字体驱动模块说明文档

综述

字体驱动模块负责加载字体资源(文件,内存数据等等),解析字节码,输出字体的位图数据(bitmap).

模块包含了 freetype2 字体引擎,可以解析以下字体格式:

TTC,TTF,

BMF(ChinaChip Copyright)

模块位置

模块根目录: /lib/source/font/

主文件: font.c

接口头文件: font.h

函数接口说明

字体部分

打开一个字体文件

XFONT * xfont_open(u8 *path);

参数	u8 *path	字体文件路径
返回值	XFONT *	打开的字体句柄

字体文件打开失败则返回 NULL 值.

以 Unicode 格式读取一个字的位图数据

int xfont_uread(XFONT * xfont, int code, int size, int style,
u8 * buf, int * width, int * height, int limit);

参数	XFONT * xfont	字体句柄	
	int code	字编码(Unicode 格式)	
	int size	字体大小(像素单位)	
	int style	字体风格(见下表)	
	u8 * buf	接收字体位图的缓存	
	int * width	接收字体实际宽度的 int 型地址	
	int * height	接收字体实际高度的 int 型地址	
	int limit	字体位图缓存大小(字节单位)	
返回值	int	0	读取成功
		其他值	读取失败

1. 字编码必须是 Unicode(Little Endian)格式的.
2. 字体大小接近于像素单位,但是实际大小请读取返回值得到.
3. 字体位图格式为 256 深度的灰度(即每像素占 8 位),一个字节,以先行后列的方式线性排列. 行宽度等同于返回的字体宽度数据.

4. limit 值应该设为你提供的字体缓存的尺寸大小.
5. 返回值为非 0 时,读取字体失败,此时请不要使用返回的高度和宽度数据.

关闭一个字体

void xfont_close(XFONT * xfont);

参数	XFONT * xfont	字体句柄
返回值	无	

使用完这个字体后,你需要关闭它以释放相关的资源.不要关闭无效的句柄,不要重复关闭同一个句柄.

Unicode 字体绘制

```
void xfont_udraw( XFONT * xfont, int code, int size, int style, u32 color, u32
bgcolor, COLOR_MODE color_mode, XFONT_BLEND_MODE blending_mode,
 int startx, int starty, int pitch,
 void * buf, int * width, int * height, int limit );
```

参数	XFONT * xfont	字体句柄
	int code	字编码(Unicode 格式)
	int size	字体大小(像素单位)
	int style	字体风格(见下表)
	u32 color	字体颜色
	u32 bgcolor	背景颜色
	COLOR_MODE color_mode	颜色模式(见下表)
	XFONT_BLEND_MODE blending_mode	颜色混合模式
	int startx	绘制起始 x 坐标
	int starty	绘制起始 y 坐标
	int pitch	行宽度
	u8 * buf	绘制输出缓存
	int * width	接收实际字体宽度
	int * height	接收实际字体高度
	int limit	缓存大小限制(字节单位)
返回值	无	

1. 这个函数其实是 xfont_uread 的高级版本,提供了直接到屏幕色彩的绘制功能.
2. 字体颜色必须是和你设置的颜色模式相对应.
3. 缓存限制请按实际设置,实际的值应该是 参数 buf 到这个 buffer 结尾之间的字节数.
4. 行宽度是每一行的像素数目.
5. startx, starty 基本上是字体的左下角坐标,但有些字体可能会往下后者往上多显示几个像素,具体取决于字体本身.

可用的字体风格

XFONT_STYLE_NORMAL	正常字体
XFONT_STYLE_BOLD	粗体

XFONT_STYLE_ITALIC	斜体

可用的颜色模式

LCD_A8R8G8B8	GUI 默认使用的色彩模式 32 位 ARGB 色彩

颜色混合模式

XFONT_BLEND_MODE_COVER	用指定色覆盖
XFONT_BLEND_MODE_ALPHA	与原背景混合

获取字体尺寸

```
int xfont_getsize( XFONT * xfont, int code, int size, int style, XFONT_INFO * info );
```

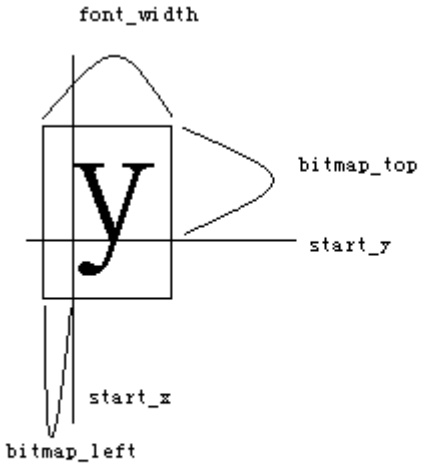
参数	XFONT * xfont	字体对象
	int code	字 Unicode 编码
	int size	字体大小(像素单位)
	int style	字体风格
	XFONT_INFO * info	用于接受字体信息的结构指针
返回值	int	返回 0 成功,其他值失败

