B. 非等宽 (拉丁文)

- a. 先读出 FL_Header 信息;
- b. 根据这个 $\text{sizeof}(\text{FL_Header}) + \text{code} * 2$ 找到 code 的 UFL_CHAR_INDEX 信息;
- c. 根据 UFL_CHAR_INDEX 的 OffAddr 再找到当前 code 的点阵数据;
- d. 最后根据 FL_Header.Ysize、UFL_CHAR_INDEX.Width 以及点阵数据即可 show 出当前字符。

例程: [.\Example\font\MbcS](#)

2. Unicode

- A. 先读出 FL_Header 信息;
- B. 分析当前字符属第几段。比如在第 n 段, 就可根据这个 $\text{xxx}[n].\text{OffAddr} + (\text{code} - \text{xxx}[n].\text{First}) * \text{sizeof}(\text{UFL_CHAR_INDEX})$ 找到字符检索信息 (UFL_CHAR_INDEX);
- C. 根据 UFL_CHAR_INDEX 的 OffAddr 再找到当前字符的点阵数据;
- D. 最后根据 FL_Header.Ysize、UFL_CHAR_INDEX.Width 以及点阵数据即可 show 出当前字符。

例程: [.\Example\font\Unicode](#)

四. 字库文件

➤ 文件结构

字库文件指的是通过 tool 转换出来的 bin 文件(低字节序)。通常由以下几部分组成。

1. 文件头

指的是文件的前 16 个字节 (BYTE)，描述信息如下结构：

```
typedef struct tagFontLibHeader{
    BYTE    magic[4]; //U('S', 'M'), 'F', 'L', X---Unicode(Simple or MBCS), X: Version
    DWORD   dwFileSize;          /* File total size */
    BYTE    nSection; /* total sections */
    BYTE    YSize; /* height of font */
    WORD    wCpFlag; /* codepageflag 每个 bit 位表示一个字符集。即最多可表示 16 个字符集。
*/
    WORD    nTotalChars; /* 总的有效字符数 */
    char     reserved[2];
} FL_Header, *PFL_Header;
```

2. 段信息

只针对 Unicode 编码有效，占字节数：nSection*sizeof(FL_SECTION_INF)。结构如下：

```
struct tagFlSectionInfo{
    WORD    First;          /* first character */
    WORD    Last;           /* last character */
    DWORD   OffAddr; /* 指向的是当前 SECTION 包含的 UFL_CHAR_INFO 第一个字符
信息的起始地址 */
} FL_SECTION_INF, *PFL_SECTION_INF;
```

3. 检索表

只针对 Unicode 字库 和 非等宽的 MbcS 有效。不包含 MbcS 的 CJK (简中、繁中、日文、韩文)，因这些都将是等宽处理。

```
typedef struct tagUflCharInfo{
    DWORD   OffAddr : 26; /* 当前字符点阵数据的起始地址
    DWORD   Width : 6; /* 字符点阵的像素的宽度( 目前最大支持点阵 < 64)
} UFL_CHAR_INDEX;
```

4. 点阵数据

即当前字库中所有包含字符的点阵数据集合。数据存储方式为：横向高到底位存储。如： 10110011 00011010 即为 B3.1A

补充说明：

(Y: 包含 / N: 不包含)

	文件头	段信息	检索表	点阵数据
MbcS 等宽 (CJK)	Y	N	N	Y
MbcS 非等宽 (拉丁文)	Y	N	Y	Y
Unicode	Y	Y	Y	Y

➤ 图文详解

A. MBCS-等宽 (CJK)

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	4D	46	4C	11	50	FE	03	00	00	10	02	00	F2	1F	00	00	; MFL.P?.....?..
00000010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000040h:	00	00	30	00	18	00	0C	00	04	00	00	00	00	00	00	00	; ..0.....
00000050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
...																	

解析如下:

1). 文件头

蓝底标记部分, 即前 16 Byte, 如下图示。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	4D	46	4C	11	50	FE	03	00	00	10	02	00	F2	1F	00	00	; MFL.P?.....?..

