

## 第13章 窗口对象（控件）

---

控件是具有对象性质的窗口，在视窗世界中它们被称为控件，构造用户接口的元素。它们能自动对某些事件起反应；例如，当一个按钮被按下时，它可能以不同的状态显示。控件需要建立，具有能在它们存在的任何时间修改的特性，在它们不再需要时被删除。与窗口类似，一个控件通过它的建立函数返回的句柄而引用。

控件需要使用视窗管理器。一旦一个控件被建立，它被处理成与其它窗口一样；WM 保证它在需要的时候能正确的显示（及重绘）。当编写一个应用或一个用户接口时，控件并不需要，但是它们能使编程更容易。

## 13.1 一些基础知识

### 有效的控件

下面的  $\mu$ C/GUI 控件在当前是有效的:

名 称	说 明
BUTTON	可以按下的按钮。在按钮上可以显示文本或位图。
CHECKBOX	复选框, 提供多个选项。
EDIT	单行文本编辑框, 提示用户输入数字或文本。
FRAMEWIN	框架窗口, 建立典型的 GUI 外形。
LISTBOX	列表框, 当被选中的项目会高亮显示。
PROGBAR	进度条, 用于观察。
RADIOBUTTON	单选按钮, 可以选择。同一时间只有一个按钮被选中。
SCROLLBAR	滚动条, 可以是水平或垂直的。
SLIDER	滑动条, 用于改变数值。
TEXT	文本控件, 典型应用在对话框中。

### 理解重绘机制

一个控件根据它的特性绘制自己。这一工作通过调用 WM 的 API 函数 `WM_Exec()` 来完成。如果在程序中没有调用 `WM_Exec()`, 就必须调用 `WM_Paint()` 函数来绘制控件。在多任务环境中的  $\mu$ C/GUI, 一个后台任务通常用于调用 `WM_Exec()` 并更新控件 (及其它所有带有回调函数的窗口)。这样就不必手工调用 `WM_Paint()`; 然而, 手工调用仍然是合法的, 如果你想保证控件能立即被重绘的话, 这样做也是有意义的。

当一个控件的属性被改变时, 控件的窗口 (或者它的一部分) 被标记为无效, 但是它不会立即重绘。因此, 这部分代码运行非常快。重绘在后面的时间通过 WM 完成, 或通过为控件调用 `WM_Paint()` 函数 (或者 `WM_Exec()`, 直到所有的窗口都被重绘) 来强制执行。

### 如何使用控件

假设我们要显示一个进度条。需要写以下的代码:

```
PROGBAR_Handle hProgBar;
GUI_DispStringAt("Progress bar", 100, 20);
hProgBar = PROGBAR_Create(100, 40, 100, 20, WM_CF_SHOW);
```

第一行为控件的处理保留内存。最后一行实际建立控件。如果晚一些时候或在一个单独

任务中 `WM_Exec()` 被调用的话，控件会被视窗管理器自动绘出。



对于每一种控件来说，成员函数是有效的，通过它们允许修改控件的外观。一旦控件被建立，它的特性可以通过调用一个它的成员函数来改变。这些函数以控件的句柄作为它们的第一个参数。为了建立一个进度条，上面显示 45%，并将其颜色从默认值（深灰/浅灰）改为绿/红，要使用到以下部分代码：

```
PROGBAR_SetBarColor(hProgBar, 0, GUI_GREEN);  
PROGBAR_SetBarColor(hProgBar, 1, GUI_RED);  
PROGBAR_SetValue(hProgBar, 45);
```



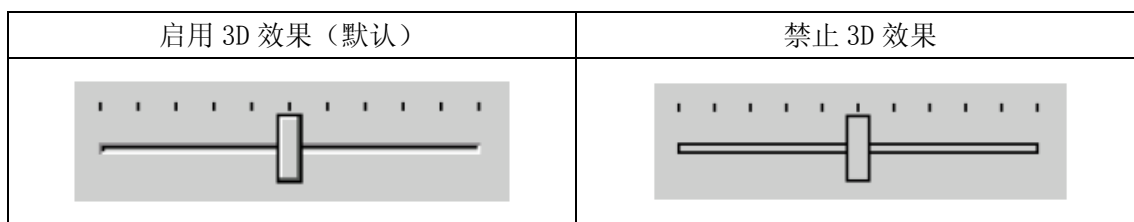
所有的控件也都有一个或多个配置宏，用于定义不同的默认设置，诸如字体和使用的颜色等。在本章后面每个控件各自部分，对应的有效的配置选项都会列出来。

### 用于控件的动态存储器

在嵌入式应用当中，通常来说，使用动态存储器确实不是非常合适，因为存储残片效果的缘故。有很多不同的方式可以用来这种情况，但是它们都工作在一个受限制的方式里，随时内存区域都可能被应用程序中的一个指针引用。因为这个原因， $\mu$ C/GUI 使用一个不同的方法：所有物体（以及所有在运行时存储的数据）被存入一个句柄引用的内存区域当中。这会使已分配好的内存区域在运行时重新分配成为可能。因而避免了使用指针时出现的长时间分配的问题。所有控件因此通过句柄引用。

### 3D 支持

一些控件能够以 3D 或非 3D 效果显示。默认情况下是启用 3D 支持，但是也可能通过将配置宏 `<WIDGET>_USE_3D` 设置为 0 而禁止它。不管它是否使用三维效果，一个控件的功能完全一样的；唯一的不同是它的外观。下面是一个滑块控件的范例：



## 13.2 通用控件 API 函数

### API 参考：用于控件的 WM 函数

既然控件本质上是一个窗口，那么任何一种视窗管理器 API 函数都适用于它们。控件的句柄作为 `hWin` 参数，控件象其它窗口一样处理。最普遍用于控件的 WM 函数在下表列出：

函数	说明
<code>WM_DeleteWindow()</code>	删除一个窗口。
<code>WM_DisableMemdev()</code>	禁止存储设备的重绘功能。
<code>WM_EnableMemdev()</code>	启用存储设备的重绘功能。
<code>WM_InvalidateWindow()</code>	使一个窗口无效。
<code>WM_Paint()</code>	立即绘制或重绘一个窗口。

对于全部与 WM 相关的函数目录，请参阅第 12 章：视窗管理器。

### API 参考：所有控件共有的函数

下表按字母顺序列出了与控件相关的函数。这些函数适用于所有的控件。在此列出是为了避免重复。函数的详细描述在后面给出。每个控件附加的成员函数在后面部分可以找到。

函数	说明
<code>&lt;WIDGET&gt;_CreateIndirect()</code>	在对话框中自动创建时使用。
<code>WM_EnableWindow()</code>	将控件状态设置为启用 (默认)。
<code>WM_DisableWindow()</code>	将控件状态设置为禁止。

#### `<WIDGET>_CreateIndirect()`

##### 描述

建立一个用于对话框中的控件。

##### 函数原型

```
<WIDGET>_Handle <WIDGET>_CreateIndirect(    const GUI_WIDGET_CREATE_INFO*
                                             pCreateInfo, WM_HWIN hParent,
                                             int x0, int y0,
                                             WM_CALLBACK* cb);
```

参 数	含 意
<code>pCreateInfo</code>	指向一个 <code>GUI_WIDGET_CREATE_INFO</code> 结构 (下面提到) 的指针。
<code>hParent</code>	父窗口的句柄。
<code>x0</code>	控件最左边的像素 (桌面座标)。