字体信息 XFONT_INFO 的结构如下:

```
typedef struct XFONT_INFO {
    int bitmap_width;
    int bitmap_height;
    int bitmap_left;
    int bitmap_top;
    int font_width;
} XFONT_INFO;
```

int bitmap_width	字体位图宽度
int bitmap_height	字体位图高度
int bitmap_left	字体位图左偏移
int bitmap_top	字体位图顶偏移
int font_width	字体宽度

下面以一张图说明字体的显示位置:



多国语言部分

该模块可以进行多国语言转换,将各国的本地码转换成 Unicode-16 编码

打开一个多国语言转换句柄

```
XFONT_NLS * xfont_nls_open( XFONT_CODE_PAGE code_page );
```

参数	XFONT_CODE_PAGE code_page	需要转换的代码页
返回值	XFONT_NLS *	返回的对象句柄

1. 如果返回值为 NULL,则这个转换对象打开失败,其原因可能是待转换的代码页不存在.

关闭一个多国语言转换句柄

```
int xfont_nls_close( XFONT_NLS * hnls );
```

参数	XFONT_NLS * hnls	需要关闭的转换句柄
返回值	int	返回 0 成功,其他值失败

1. 不要关闭同一个句柄,不要关闭不存在的,或者 NULL 的对象.

转换一个本地码到 Unicode

```
u16 xfont_nls_a2u( XFONT_NLS * hnls, u16 achar );
```

参数	XFONT_NLS * hnls	转换对象句柄
	u16 achar	本地码表示的待转换字符

返回值	u16	Unicode-16 表示的字符

1. 该函数没有错误返回码,对于不可识别的转换,函数均返回 0 值.

转换一个 Unicode 到本地码

```
u16 xfont_nls_u2a( XFONT_NLS * hnls, u16 uchar );
```

参数	XFONT_NLS * hnls	转换对象句柄
	u16 achar	Unicode 表示的待转换字符
返回值	u16	本地码表示的字符

1. 该函数没有错误返回码,对于不可识别的转换,函数均返回 0 值.

XFONT_CODE_PAGE 代码页解释

XFONT_CP_DEFAULT	使用由系统设定的默认值
XFONT_CP_437	美国英语
XFONT_CP_874	泰国语
XFONT_CP_932	日语
XFONT_CP_936	简体中文
XFONT_CP_949	朝鲜语
XFONT_CP_950	繁体中文
XFONT_CP_1250	中欧语言
XFONT_CP_1251	西里尔语
XFONT_CP_1253	希腊语
XFONT_CP_1254	土耳其语
XFONT_CP_1256	阿拉伯语

并不仅限于这个列表,将来可能还会添加更多的代码页支持.

字体高速缓存部分

由于矢量字体实时渲染速度比较慢,重复渲染已经存在的字体,严重浪费 cpu 资源,因此 XFONT 内建了一个字体高速缓存,用来快速读取字体数据.

这个缓存对于用户程序来说完全是透明的,只要你使用初始化函数激活这个缓存模块,它就会默默的起作用了.激活缓存模块需要耗费一定数量的内存,所以在程序不需要字体引擎时,务必把它关闭.

初始化(激活)字体高速缓存

```
int xfont_cache_init( void );
```

参数	无	
返回值	int	返回 0 成功,其他失败

卸载字体高速缓存

```
int xfont_cache_deinit( void );
```

参数	无	
返回值	int	返回 0 成功,其他失败

有关字体高速缓存的设置:(全局变量)

```
extern int xfont_cache_memory_size;        //xfont 内置的 cache 占用内存大小(字节单位)
extern int xfont_cache_font_max_size;      //xfont 内置的 cache 最大字体大小(像素单位)
```

公共字体绘制函数(TEXTOUT)

该函数实现一个统一风格的字体绘制,用来统一绘制 GUI 字体,简单快速的显示字符等等.

已经添加一个特殊的功能,可以不初始化就使用 textout,也可以在初始化失败的时候使用 textout,此时仅可以使用 8*16 点阵,输出 ASCII 字符(英文字符).这个功能是用来给当用户没有提供任何的 platform 支持的情况下,"能够艰难的工作下去",或者给用户一个英文的错误提示.