- 4D 46 4C 11 -- 标识头, 判断是否为合法的字库文件。
 4D = 'M', 表示该文件为 MBCS 编码格式的字库文件。
 46 = 'F', 4C = 'L'
 11 表示该字库文件版本信息为: Version 1.1
- 50 FE 03 00 -- 文件总长度, 即文件大小为: 0x3fe50
- 00 -- 是否包含检索表。0-标识无检索表。
- 10 -- 字体高度 (宽高都以像素为单位), 即表示 16 点阵。
- 02 00 -- 选择的字符集标志位。即 0x0002 (00000000 00000010), 根据字符集序, 故得出当前选择为: 简中字符集。
- F2 1F -- 有效点阵字符数。即表示有 0x1ff2 个字符的点阵数据。
- 00 00 -- 预留字节

2). 点阵数据

除去文件头 16 Byte 外, 其它数据都是纯字符点阵数据。

因为 GB2312 (简体中文) 的首个字符为: 0xA1A1, 它的点阵数据起始地址为 0x10 (除去文件头), 数据长度为: $((\text{字体高度}+7)/8) * \text{字体高度} = ((16+7)/8)*16=32$ 。

故从 0x10 开始连续取 32 字节, 即为字符 0xA1A1 的点阵数据。如下图示:

```
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

同理如下即为字符 0xA1A2 的点阵数据。

```
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040h: 00 00 30 00 18 00 0C 00 04 00 00 00 00 00 00 00 00 00
```

由于等宽, 故所有字符的点阵数据长度都为: $((\text{字体高度}+7)/8) * \text{字体高度} = ?$

B. MBCS-非等宽（拉丁文）

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	4D	46	4C	11	70	1A	00	00	01	10	80	00	DA	00	00	00	; MFL.p....€.?...
00000010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000090h:	10	04	00	10	20	04	00	10	30	04	00	18	40	04	00	24	;0...@...\$
000000a0h:	60	04	00	24	80	04	00	38	A0	04	00	2C	C0	04	00	0C	; `..\$€..8?.,?..
000000b0h:	D0	04	00	14	E0	04	00	14	F0	04	00	18	00	05	00	24	; ?..?..?.....\$
...																	
00000410h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	;
00000420h:	00	40	40	40	40	40	40	40	00	40	40	00	00	00	00	00	; .00000000.00....
00000430h:	00	48	48	48	48	00	00	00	00	00	00	00	00	00	00	00	; .HHHH.....
00000440h:	00	00	09	00	09	00	12	00	FF	80	12	00	34	00	24	00	; €..4.\$.

解析如下：

1). 文件头

蓝底标记部分，即前 16 Byte，如下图所示。

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	4D	46	4C	11	70	1A	00	00	01	10	80	00	DA	00	00	00	; MFL.p....€.?...

4D 46 4C 11 -- 标识头，判断是否为合法的字库文件。
 4D = 'M'， 表示该文件为 MBCS 编码格式的字库文件。
 46 = 'F'， 4C = 'L'
 11 表示该字库文件版本信息为： Version 1.1
 70 1A 00 00 -- 文件总长度，即表示文件总长度为： 0x1A70
 01 -- 是否包含检索表。 1-标识有检索表
 10 -- 字体高度，表示字体大小为 16 点阵。
 80 00 -- 选择的字符集标志位。即 0x0080 (00000000 10000000)，根据[字符集序](#)，故得出当前选择为： 西欧字符集。
 DA 00 -- 有效点阵字符数。 即表示有 0xDA 个字符的点阵数据。
 00 00 -- 预留字节

2). 检索表

从 00000010h 开始，每 4 个字节表示一个字符的检索信息，且从字符 0x0 开始。故空格字符(' '，编码为 0x20)的检索信息（文件头的长度+字符编码*4 = 0x10 + 0x20 *4 = 0x90， 即 000000090h）为：10 04 00 10，即得出一个 32 位数为： 0x10000410（十六进制） --- (00010000 00000000 00000100 00010000)。

高 6 位，表示当前字符的宽度，故得出 000100 -- 4（字库宽度为 4）。

低 26 位，表示当期字符的点阵数据的偏移地址，故得出 00 00000000 00000100 00010000 -- 0x410（点阵信息的起始地址为 0x410）。

3). 点阵数据

由于空格字符的起始地址为 0x410, 且数据长度为: $((\text{字体宽度}+7)/8) * \text{字体高度} = ((4+7)/8)*16 = 16$ 。

故取如下 16 字节, 即为空格字符的点阵数据。