### 附加信息

通过使用适当的前缀, 任何控件都可能被间接的建立。

例如: `BUTTON_CreateIndirect()` 间接建立一个按钮控件, `CHECKBOX_CreateIndirect()` 间接建立一个复选框控件, 等等。

如果一个控件包括在一个对话框当中, 它只需要间接建立。否则, 它可以通过使用 `<WIDGET>_Create()` 函数直接建立。更多关于对话框的信息, 请参阅第 14 章: 对话框。

### 资源表

对话框资源表中的 `GUI_WIDGET_CREATE_INFO` 结构定义如下:

```
typedef struct
{
    const char* pName;           // 文本 (不是所有的控件都使用)
    I16 Id;                      // 控件的窗口 ID
    I16 x0, y0, xSize, ySize;    // 控件的大小和位置
    I16 Flags;                   // 控件专用标识 (或 0)
    I32 Para;                   // 控件专用参数 (或 0)
} GUI_WIDGET_CREATE_INFO;

GUI_WIDGET_CREATE_FUNC* pfCreateIndirect;    // 创建函数
```

控件标识和参数是可选的, 这非常依赖于控件的类型。每个控件的标识和参数 (如果存在) 会在本章后面的适当部分列出。

## WM\_EnableWindow()

### 描述

将一个指定控件状态设置为激活（启用）。

### 函数原型

```
void WM_EnableWindow(WM_Handle hObj);
```

参 数	含 意
<a href="#">hObj</a>	控件的句柄。

### 附加信息

对于任何控件来说，这是默认设置。

## WM\_DisableWindow()

### 描述

将一个指定的控件状态设置为非活动（无用）。

### 函数原型

```
void WM_DisableWindow(WM_Handle hObj);
```

参 数	含 意
<a href="#">hObj</a>	控件的句柄。

### 附加信息

一个被禁止的控件以灰色显示，不会接收用户的输入。但其实际外观可以改变（根据 控件/配置 的设置等等）

## 13.3 BUTTON: 按钮控件

一般情况下，按钮控件作为触摸屏主要的用户接口部件使用。按钮可以显示文本，如下面所示，或者一幅位图。



所有与按钮相关的函数仅次于文件 `BUTTON*.c`，`BUTTON.h` 当中。所有的标识是前缀“BUTTON”。

### 配置选项

类型	宏	默认值	说明
N	<code>BUTTON_3D_MOVE_X</code>	1	按下状态文本/位图在水平方向偏移的像素数。
N	<code>BUTTON_3D_MOVE_Y</code>	1	按下状态文本/位图在垂直方向偏移的像素数。
N	<code>BUTTON_BKCOLOR0_DEFAULT</code>	<code>0xAAAAAA</code>	非按下状态的背景颜色。
N	<code>BUTTON_BKCOLOR1_DEFAULT</code>	<code>GUI_WHITE</code>	按下状态的背景颜色。
S	<code>BUTTON_FONT_DEFAULT</code>	<code>&amp;GUI_Font13_1</code>	按钮文本的字体。
N	<code>BUTTON_TEXTCOLOR0_DEFAULT</code>	<code>GUI_BLACK</code>	非按下状态的文本颜色。
N	<code>BUTTON_TEXTCOLOR1_DEFAULT</code>	<code>GUI_BLACK</code>	按下状态的文本颜色。
B	<code>BUTTON_USE_3D</code>	1	启用3D效果支持。

默认状态下按钮按下时背景色是白色，是特意这样做的，因为这会使按钮按下时在任何显示屏上显示得更明显些。如果你想让按钮的背景色无论是按下还是不按下的状态下都是一样的，需将 `BUTTON_BKCOLOR1_DEFAULT` 改为 `BUTTON_BKCOLOR0_DEFAULT`。

### BUTTON API 函数

下表按字母顺序列出了  $\mu\text{C}/\text{GUI}$  与按钮有关的函数，详细的描述在本章后面部分绘出。

函数	说明
<code>BUTTON_Create()</code>	建立按钮。
<code>BUTTON_CreateAsChild()</code>	将按钮作为一个子窗口建立。
<code>BUTTON_CreateIndirect()</code>	从资源表项目中建立按钮。
<code>BUTTON_SetBitmap()</code>	设置按钮显示时使用的位图。
<code>BUTTON_SetBitmapEx()</code>	设置按钮显示时使用的位图。
<code>BUTTON_SetBkColor()</code>	设置按钮的背景颜色。
<code>BUTTON_SetFont()</code>	选择文本字体。
<code>BUTTON_SetState()</code>	设置按钮状态（由触摸屏模块自动处理）。
<code>BUTTON_SetStreamedBitmap()</code>	设置按钮显示时使用的位图。
<code>BUTTON_SetText()</code>	设置文本。
<code>BUTTON_SetTextColor()</code>	设置文本的颜色。

**BUTTON\_Create()****描述**

在一个指定位置，以指定的大小建立一个 BUTTON 控件。

**函数原型**

```
BUTTON_Handle BUTTON_Create(    int x0, int y0,
                                int xsize, int ysize,
                                int ID,
                                int Flags);
```

参 数	含 意
x0	按钮最左边的像素（在桌面坐标中）。
y0	按钮最顶部的像素（在桌面坐标中）。
xsize	按钮的水平尺寸（以像素为单位）。
ysize	按钮的垂直尺寸（以像素为单位）。
ID	按钮按下时返回的 ID。
Flags	窗口建立标识。具有代表性的，WM_CF_SHOW 是为了使控件立即可见（请参阅第 12 章“视窗管理器”中 WM_CreateWindow() 的参数值列表）。

**返回数值**

所建立的按钮控件的句柄，如果函数执行失败，则返回 0。

**BUTTON\_CreateAsChild()****描述**

以子窗口的形式建立一个按钮。

**函数原型**

```
BUTTON_Handle BUTTON_CreateAsChild( int x0, int y0,
                                      int xsize, int ysize,
                                      WM_HWIN hParent,
                                      int ID,
```



```
int Flags);
```

参 数	含 意
<code>x0</code>	按钮最左边的像素（在桌面坐标中）。
<code>y0</code>	按钮最顶部的像素（在桌面坐标中）。
<code>xsize</code>	按钮的水平尺寸（以像素为单位）。
<code>yssize</code>	按钮的垂直尺寸（以像素为单位）。
<code>hParent</code>	父窗口的句柄。如果为 0，新的按钮窗口作为桌面（顶层窗口）的子窗口。
<code>ID</code>	按钮按下时返回的 ID。
<code>Flags</code>	窗口建立标识（参阅 <code>BUTTON_Create()</code> ）。

### 返回数值

所建立的按钮控件的句柄，如果函数执行失败，则返回 0。

### **BUTTON\_CreateIndirect()**

函数原型的说明在本章开始部分。The elements `Flags` and `Para` of the resource passed as parameter are not used.