初始化 textout 模块

```
int textout_init( char * fname, int size, int line_height, int word_spacing,
int style, XFONT_CODE_PAGE codepage );
```

参数	char * fname	textout 使用的字体文件路径
	int size	字体大小(像素单位)
	int line_height	行高
	int word_spacing	字间距
	int style	字体风格
	XFONT_CODE_PAGE codepage	使用的代码页
返回值	int	0 成功,其他失败

1. 返回失败值的时候,textout 模块并不可用,调用 textout 输出字符将使用内置的 8*16 点阵英文输出.
2. 字体文件路径可以仅使用文件名,此时默认在 xfont_fonts_path 中查找字体
3. xfont_default_font 全局变量是由 INI 配置的字体路径,可以读取这个变量的值作为字体路径参数.
4. 字间距是指两个字体之间的空白宽度,以像素为单位.
5. 代码页可以使用 XFONT_CP_DEFAULT 值,让系统使用 INI 配置的代码页,等效于使用 xfont_default_codepage 参数.

卸载 textout 模块

```
int textout_deinit( void );
```

参数	无	
返回值	int	0 成功,其他失败

1. 在结束使用 textout 模块后,务必关闭它以释放资源.

使用 textout 显示一行文字

ANSI 版本

```
STRING_INFO * textout_line( const char *string, int min_x, int min_y, int max_x,
int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode版本

```
STRING_INFO * textout_lineW( const wcar_t  *string, int min_x, int min_y, int
max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

参数	* string	待显示的字符串
	int min_x	起始x坐标
	int min_y	起始y坐标
	int max_x	结束x坐标
	int max_y	结束y坐标
	unsigned int fontcolor	字体颜色
	unsigned int bgcolor	背景颜色
返回值	STRING_INFO *	字符串属性

1. min_x, min_y, max_x, max_y限定了一个矩形区域,字符串将在这个区域中绘制,如果字符串长度大于这个区域,或者遇到换行符,则函数将截断显示.
2. 背景颜色可以使用alpha通道,在颜色最高8位存储,如0x80FFFFFF
3. STRING_INFO *是一个指向当前字符串的属性结构指针,

```
typedef struct STRING_INFO{
    s32 width;
    s32 height;
    s32 length;
} STRING_INFO;
```

s32 width	字符串宽度
s32 height	字符串高度
s32 length	已经显示的字符串长度(字节)

显示多行字符串

ANSI 版本

```
STRING_INFO * textout( const char * string, int min_x, int min_y, int max_x,
int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode版本

```
STRING_INFO * textoutW( const char * string, int min_x, int min_y, int max_x,
int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

1.该函数的参数意义与 textout_line 完全一致,不同点在于这个函数可以自动在限定的矩形区域内换行显示字符串.
2.其他注意事项同 textout_line

内存 textout 输出

32 位内存输出

ANSI 编码输出单行字符

```
STRING_INFO * mtextout_line( u32 * buf, int bufwidth, int bufheight, const char *string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

ANSI 编码输出多行字符

```
STRING_INFO * mtextout( u32 *buf, int bufwidth, int bufheight, const char * string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode 编码输出单行字符

```
STRING_INFO * mtextoutW_line( u32 * buf, int bufwidth, int bufheight, const wchar_t *string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode 编码输出多行字符

```
STRING_INFO * mtextoutW( u32 *buf, int bufwidth, int bufheight, const wchar_t * string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

16 位内存输出

ANSI 编码输出单行字符

```
STRING_INFO * mtextout16_line( u16 * buf, int bufwidth, int bufheight, const char *string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

ANSI 编码输出多行字符

```
STRING_INFO * mtextout16( u16 *buf, int bufwidth, int bufheight, const char * string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode 编码输出单行字符

```
STRING_INFO * mtextoutW16_line( u16 * buf, int bufwidth, int bufheight, const wchar_t *string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor );
```

Unicode 编码输出多行字符
STRING_INFO * mtextoutW16(u16 *buf, int bufwidth, int bufheight, const wchar_t * string, int min_x, int min_y, int max_x, int max_y, unsigned int fontcolor, unsigned int bgcolor);

GUI 模拟控制台函数

该模块使用用户屏幕"仿造"一个控制台窗口,可以实现 printf 输出的功能,用来在没有串口的情况下打印调试信息.

这个模块调用了 textout 实现字符绘制,所以请先初始化 textout,这个模块才可用.

格式化输出
```
void ui_printf( const char *format ,...);
```
用法和 printf 一样,就不多说了.

指定坐标的格式化输出
```
void lprintf( u32 x, u32 y ,const char *format ,...);
```

参数	u32 x	起始 x 坐标
	u32 y	起始 y 坐标
	const char *format ,...	格式化字符串

清空控制台窗口
```
void DbgCon_Cls( void );
```

将控制台窗口显示坐标复位到左上角
```
void DbgCon_Reset( void );
```

设置控制台颜色
```
void DbgCon_SetColor( u32 font, u32 bg );
```

参数	u32 font	字体颜色
	u32 bg	背景颜色
返回值	无	

1. 背景颜色可以使用 Alpha 通道