```
00000410h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

同理如下即为字符'!' 的点阵信息。

```
00000420h: 00 40 40 40 40 40 40 40 00 40 40 00 00 00 00 00
```

Unicode (重点)

```
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 55 46 4C 11 3C 52 00 00 03 18 A0 00 0F 01 00 00 ; UFL.<R....?....
00000010h: 20 00 D2 06 28 00 00 00 01 0E 5B 0E F4 1A 00 00 ; .?{.....[?..
00000020h: 0C 20 22 21 60 1C 00 00 BC 20 00 18 D4 20 00 1C ; ."!`...?..?..
00000030h: EC 20 00 24 1C 21 00 34 4C 21 00 34 7C 21 00 54 ; ?.$.!4L!.4|!.T
00000040h: C4 21 00 40 F4 21 00 14 0C 22 00 20 24 22 00 20 ; ?.0?...". $".
...
000000b0h: 44 26 00 40 74 26 00 44 BC 26 00 44 04 27 00 40 ; D&.0t&.D?.D.'.@
000000c0h: 34 27 00 3C 64 27 00 4C AC 27 00 44 F4 27 00 1C ; 4'<.d'.L?.D?..
...
00002700h: 00 00 00 00 00 00 3F FC 3F FC 30 00 30 00 30 00 ; ....???0.0.
00002710h: 30 00 30 00 3F F8 3F F8 30 00 30 00 30 00 30 00 ; 0.0.???0.0.0.
00002720h: 30 00 30 00 3F FC 3F FC 00 00 00 00 00 00 00 00 ; 0.0.???.....
00002730h: 00 00 00 00 00 00 3F F8 3F F8 30 00 30 00 30 00 ; .....???0.0.
00002740h: 30 00 30 00 3F F0 3F F0 30 00 30 00 30 00 30 00 ; 0.0.???0.0.0.
```

解析如下:

1). 文件头 (上图蓝色框选部分); 前 16 Byte

55 46 4C 11 -- 标识头, 判断是否为合法的字体文件。

55 = 'U', 表示该文件为 Unicode 编码格式的字体文件。

46 = 'F', 4C = 'L'

11 表示该字体文件版本信息为: Version 1.1

3C 52 00 00 -- 文件总长。

03 -- 包含几个 Section。03 即表示分为 3 段。

18 -- 字体高度。即表示字体大小为 24 (0x18) 点阵。

A0 00 -- 选择的字符集标志位。即 0x00A0 (00000000 10100000), 根据字符集序, 故得出当前选择为: 中欧+西欧字符集。

0F 01 -- 有效点阵字符数。即表示有 0x10F 个字符的点阵数据。

00 00 -- 预留字。

2). 段信息 (上图粉红色框选部分); $nSection * sizeof(FL_SECTION_INF) = 3 * 8 = 24Byte$

此文件分 3 段, 解析如下:

Section1.

20 00 -- First character, 即此段首字符为 0x20

D2 06 -- Last character, 即此段尾字符为 0x6D2

28 00 00 00 — 指向的是首字符（0x20）的字符信息（即检索表）的起始地址，即为 0x28.

Section2.

01 0E — First character, 即此段首字符为 0xE01

5B 0E — Last character, 即此段尾字符为 0xE5B

F4 1A 00 00 — 指向的是首字符（0xE01）的字符信息（即检索表）的起始地址，即为 0x1AF4.

Section3.