### **BUTTON\_SetBitmap()**

#### 描述

显示一个指定按钮时使用的位图。

#### 函数原型

```
void BUTTON_SetBitmap( BUTTON_Handle hObj,
                      int Index,
                      const GUI_BITMAP* pBitmap);
```

参 数	含 意
<code>hObj</code>	按钮的句柄。
<code>Index</code>	位图的索引（参阅下表）。
<code>pBitmap</code>	位图结构的指针。

#### 参数 `Index` 的允许值

0	设置按钮未按下时使用的位图。
---	----------------

1	设置按钮按下时使用的位图。
---	---------------

**BUTTON\_SetBitmapEx()****描述**

显示一个指定按钮时使用的位图。

**函数原型**

```
void BUTTON_SetBitmapEx(    BUTTON_Handle hObj,
                           int Index,
                           const GUI_BITMAP* pBitmap,
                           int x, int y) ;
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">Index</a>	位图的索引（参阅 <a href="#">BUTTON_SetBitmap()</a> ）。
<a href="#">pBitmap</a>	位图结构的指针。
<a href="#">x</a>	位图相对按钮的 X 轴坐标。

**BUTTON\_SetBkColor()****描述**

设置按钮的背景颜色。

**函数原型**

```
void BUTTON_SetBkColor(BUTTON_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">Index</a>	颜色的索引（参阅下表）。
<a href="#">Color</a>	设置的背景颜色。

**参数 [Index](#) 的允许值**

0	设置按钮未按下时使用的颜色；
1	设置按钮按下时使用的颜色。

## BUTTON\_SetFont()

### 描述

设置按钮字体。

### 函数原型

```
void BUTTON_SetFont(BUTTON_Handle hObj, const GUI_FONT* pFont);
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">pFont</a>	字体的指针。

### 附加信息

如果没有选择字体，将使用 `BUTTON_FONT_DEF` 字体。

## BUTTON\_SetState()

### 描述

设置一个指定按钮物体的状态。

### 函数原型

```
void BUTTON_SetState(BUTTON_Handle hObj, int State)
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">State</a>	按钮的状态（参阅下表）。

### 参数 [State](#) 允许的值

<a href="#">BUTTON_STATE_PRESSED</a>	按钮被按下。
<a href="#">BUTTON_STATE_INACTIVE</a>	按钮处于非活动状态。

### 附加信息

该函数用于触摸屏模块。通常你不必调用这个函数。

## BUTTON\_SetStreamedBitmap()

### 描述

显示一个指定按钮对象时设置使用的位图数据流。

### 函数原型

```
void BUTTON_SetStreamedBitmap( BUTTON_Handle hObj,
                               int Index,
                               const GUI_BITMAP_STREAM* pBitmap);
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">Index</a>	位图的索引 (参阅 <a href="#">BUTTON_SetBitmap()</a> )。
<a href="#">pBitmap</a>	一个位图数据流的指针。

### 附加信息

为了能使用这个函数, 你必须在 LCDConf.h 中包括下面一行:

```
#define BUTTON_SUPPORT_STREAMED_BITMAP 1
```

想了解关于 `streamed` 位图的详细情况, 请参阅第 6 章的 `GUI_DrawStreamedBitmap()` 函数。

## BUTTON\_SetText()

### 描述

设置在按钮上显示的文本。

### 函数原型

```
void BUTTON_SetText(BUTTON_Handle hObj, const char* s);
```

参 数	含 意
<a href="#">hObj</a>	按钮的句柄。
<a href="#">s</a>	显示的文本。

**BUTTON\_SetTextColor()****描述**

设置按钮文本的颜色。

**函数原型**

```
void BUTTON_SetTextColor(BUTTON_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	按钮的句柄。
<code>Index</code>	文本颜色的索引（参阅 <code>BUTTON_SetBkColor()</code> ）。
<code>Color</code>	设置的文本颜色。

**范例****使用 BUTTON 控件**

下面的范例展示如何使用两幅位图创建一个简单的按钮。本范例是  $\mu$ C/GUI 提供的范例 `WIDGET_SimpleButton.c` :

```

/*-----*/
文件:      WIDGET_SimpleButton.c
目的:      展示一个按钮控件使用的例子
-----*/

#include "gui.h"
#include "button.h"

/*****
*          按钮控件使用的展示          *
*****/

static void DemoButton(void)
{
    BUTTON_Handle hButton;
    int Key = 0;
    GUI_Init() ;

```

```

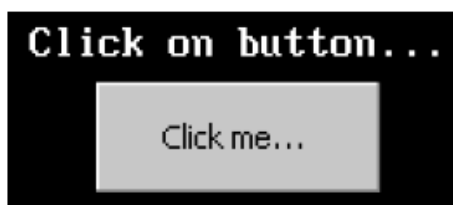
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Click on button...", 160, 0);
    /* 建立按钮 */
    hButton = BUTTON_Create(110, 20, 100, 40, GUI_ID_OK, WM_CF_SHOW);
    /* 设置按钮文本 */
    BUTTON_SetText(hButton, "Click me...");
    Key = GUI_WaitKey();
    /* 删除按钮对象 */
    BUTTON_Delete(hButton);
}

/*****
*                                     主函数                                     *
*****/

void main(void)
{
    GUI_Init();
    DemoButton();
}

```

上面范例执行结果的截图



### BUTTON 控件的高级应用

下面范例展示如何创建一个显示一幅电话图片的按钮。当按钮按下时，图片变成话筒提起的样子。本范例是μC/GUI 提供的范例 WIDGET\_PhoneButton.c:

```

/*-----
文件:      WIDGET_PhoneButton.c
目的:      展示一个按钮控件使用的例子
-----*/

```



```

static const unsigned char acPhone1[] = {

    __XX, X__, ____, ____,
    ____XXXX, XXXXX__, ____, ____,
    ____XXXX, XXXXXXXX__, ____, ____,
    ____XXXXX, XXXXXXXXX, X__, ____,
    ____XXXXX, XXXXXXXXX, XXX__, ____,
    ____XXX, XXXX__X__, XXXXX__, ____,
    ____X, XXXX__X, XXXXXXXX__, ____,
    ____, __XX__, __XXXXXXX, X__,
    ____, ____, __XXXX, XXX__,
    ____, ____, __XXX, XXXXX__,
    ____, ____, __X, XXXXXXX__,
    ____, ____, ____, XXXXXXXX__,
    ____, ____, ____, XXXXXXXX__,
    ____, ____, __X, XXXXXXXXX,
    ____, ____, __X, XXXXXXXXX,
    ____, __XX__, __XX__, __XXXXXXX,
    ____, __XX__, __XX__, __XXXX__,
    ____, __XX__, __XX__, __XX__,
    ____, __XX__, __XX__, ____,
    ____X, XXXXXXXXXX, XXXXXXXXXX, X__,
    ____XX, XXXXXXXXXX, XXXXXXXXXX, XX__,
    ____XXX, XXXX__X__, __X__XXXX, XXX__,
    ____XXXX, XXXX__X__, __X__XXXX, XXXX__,
    ____XXXXX, XXXXXXXXXX, XXXXXXXXXX, XXXXX__,
    ____XXXXX, XXXX__X__, __X__XXXX, XXXXX__,
    ____XXXXX, XXXX__X__, __X__XXXX, XXXXX__,
    ____XXXXX, XXXXXXXXXX, XXXXXXXXXX, XXXXX__,
    ____XXXXX, XXXX__X__, __X__XXXX, XXXXX__,
    ____XXXXX, XXXX__X__, __X__XXXX, XXXXX__,
    ____XXXXX, XXXXXXXXXX, XXXXXXXXXX, XXXXX__,
    ____XXXXX, XXXXXXXXXX, XXXXXXXXXX, XXXXX__
};

```

```

static const GUI_BITMAP bm_1bpp_0 = {32, 31, 4, 1, acPhone0, &Palette};
static const GUI_BITMAP bm_1bpp_1 = {32, 31, 4, 1, acPhone1, &Palette};

```

```

/*****
*                               按钮控件使用的展示                               *
*****/