0C 20 — First character, 即此段首字符为 0x200C

22 21 — Last character, 即此段尾字符为 0x2122

60 1C 00 00 — 指向的是首字符（0x200C）的字符信息（即检索表）的起始地址，即为 0x1C60.

3). 检索表（上图绿色下划线标记部分）

$((\text{Section}[0].\text{Last} - \text{Section}[0].\text{First} + 1) * \text{sizeof}(\text{UFL_CHAR_INDEX}) + \dots + (\text{Section}[n-1].\text{Last} - \text{Section}[n-1].\text{First} + 1) * \text{sizeof}(\text{UFL_CHAR_INDEX}))$

如何快速找到一个字符的字符信息（检索表）？

- 先比较确定该字符属于哪一段。
- 精确计算其起始地址。（起始地址=字符的 unicode 码 - Section[x].First）* sizeof(UFL_CHAR_INDEX) + Section[x].OffAddr）

4). 点阵数据（上图蓝色下划线标记部分）

例如：字符大写字母 ‘E’（0x45）。

- 读取字库文件头，获取字库的基本信息。
- 确定属第几段。因为 $0x45 \geq 0x20$ && $0x45 \leq 0x6D2$ ，故可确定其在第 1 段。
- 计算点阵信息的起始地址。起始地址= $(0x45-0x20) * 4 + 0x28 = 0xBC$ 故可得出点阵信息：04 27 00 40，即为 0x40002704（01000000 00000000 00100111 00000100）。
高 6 位，表示当前字符的宽度。故得出 010000—16，即当前字符的宽度为 16。
低 26 位，表示当前字符点阵数据的偏移地址。故得出 00 00000000 00100111 00000100 ---0x2704。即当前字符点阵数据的偏移地址为 0x2704。
- 蓝色下划线 标记部分, 即为大写字母 ‘E’ 的点阵数据。因为它的字体高度为 24，宽度为 16，故其点阵数据长度为： $24 * (16+8-1) / 8 = 48$ 。

五. 字符集

➤ 支持所有 windows 字符集

CP932, 日文 Shift-JIS, 如: 日语
CP936, 简体中文 GBK, 如: 中文(中华人民共和国), 中文(香港特别行政区), 中文(新加坡)
CP949, 韩文, 如: 朝鲜语
CP950, 繁体中文 Big5, 如: 中文(台湾), 中文(澳门特别行政区)
CP874, 泰文, 如: 泰语
CP1250, 中欧, 如: 捷克语, 匈牙利语, 波兰语, 罗马尼亚语, 克罗地亚语, 斯洛伐克语, 阿尔巴尼亚语, 斯洛文尼亚语, 塞尔维亚语(拉丁文)
CP1251, 西里尔文, 如: 保加利亚语, 俄语, 乌克兰语, 比利时语, 马其顿语(FYROM), 哈萨克语, 吉尔吉斯语, 鞑靼语, 蒙古语, 阿塞拜疆语, 乌兹别克语, 塞尔维亚语
CP1252, 西欧(拉丁文 I), 如: 加泰隆语, 丹麦语, 德语, 英语, 西班牙语, 芬兰语, 法语, 冰岛语, 意大利语, 荷兰语, 挪威语, 葡萄牙语, 印度尼西亚语, 巴士克语, 南非语, 法罗语, 马来语, 斯瓦希里语, 加里西
亚语, 瑞典语
CP1253, 希腊文, 如: 希腊语
CP1254, 土耳其文, 如: 土耳其语, 阿塞拜疆语, 乌兹别克语
CP1255, 希伯来文, 如: 希伯来语
CP1256, 阿拉伯文, 如: 乌都语, 波斯语, 阿拉伯语(伊拉克, 埃及, 利比亚, 阿尔及利亚, 摩洛哥, 突尼斯, 阿曼, 也门, 叙利亚, 约旦, 黎巴嫩, 科威特, 阿联酋, 巴林, 卡塔尔)
CP1257, 波罗的海文, 如: 爱沙尼亚语, 拉脱维亚语, 立陶宛语,
CP1258, 越南, 如: 越南语

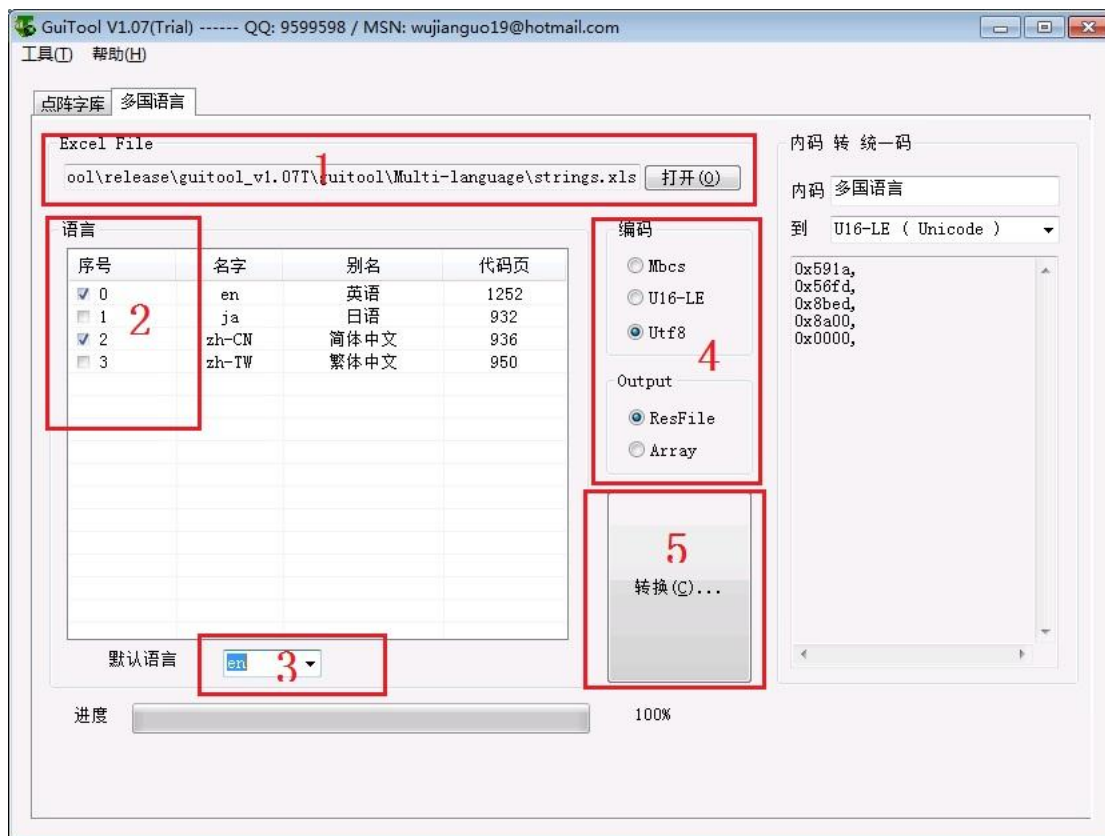
➤ 字符集顺序

0. 日文,
1. 简中,
2. 韩文,
3. 繁中,
4. 泰文,
5. 中欧,
6. 西里尔,
7. 西欧,
8. 希腊,
9. 土耳其文,
10. 希伯来文,
11. 阿拉伯文,
12. 波罗的海文,
13. 越南文。