```



```

static void DemoButton(void)
{
    BUTTON_Handle hButton;
    int Stat = 0;
    GUI_Init();
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Click on phone button...", 160, 0);
    /* 创建按钮 */
    hButton = BUTTON_Create(142, 20, 36, 40, GUI_ID_OK, WM_CF_SHOW);
    /* 修改按钮属性 */
    BUTTON_SetBkColor(hButton, 1, GUI_RED);
    BUTTON_SetBitmapEx(hButton, 0, &bm_1bpp_0, 2, 4);
    BUTTON_SetBitmapEx(hButton, 1, &bm_1bpp_1, 2, 4);
    /* 循环, 直到按钮按下 */
    while(GUI_GetKey() != GUI_ID_OK)
    {
        if(Stat ^= 1)
        {
            BUTTON_SetState(hButton, BUTTON_STATE_HASFOCUS |
                BUTTON_STATE_INACTIVE);
        }
        else
        {
            BUTTON_SetState(hButton, BUTTON_STATE_HASFOCUS |
                BUTTON_STATE_PRESSED);
        }
        GUI_Delay(500);
    }
    /* 删除按钮对象 */
    BUTTON_Delete(hButton);
}

/*****
*                               主函数                               *
*****/

void main(void)

```

```

{
    GUI_Init();
    DemoButton();
}

```

上面范例执行结果的截图



## 13.4 CHECKBOX: 复选框控件

最常见的多选项的选择的控件中的一个复选框。一个复选框可由用户决定是选中还是未选中，在同一时间可以有多个复选框选中。如果复选框被禁止，它会以灰色显示。如下表所看到的一样，下表图示了四种可能的显示状态：

	选中	未选中
允许	<input checked="" type="checkbox"/>	<input type="checkbox"/>
禁止	<input checked="" type="checkbox"/>	<input type="checkbox"/>

所有与复选框相关的函数位于文件 CHECKBOX\*.c, CHECKBOX.h 当中。所有标识符是前缀 CHECKBOX。

### 配置选项

类型	宏	默认值	说明
N	CHECKBOX_BKCOLOR0_DEFAULT	0x808080	禁止状态的背景颜色。
N	CHECKBOX_BKCOLOR1_DEFAULT	GUI_WHITE	允许状态的背景颜色。
N	CHECKBOX_FGCOLOR0_DEFAULT	0x101010	禁止状态的前景颜色。
N	CHECKBOX_FGCOLOR1_DEFAULT	GUI_BLACK	允许状态的前景颜色。
S	CHECKBOX_FONT_DEFAULT	&GUI_Font13_1	复选标志使用的字体。
B	CHECKBOX_USE_3D	1	3D 效果支持启用。

## 复选框 API 函数

下表按字母顺序列出了  $\mu$ C/GUI 与复选框相关的函数。函数详细的描述在下面给出。

函 数	说 明
<a href="#">CHECKBOX_Check()</a>	将复选框设置为选中状态。
<a href="#">CHECKBOX_Create()</a>	创建复选框。
<a href="#">CHECKBOX_CreateIndirect()</a>	从资源表项目中创建复选框。
<a href="#">CHECKBOX_IsChecked()</a>	返回复选框当前的状态（选中或未选中）。
<a href="#">CHECKBOX_Uncheck()</a>	将复选框设置为未选中状态（默认）。

### CHECKBOX\_Check()

#### 描述

将一个复选框设置为选中状态。

#### 函数原型

```
void CHECKBOX_Check(CHECKBOX_Handle hObj);
```

参 数	含 意
<a href="#">hObj</a>	复选框的句柄。

### CHECKBOX\_Create()

#### 描述

在一个指定位置，以指定的大小建立一个复选框的控件。

#### 函数原型

```
CHECKBOX_Handle CHECKBOX_Create( int x0, int y0,
                                  int xsize, int ysize,
                                  WM_HWIN hParent,
                                  int ID,
                                  int Flags);
```

参 数	含 意
<a href="#">x0</a>	复选框最左边的像素（在桌面坐标）。

<code>y0</code>	复选框最顶端的像素（在桌面坐标）。
<code>xsize</code>	复选框水平方向大小（以像素为单位）。
<code>ysize</code>	复选框垂直方向水平大小（以像素为单位）。
<code>hParent</code>	父窗口的句柄。
<code>ID</code>	返回的 ID。
<code>Flags</code>	窗口创建标识。具有代表性的是 <code>WM_CF_SHOW</code> ，表示控件立即可见（请参阅第 12 章“视窗管理器”中的 <code>WM_CreateWindow()</code> 函数列出的可用的参数值）。

**返回数值**

建立的复选框的句柄，如果函数执行失败，则返回 0。

**CHECKBOX\_CreateIndirect()**

函数原型的说明在本章开始部分。未使用当作参数传递的资源表的元素 `Flags` 和 `Para`。

**CHECKBOX\_IsChecked()****描述**

返回一个指定的复选框控件当前的状态（选中或未选中）。

**函数原型**

```
int CHECKBOX_IsChecked(CHECKBOX_Handle hObj);
```

参 数	含 意
<code>hObj</code>	复选框的句柄。

**返回数值**

0：未选中；1：选中

**CHECKBOX\_Uncheck()****描述**

设置一个指定的复选框状态为未选中。

## 函数原型

```
void CHECKBOX_Uncheck(CHECKBOX_Handle hObj);
```



参 数	含 意
<code>hObj</code>	复选框的句柄。

## 附加信息


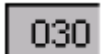
这是复选框的默认设置。

## 13.5 EDIT：文本编辑框控件

编辑区通常用作文本输入主要的用户接口：

空白编辑区	用户输入的编辑区
	

你也可能用编辑区输入二进制，十进制，或十六进制数值。一个十进制模式的编辑区可能与下表显示的相似。类似与复选框，编辑区如果被禁止的话，它会以灰色显示：

用户输入的编辑区（十进制）	禁止的编辑区
	

所有与编辑框相关的函数位于文件 `EDIT*.c`，`EDIT.h` 当中。所有标识符都是前缀 `EDIT`。

## 配置选项

类型	宏	默认值	说明
S	<code>EDIT_ALIGN_DEFAULT</code>	<code>GUI_TA_RIGHT</code>   <code>GUI_TA_VCENTER</code>	编辑区域文本对齐方式
N	<code>EDIT_BKCOLOR0_DEFAULT</code>	<code>0xc0c0c0</code>	禁止状态的背景颜色。
N	<code>EDIT_BKCOLOR1_DEFAULT</code>	<code>GUI_WHITE</code>	允许状态的背景颜色。
N	<code>EDIT_BORDER_DEFAULT</code>	1	边框线宽（以像素为单位）
S	<code>EDIT_FONT_DEFAULT</code>	<code>&amp;GUI_Font13_1</code>	编辑区域文本的字体。
N	<code>EDIT_TEXTCOLOR0_DEFAULT</code>	<code>GUI_BLACK</code>	禁止状态文本颜色。
N	<code>EDIT_TEXTCOLOR1_DEFAULT</code>	<code>GUI_BLACK</code>	允许状态文本颜色。
N	<code>EDIT_XOFF</code>	2	文本相对编辑区左边框的偏移量