第二章 多国语言

一. 多国语言文本管理

1. 操作简介， 操作界面如下：

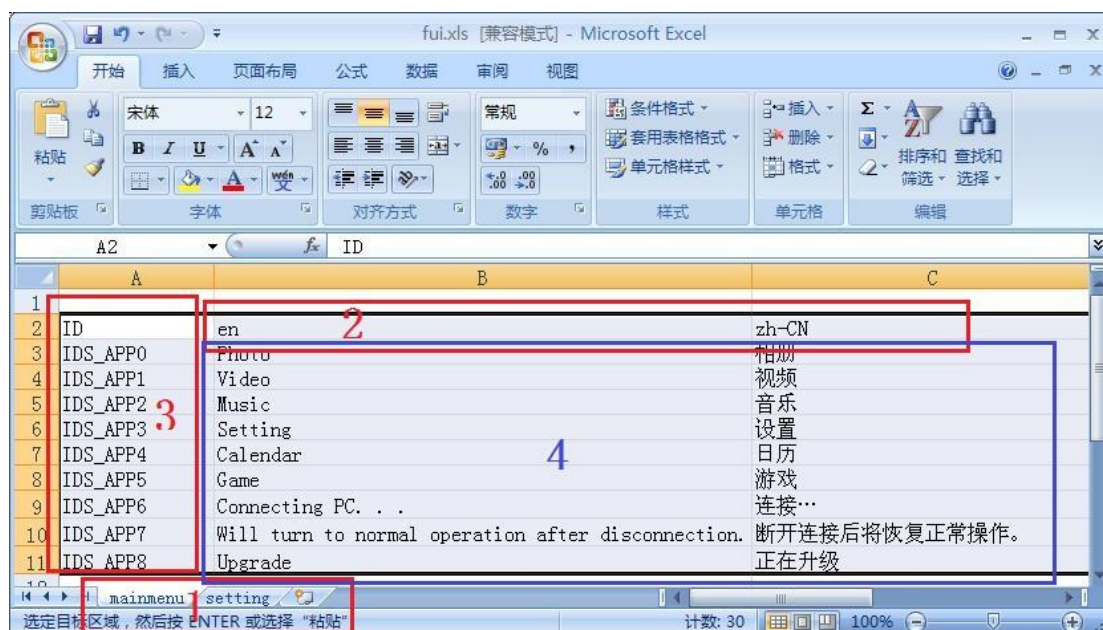


◆ 说明（见上图示红色框选部分）：

- 1). 打开 Excel 格式资源 (*.xls) 文件（详见后述）；
- 2). 选择需要支持的语言；
如出现未知（Unknow）语言，请与作者联系。
- 3). 选择开机默认语言；
注意：必须先选择支持语言，否则无法选择。
- 4). 选择编码格式及输出文件格式；
 - a. 编码格式主要配合字库使用，视情况而定。
 - b. 输出文件格式， ResFile 相当于二进制文件，需要了解文件格式，方能解析； Array 生成的是一个.h 的头文件，里面是一个个数组，包含重新编译即可。
- 5). 转换资源。

2. Excel 资源文件格式

采用 Excel 表格管理多国语言文本资源的最大优点就是**便于管理**（翻译、编辑修改），**可阅读性强**。



◆ **说明**（如上图框选部分所示）：

- 1). **子表**，一个项目的所有文本资源可以分成一个或多个子表存放，子表名称尽量做到简短，见名知意，不能重复，在输出文件中有特别意义；
- 2). **语言**，即语言短名，具体详见工具目录下的 `guitool.ini` 文件，如需添加不在支持列表中的语言，请参考已支持语言，或与作者联系；
- 3). **字符串 ID**，在同一个子表中为每一个字符串定义唯一字符串，不同子表允许出现同名字串 ID，例如：在子表 `mainmenu` 中有 `IDS_APP0`，子表 `setting` 中也可以有 `IDS_APP0`，它们通过子表名区分；
- 4). **多语言字符串**，存放的都是 unicode 编码，所以基本所见即所得。

注意：目前每个子表格第一行都为空行，其目的是预留待改日扩展用。

3. 输出文件格式

1). ResFile

文件结构中主要由文件头、一级检索表、二级检索表和字符数据四个部分组成，

其中，文件头起始位置 `FileStartAdd=0`；

一级检索表起始位置 `FirstIndexAdd=16+6*LangCnt`；

二级检索表起始位置 `SecondIndexAdd=FirstIndexAdd+（16+2+4）* SheetCnt`

字符数据起始位置 `StringAdd=SecondIndexAdd+60+4+2*LangCnt）*StrCnt`

Section	Byte (start: len)	Remark
[文件头]	0: 2B	Magic: 标识头, 固定为 “FR”
	2: 2B	Version: 0x0120 表示 v1.32
	4: 4B	FileLen: 文件总长度
	8: 2B	SheetCnt: 表格数
	10: 2B	LangCnt: 语言数
	12: 4B	Reverser
	16: 6* LangCntB	语言 ID, 例如: en<英文>, fi<法文>, zh-cn<简中>...
[一级检索表]	FirstIndexAdd: 16B	API 名 (sheet1 名), 属于主检索关键字
	FirstIndexAdd+16: 2B	StrCnt: 字符串数
	FirstIndexAdd+18: 4B	StartAddr: 指向二级索引的起始地址 (相对于文件头开始-0)
	多个 AP, 多个 sheet, 重复如上结构
[二级检索表]	SecondIndexAdd: 60B	IDS: 字符串 ID, 属于检索关键字
	SecondIndexAdd+60: 4B	StartAddr: 字符串内容的起始地址 (相对于文件头开始-0)
	SecondIndexAdd+64: 2B	Offset0: 第一种语言的长度。
	SecondIndexAdd+66: 2B	Offset1: 前两种语言的长度和。
	LandCnt 个语种, OffsetX 重复: 例如: 字符串 “CHINA”, 包含 4 种语言, 则将记录 4 个 offset (分别记录 2, 3, 4 的偏移值) 它的其实地址 (startaddr): 0x2000 ID: CHINA..... (60 B) StartAddr: 0x2000 (4B) Offset: offset0 (第一种语言的长度), offset 1 (前两种语言的长度和), offset2 (前三种语言的长度和), Offset3 (前四种语言的长度和) 最后一个 offset (此例即 Offset3) 主要是用来计算出最后一种语言字符串的长度。
[字符数据]	StringAdd: Offset	数据视编码类型而定

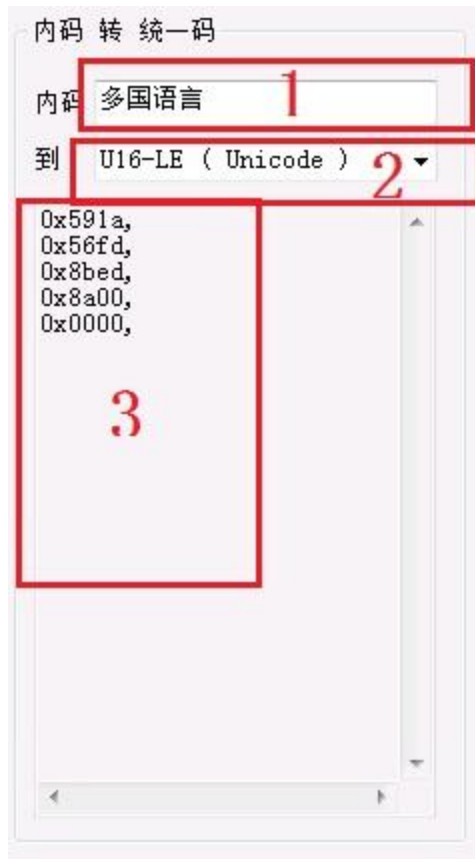
2). Array

生成出来.h 文件即标准 c, 可读性强, 在此不做详述。

二. 内码 转 Unicode

将当前键入的内码转换成指定的 Unicode 编码格式, 可以直接粘贴到程序中使用。

操作方法: 键入内码字符串, 选择转换编码即可。



◆ 说明（如上图红色框选部分所示）：

1. 键入内码字符串
2. 选择转换编码（U16-LE / U16-BE / UTF8 ）
3. 显示转换后的编码

附注

1. 注解

MBCS: Multi Byte Charset，即本地字符集（多字节字符集）。

Unicode: 即统一编码（宽字节字符集）

CJK: China, Japan, Korea，即中（简体，繁体）日韩字符集。

非等宽: 每个字符的显示宽度不等。如： 字符'I', 'M'。

等宽: 每个字符的显示宽度相等。如： CJK。

DPI: Dots per inch 的缩写，即每英寸的点数。