可用的对齐标志：

水平对齐：GUI\_TA\_LEFT，GUI\_TA\_RIGHT，GUI\_TA\_HCENTER。

垂直对齐：GUI\_TA\_TOP，GUI\_TA\_BOTTOM，GUI\_TA\_VCENTER。

## 编辑框 API 函数

下表按字母的顺序列出了  $\mu$ C/GUI 编辑框相关的函数。函数的详细描述在稍后的部分给出。

函 数	说 明
EDIT_AddKey()	键盘输入函数。
EDIT_Create()	创建编辑区。
EDIT_CreateIndirect()	从资源表项目中创建编辑区。
EDIT_GetDefaultFont()	返回默认字体。
EDIT_GetText()	获得用户输入。
EDIT_GetValue()	返回当前数值。
EDIT_SetBinMode()	启用二进制编辑模式。
EDIT_SetBkColor()	设置编辑区的背景颜色。
EDIT_SetDecMode()	启用十进制编辑模式。
EDIT_SetDefaultFont()	设置用于编辑区的默认字体。
EDIT_SetDefaultTextAlign()	设置编辑区默认的文本对齐方式。
EDIT_SetFont()	给文本选择字体。
EDIT_SetHexMode()	启用十六进制模式。
EDIT_SetMaxLen()	设置编辑区的最大字符数量。
EDIT_SetText()	设置文本。
EDIT_SetTextAlign()	设置编辑区文本的对齐方式。
EDIT_SetTextColor()	设置文本的颜色。
EDIT_SetValue()	设置当前数值。
GUI_EditBin()	在当前光标位置编辑一个二进制数值。
GUI_EditDec()	在当前光标位置编辑一个十进制数值。
GUI_EditHex()	在当前光标位置编辑一个十六进制数值。
GUI_EditString()	在当前光标位置编辑一个字符串。

### EDIT\_AddKey()

描述

向一个指定编辑区输入内容。

### 函数原型

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

参 数	含 意
<code>hObj</code>	编辑区的句柄。
<code>Key</code>	输入的字符。

### 附加信息

向文本编辑框控件的用户输入区加入指定的字符。如果需要擦除最后一次显示的字符，你必须调用 `GUI_ID_BACKSPACE`。如果输入字符的数量已经到达最大值，其它字符将不会被加入。

## EDIT\_Create()

### 描述

在一个指定位置，以一个指定的大小创建一个文本编辑框控件。

### 函数原型

```
EDIT_Handle EDIT_Create(int x0, int y0, int xsize, int ysize, int ID, int MaxLen, int Flags);
```

参 数	含 意
<code>x0</code>	编辑区最左边像素（桌面座标）。
<code>y0</code>	编辑区最顶部像素（桌面座标）。
<code>xsize</code>	编辑框水平方向尺寸（以像素为单位）。
<code>ysize</code>	编辑框垂直方向尺寸（以像素为单位）。
<code>ID</code>	返回的 ID。
<code>MaxLen</code>	字符的最大数量。
<code>Flags</code>	窗口创建标识。具有代表性的是 <code>WM_CF_SHOW</code> ，表示控件立即可见（请参阅第 12 章“视窗管理器”中的 <code>WM_CreateWindow()</code> 函数列出的可用的参数值）。

### 返回数值

建立的文本编辑框控件的句柄，如果函数执行失败，则返回 0。

### EDIT\_CreateIndirect()

原型的说明在本章开始部分。未使用资源表中以参数形式传递的元素 Flags，但是编辑区允许的最大字符数量指定为元素 Para。

### EDIT\_GetDefaultFont()

#### 描述

设置用于文本编辑框控件的默认字体。

#### 函数原型

```
const GUI_FONT* EDIT_GetDefaultFont(void);
```

#### 返回数值

用于文本编辑框控件的默认字体。

### EDIT\_GetText()

#### 描述

获得一个指定的编辑区的用户输入内容。

#### 函数原型

```
void EDIT_GetText(EDIT_Handle hObj, char* sDest, int MaxLen);
```

参 数	含 意
<a href="#">hObj</a>	编辑区的句柄。
<a href="#">sDest</a>	缓冲区的指针。
<a href="#">MaxLen</a>	缓冲区的大小。

### EDIT\_GetValue()

#### 描述

返回编辑区当前的数值。当前数值只有在编辑区是二进制、十进制或十六制模式的情况



下才有用。

### 函数原型

```
I32 EDIT_GetValue(EDIT_Handle hObj);
```

参 数	含 意
<code>hObj</code>	编辑区的句柄。

### 返回数值

当前数值。

## EDIT\_SetBinMode()

### 描述

启用编辑区的二进制编辑模式。给出的数值可以在给出的范围内修改。

### 函数原型

```
void EDIT_SetBinMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

参 数	含 意
<code>hObj</code>	编辑区的句柄。
<code>Value</code>	修改的数值。
<code>Min</code>	最小数值。
<code>Max</code>	最大数值。

## EDIT\_SetBkColor()

### 描述

设置编辑区颜色。

### 函数原型

```
void EDIT_SetBkColor(EDIT_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	编辑区的句柄。

Index	必须设为 0，为将来的使用而保留。
Color	设置的顏色。

## EDIT\_SetDecMode()

### 描述

启用编辑区的十进制编辑模式。给出的数值可以在给出的范围内修改。

### 函数原型

```
void EDIT_SetDecMode(EDIT_Handle hEdit, I32 Value, I32 Min, I32 Max, int Shift,
U8 Flags);
```

参 数	含 意
hObj	编辑区的句柄。
Value	修改的数值。
Min	最小值。
Max	最大值。
Shift	如果大于 0，它指定小数点的位置。
Flags	参阅下表。

参数 `Flags` 允许的数值（以“OR”组合）

0（默认）	在正常模式编辑，当数值为负数时符号才显示。
GUI_EDIT_SIGNED	显示“+”和“-”符号。

## EDIT\_SetDefaultFont()

### 描述

设置用于编辑区的默认字体。

### 函数原型

```
void EDIT_SetDefaultFont(const GUI_FONT* pFont);
```

参 数	含 意
pFont	置为默认值的字体的指针。

**EDIT\_SetDefaultTextAlign()****描述**

设置编辑区默认的文本对齐方式。

**函数原型**

```
void EDIT_SetDefaultTextAlign(int Align);
```

参 数	含 意
<a href="#">Align</a>	默认的文本对齐方式 (参阅 GUI_SetTextAlign())。

**EDIT\_SetFont()****描述**

设置编辑区字体。

**函数原型**

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT* pfont);
```

参 数	含 意
<a href="#">hObj</a>	编辑区的句柄。
<a href="#">pFont</a>	字体的指针。

**EDIT\_SetHexMode()****描述**

启用编辑区的十六进制编辑模式。给出的数值可以在给出的范围内修改。

**函数原型**

```
void EDIT_SetHexMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

参 数	含 意
<a href="#">hObj</a>	编辑区的句柄。
<a href="#">Value</a>	修改的数值。
<a href="#">Min</a>	最小值。

Max	最大值。
-----	------

**EDIT\_SetMaxLen()****描述**

设置绘出的编辑区能编辑的字符的最大数量。

**函数原型**

```
void EDIT_SetMaxLen(EDIT_Handle hObj, int MaxLen);
```

参 数	含 意
hObj	编辑区的句柄。
MaxLen	最大的字符数量。

**EDIT\_SetText()****描述**

设置在编辑区中显示的文本。

**函数原型**

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

参 数	含 意
hObj	编辑区的句柄。
s	显示的文本。

**EDIT\_SetTextAlign()****描述**

设置编辑区的文本对齐方式。

**函数原型**

```
void EDIT_SetTextAlign(EDIT_Handle hObj, int Align);
```

参 数	含 意
hObj	编辑区的句柄。

[Align](#)

默认的文本对齐方式 (参阅 GUI\_SetTextAlign())。

**EDIT\_SetTextColor()****描述**

设置编辑区文本颜色。

**函数原型**

```
void EDIT_SetBkColor(EDIT_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<a href="#">hObj</a>	编辑区的句柄。
<a href="#">Index</a>	必须设为 0, 保留为将来使用。
<a href="#">Color</a>	设置的顏色。

**EDIT\_SetValue()****描述**

设置编辑区当前的数值。只有在设置了二进制, 十进制或十六进制编辑模式的情况下才有用。

**函数原型**

```
void EDIT_SetValue(EDIT_Handle hObj, I32 Value);
```

参 数	含 意
<a href="#">hObj</a>	编辑区的句柄。
<a href="#">Value</a>	新的数值。

**GUI\_EditBin()****描述**

在当前光标位置编辑一个二进制数。

**函数原型**

```
U32 GUI_EditBin(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

参 数	含 意
Value	修改的数值。
Min	最小值。
Max	最大值。
Len	编辑的数字的数量。
xsize	编辑区 X 轴的尺寸（以像素为单位）。

### 返回数值

如果<ENTER>键按下，返回新的数值。如果<ESC>键按下，返回旧的数值。

### 附加信息

在<ENTER> 或 <ESC>键按下后，函数返回。给出文本的内容只有在<ENTER>键按下后才修改。

## GUI\_EditDec()

### 描述

在当前光标位置编辑一个十进制数。

### 函数原型

```
U32 GUI_EditDec ( I32 Value,
                  I32 Min, I32 Max,
                  int Len, int xsize, int Shift,
                  U8 Flags);
```

参 数	含 意
Value	修改的数值。
Min	最小值。
Max	最大值。
Len	编辑的数字的数量。
xsize	编辑区 X 轴的尺寸（以像素为单位）。
Shift	如果>0，它指定十进制数小数点的位置。

### 返回数值

如果<ENTER>键按下，返回新的数值。如果<ESC>键按下，返回旧的数值。

**附加信息**

在<ENTER> 或 <ESC>键按下后，函数返回。给出文本的内容只有在<ENTER>键按下后才修改。

**GUI\_EditHex()****描述**

在当前光标位置编辑一个十六进制数。

**函数原型**

```
U32 GUI_EditHex(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

参 数	含 意
Value	修改的数值。
Min	最小值。
Max	最大值。
Len	编辑的数字的数量。
xsize	编辑区 X 轴的尺寸（以像素为单位）。

**返回数值**

如果<ENTER>键按下，返回新的数值。如果<ESC>键按下，返回旧的数值。

**附加信息**

在<ENTER> 或 <ESC>键按下后，函数返回。给出文本的内容只有在<ENTER>键按下后才修改。

**GUI\_EditString()****描述**

在当前光标位置编辑一个字符串。

**函数原型**

```
void GUI_EditString(char * pString, int Len, int xsize);
```

参 数	含 意
<code>pString</code>	要编辑字符串的指针。
<code>Len</code>	字符的最大数量。
<code>xsize</code>	编辑区 X 轴的尺寸（以像素为单位）。

### 附加信息

在<ENTER> 或 <ESC>键按下后，函数返回。给出文本的内容只有在<ENTER>键按下后才修改。

### 范例

下面范例展示了编辑框控件的使用。范例文件是随μC/GUI 一起发布的范例当中的：

```

/*-----
文件:      WIDGET_Edit.c
目的:      展示文本编辑框控件使用的例子
-----*/

#include "gui.h" #include "edit.h"

/*****
*          编辑一个字符串直到 ESC 键或 ENTER 键按下          *
*****/

static int Edit(void)
{
    int Key;
    EDIT_Handle hEdit;
    char aBuffer[28] ;
    GUI_SetFont (&GUI_Font8x16);
    GUI_DispStringHCenterAt("Use keyboard to modify string...", 160, 0);
    /* 创建编辑框控件 */
    hEdit = EDIT_Create( 50, 20, 219, 25, ' ', sizeof(aBuffer), 0 );
    /* 修改编辑框控件 */
    EDIT_SetText(hEdit, "Press <ENTER> when done...");
    EDIT_SetFont(hEdit, &GUI_Font8x16);
    EDIT_SetTextColor(hEdit, 0, GUI_RED);
    /* 操作键盘直到 ESC 或 ENTER 键被按下 */

```



```

do
{
    Key = GUI_WaitKey();
    switch(Key)
    {
        case GUI_ID_ESCAPE:
        case GUI_ID_CANCEL:
            break;
        default:
            EDIT_AddKey(hEdit, Key);
    }
} while((Key != GUI_ID_ESCAPE) &&(Key != GUI_ID_ENTER) &&(Key != 0)); /* 从
编辑框控件取出结果 */
if(Key == GUI_ID_ENTER)
    EDIT_GetText(hEdit, aBuffer, sizeof(aBuffer));
else
    aBuffer[0] = 0;
EDIT_Delete(hEdit);
GUI_DispStringHCenterAt(aBuffer, 160, 50);
return Key;
}

/*****
*                               主函数                               *
*****/

void main(void)
{
    GUI_Init();
    Edit();
    while(1)
        GUI_Delay(100);
}

```

上面范例执行结果的屏幕截图



## 13.6 FRAMEWIN：框架窗口控件

框架窗口给你的应用程序一个 PC 的应用程序——窗口外形。它们由一个环绕的框架、一个标题栏和一人用户区组成。标题栏的颜色改变展示窗口是活动的还是非活动的，如下所示：

所有与框架窗口相关的函数位于文件 FRAMEWIN\*.c 和 FRAMEWIN.h 当中。所有的标识符是前缀 FRAMEWIN。

### 配置选项

类型	宏	默认值	说明
N	FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT	0xff0000	活动状态标题栏颜色。
N	FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT	0x404040	非活动状态标题栏颜色。
N	FRAMEWIN_BORDER_DEFAULT	3	外部边界宽度（以像素为单位）。
N	FRAMEWIN_CLIENTCOLOR_DEFAULT	0xc0c0c0	窗口客户区颜色。
S	FRAMEWIN_DEFAULT_FONT	&GUI_Font8_1	标题栏文字使用的字体。
N	FRAMEWIN_FRAMECOLOR_DEFAULT	0xaaaaaa	框架颜色。
N	FRAMEWIN_IBORDER_DEFAULT	1	内部边界宽度（以像素为单位）。
B	FRAMEWIN_USE_3D	1	

### 框架窗口API函数

下表按字母顺序列出了 $\mu$ C/GUI 有效的与框架窗口相关的函数。函数的详细描述在后面部分给出。

函数	说明
FRAMEWIN_Create()	创建框架窗口。
FRAMEWIN_CreateAsChild()	创建一个作为一个子窗口的框架窗口控件。
FRAMEWIN_CreateIndirect()	从资源表条目中创建一个框架窗口。
FRAMEWIN_GetDefaultBorderSize()	返回默认边框尺寸。
FRAMEWIN_GetDefaultCaptionSize()	返回标题栏的默认尺寸。
FRAMEWIN_GetDefaultFont()	返回用于框架窗口标题的默认字体。
FRAMEWIN_SetActive()	设置框架窗口的状态。
FRAMEWIN_SetBarColor()	设置标题栏的背景颜色。

FRAMEWIN_SetClientColor()	设置客户区的背景。
FRAMEWIN_SetDefaultBarColor()	设置标题栏默认颜色。
FRAMEWIN_SetDefaultBorderSize()	设置默认边框尺寸。
FRAMEWIN_SetDefaultCaptionSize()	设置标题栏的默认高度。
FRAMEWIN_SetDefaultFont()	设置标题栏的默认字体。
FRAMEWIN_SetFont()	为标题文本选择字体。
FRAMEWIN_SetText()	设置标题文本。
FRAMEWIN_SetTextAlign()	设置标题文本的对齐方式。
FRAMEWIN_SetTextColor()	设置标题文本的颜色。
FRAMEWIN_SetTextPos()	设置标题文本的位置。

## FRAMEWIN\_Create()

### 描述

在一个指定位置以指定的尺寸创建一个框架窗口控件。

### 函数原型

```
FRAMEWIN_Handle FRAMEWIN_Create(    const char* pTitle,
                                     WM_CALLBACK* cb,
                                     int Flags,
                                     int x0, int y0,
                                     int xsize, int ysize);
```

参 数	含 意
pTitle	标题栏中显示的标题。
cb	为将来的应用保留。
Flags	窗口创建标识。具有代表性的是 WM_CF_SHOW，表示控件立即可见（请参阅第 12 章“视窗管理器”中的 WM_CreateWindow() 函数列出的可用的参数值）。
x0	框架窗口的 X 轴坐标。
y0	框架窗口的 Y 轴坐标。
xsize	框架窗口的 X 轴尺寸。
ysize	框架窗口的 Y 轴尺寸。

### 返回数值

所创建的框架窗口控件的句柄；如果函数执行失败则返回 0。

**FRAMEWIN\_CreateAsChild()****描述**

创建一个作为一个子窗口的框架窗口控件。

**函数原型**

```
FRAMEWIN_Handle FRAMEWIN_CreateAsChild (    int x0, int y0,
                                             int xsize, int ysize,
                                             WM_HWIN hParent,
                                             const char* pText,
                                             WM_CALLBACK* cb,
                                             int Flags);
```

参 数	含 意
x0	框架窗口的 X 轴坐标。
y0	框架窗口的 Y 轴坐标。
xsize	框架窗口的 X 轴尺寸。
ysize	框架窗口的 Y 轴尺寸。
hParent	父窗口的句柄。
pTitle	标题栏中显示的文本。
cb	为将来的应用保留。

**返回数值**

所创建的框架窗口控件的句柄；如果函数执行失败则返回 0。

**FRAMEWIN\_CreateIndirect()**

函数原型的说明在本章开始部分。下面的标志当作作为参数传递的资源的标志元素使用。

**允许的间接创建标志**

FRAMEWIN\_CF\_MOVEABLE 使框架窗口可移动（默认情况是固定的）。在资源表中未使用 Para 元素。

## **FRAMEWIN\_GetDefaultBorderSize()**

### **描述**

返回框架窗口边框的默认尺寸。

### **函数原型**

```
int FRAMEWIN_GetDefaultBorderSize(void);
```

### **返回数值**

一个框架窗口边框的默认尺寸。

## **FRAMEWIN\_GetDefaultCaptionSize()**

### **描述**

返回框架窗口标题栏的默认高度。

### **函数原型**

```
int FRAMEWIN_GetDefaultCaptionSize(void);
```

### **返回数值**

标题栏默认的高度。

## **FRAMEWIN\_GetDefaultFont()**

### **描述**

返回用于框架窗口标题的默认字体。

### **函数原型**

```
const GUI_FONT* FRAMEWIN_GetDefaultFont(void);
```

### **返回数值**

用于框架窗口标题的默认字体。

## FRAMEWIN\_SetActive()

### 描述

设置框架窗口的状态。标题栏的颜色根据状态改变。

### 函数原型

```
void FRAMEWIN_SetActive(FRAMEWIN_Handle hObj, int State);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>State</code>	框架窗口的状态（参阅下表）。

参数 `State` 允许的数值

0	框架窗口是非活动的。
1	框架窗口是活动的。

## FRAMEWIN\_SetBarColor()

### 描述

设置标题栏的背景颜色。

### 函数原型

```
void FRAMEWIN_SetBarColor(FRAMEWIN_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>Index</code>	必须设为 0，为将来的使用而保留。
<code>Color</code>	作为标题栏背景颜色使用的颜色。

## FRAMEWIN\_SetClientColor()

### 描述

设置客户区的颜色。

### 函数原型

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>Color</code>	设置的颜色。

## FRAMEWIN\_SetDefaultBarColor()

### 描述

设置标题栏的颜色。

### 函数原型

```
void FRAMEWIN_SetDefaultBarColor(int Index, GUI_COLOR Color);
```

参 数	含 意
<code>Index</code>	颜色索引（参阅下表）。
<code>Color</code>	设置的颜色。

参数 `Index` 允许的数值

0	设置框架窗口处于非活动状态时，使用的颜色。
1	设置框架窗口处于活动状态时，使用的颜色。

## FRAMEWIN\_SetDefaultBorderSize()

### 描述

设置框架窗口边框默认尺寸。

### 函数原型

```
void FRAMEWIN_SetDefaultBorderSize(int BorderSize);
```

参 数	含 意
<code>BorderSize</code>	设置的尺寸。

**FRAMEWIN\_SetDefaultCaptionSize()****描述**

设置标题栏 Y 轴尺寸。

**函数原型**

```
void FRAMEWIN_SetDefaultCaptionSize(int CaptionSize);
```

参 数	含 意
CaptionSize	设置的尺寸。

**FRAMEWIN\_SetDefaultFont()****描述**

设置用于显示标题的默认字体。

**函数原型**

```
void FRAMEWIN_SetDefaultFont(const GUI_FONT* pFont);
```

参 数	含 意
pFont	用作默认字体的字体的指针。

**FRAMEWIN\_SetFont()****描述**

设置标题字体。

**函数原型**

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT* pfont);
```

参 数	含 意
hObj	框架窗口的句柄。
pFont	字体的指针。



## FRAMEWIN\_SetText()

### 描述

设置标题文本。

### 函数原型

```
void FRAMEWIN_SetText(FRAMEWIN_Handle hObj, const char* s);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>s</code>	作为标题显示的文字。

## FRAMEWIN\_SetTextAlign()

### 描述

设置标题文本的对齐方式。

### 函数原型

```
void FRAMEWIN_SetTextAlign(FRAMEWIN_Handle hObj, int Align);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>Align</code>	标题栏对齐属性（参阅下表）。

参数 Align 允许的值

<code>GUI_TA_HCENTER</code>	标题居中（默认）。
<code>GUI_TA_LEFT</code>	标题左对齐。
<code>GUI_TA_RIGHT</code>	标题右对齐。

### 附加信息

如果该函数没有调用，默认行为是显示居中的文本。

**FRAMEWIN\_SetTextColor()****描述**

设置标题文本的颜色。

**函数原型**

```
void FRAMEWIN_SetTextColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>Color</code>	作为标题文字颜色使用的颜色。

**FRAMEWIN\_SetTextPos()****描述**

设置标题文本的位置。

**函数原型**

```
void FRAMEWIN_SetTextPos(FRAMEWIN_Handle hObj, int XOff, int YOff);
```

参 数	含 意
<code>hObj</code>	框架窗口的句柄。
<code>XOff</code>	标题文字的 X 轴坐标。
<code>YOff</code>	标题文字的 Y 轴坐标。

**范例**

下面的范例展示如何使用框架控件。首选创建控件，修改几个它的属性，然后创建一个客户窗口。你不能使用框架控件本身显示任何东西，必须是始终要创建一个客户窗口。该范例文件是 `samples` 目录下的 `WIDGET_FrameWin.c`：

```

/*-----
文件：      WIDGET_FrameWin.c
目的：      展示一个框架窗口控件使用的例子
-----*/

#include "gui.H"

```

```

#include "framewin.h"
#include <stddef.h>

/*****
*                               客户窗口的回调函数                               *
*****/

static WM_RESULT CallbackChild(WM_MESSAGE * pMsg)
{
    WM_HWIN hWin =(FRAMEWIN_Handle) (pMsg->hWin);
    switch(pMsg->MsgId)
    {
        case WM_PAINT:
            /* Handle the paint message */
            GUI_SetBkColor(GUI_BLUE);
            GUI_SetColor(GUI_WHITE);
            GUI_SetFont(&GUI_FontComic24B_ASCII);
            GUI_SetTextAlign(GUI_TA_HCENTER | GUI_TA_VCENTER);
            GUI_Clear();
            GUI_DispStringHCenterAt(    "Child window",
                                       WM_GetWindowSizeX(hWin) / 2,
                                       WM_GetWindowSizeY(hWin) / 2);
            break;
    }
}

/*****
*                               创建框架及子窗口                               *
*****/

static void DemoFramewin(void)
{
    /* 创建框架窗口 */
    FRAMEWIN_Handle hFrame = FRAMEWIN_Create(    "Frame window",
                                                NULL, WM_CF_SHOW,
                                                10, 10, 150, 60);

    FRAMEWIN_SetFont(hFrame, &GUI_Font16B_ASCII);
    FRAMEWIN_SetTextColor(hFrame, GUI_RED);

```

```

FRAMEWIN_SetBarColor(hFrame, 0, GUI_GREEN);
FRAMEWIN_SetTextAlign(hFrame, GUI_TA_HCENTER);
/* 创建客户窗口 */
WM_CreateWindowAsChild(0, 0, 0, 0, hFrame, WM_CF_SHOW, CallbackChild, 0);
}

/*****
*                               *
*****/

void main(void)
{
    GUI_Init();
    DemoFramewin();
    while(1)
        GUI_Delay(100);
}



```

上面范例执行结果的屏幕截图



### 13.7 LISTBOX: 列表框控件

列表框用于在一个列表中选择元素。一个列表框可以以一个无边框的窗口的形式创建，如下所示，或者作为一个框架窗口控件的子窗口（参阅本部分末的附加屏幕截图）。当列表框的项目被选中，它们会以高亮显示。注意所选择项目的背景颜色依赖于列表框窗口是否有输入焦点。

带焦点的列表框	不带焦点的列表框
	

所有与列表框相关的函数位于文件 LISTBOX\*.c, LISTBOX.h 当中。所有的标识符是前缀名 LISTBOX。

## 配置选项

类型	宏	默认值	说明
N	LISTBOX_BKCOLOR0_DEFAULT	GUI_WHITE	非选中状态的背景颜色。
N	LISTBOX_BKCOLOR1_DEFAULT	GUI_GRAY	没有焦点的选中状态的背景颜色。
N	LISTBOX_BKCOLOR2_DEFAULT	GUI_WHITE	有焦点的选中状态的背景颜色。
S	LISTBOX_FONT_DEFAULT	&GUI_Font13_1	使用的字体。
N	LISTBOX_TEXTCOLOR0_DEFAU	GUI_BLACK	非选中状态的文本颜色。
N	LISTBOX_TEXTCOLOR1_DEFAU	GUI_BLACK	没有焦点的选中状态的文本颜色。
N	LISTBOX_TEXTCOLOR2_DEFAU	GUI_BLACK	有焦点的选中状态的文本颜色。
B	LISTBOX_USE_3D	1	启用3D支持。

## 列表框API函数

下表根据字母的顺序列出了μC/GUI 与列表框有关的函数。函数详细的描述随后给出。

函数	说明
<a href="#">LISTBOX_Create()</a>	创建列表框。
<a href="#">LISTBOX_CreateAsChild()</a>	以子窗口的形式追寻列表框。
<a href="#">LISTBOX_CreateIndirect()</a>	从资源表条目追寻列表框。
<a href="#">LISTBOX_DecSel()</a>	减小选择范围。
<a href="#">LISTBOX_GetDefaultFont()</a>	返回列表框的默认字体。
<a href="#">LISTBOX_GetSel()</a>	返回所选择行的数目。
<a href="#">LISTBOX_IncSel()</a>	增加选择范围。
<a href="#">LISTBOX_SetBackColor()</a>	设置背景颜色。
<a href="#">LISTBOX_SetDefaultFont()</a>	修改列表框控件的默认字体。
<a href="#">LISTBOX_SetFont()</a>	选择字体。
<a href="#">LISTBOX_SetSel()</a>	设置选择的行。
<a href="#">LISTBOX_SetTextColor()</a>	设置前景颜色。

## LISTBOX\_Create()

### 描述

在指定位置，以指定的尺寸创建一个列表框控件。

## 函数原型

```
LISTBOX_Handle LISTBOX_Create( const GUI_ConstString* ppText,
                               int x0, int y0,
                               int xSize, int ySize,
                               int Flags);
```

参 数	含 意
<code>ppText</code>	包含有要显示的元素的一批字符串指针的指针。
<code>x0</code>	列表框 X 轴坐标。
<code>y0</code>	列表框 Y 轴坐标。
<code>xSize</code>	列表框 X 轴尺寸。
<code>ySize</code>	列表框 Y 轴尺寸。
<code>Flags</code>	窗口创建标志。典型的是 <code>WM_CF_SHOW</code> , 它使控件能立即可见 (请参考第 12 章“视窗管理器”中的 <code>WM_CreateWindow()</code> , 其中有一列有效的参数值。)

## 返回数值

所创建的列表框控件的句柄, 如果失败则为 0。

## 附加信息

如果参数 `ySize` 大于控件内容绘制所需要的间隔, 则 Y 轴尺寸将被减小到要求的数值。对于参数 `Xsize` 也是同一个道理。

## LISTBOX\_CreateAsChild()

### 描述

以一个子窗口的形式创建列表框控件。

### 函数原型

```
LISTBOX_Handle LISTBOX_CreateAsChild ( const GUI_ConstString* ppText,
                                       HBWIN hWinParent,
                                       int x0, int y0,
                                       int xSize, int ySize,
                                       int Flags);
```

参 数	含 意
<code>ppText</code>	包含有要显示的元素的一批字符串指针的指针。
<code>hWinParent</code>	父窗口句柄。
<code>x0</code>	列表框相对于父窗口的 X 轴坐标。
<code>y0</code>	列表框相对于父窗口的 Y 轴坐标。
<code>xSize</code>	列表框 X 轴尺寸。
<code>ySize</code>	列表框 Y 轴尺寸。
<code>Flags</code>	窗口创建标志 (参阅 <code>LISTBOX_Create()</code> )。

### 返回数值

所创建列表框控件的句柄，如果函数执行失败，则返回 0。

### 附加信息

如果参数 `ySize` 大于控件内容绘制所需要的间隔，则 Y 轴尺寸将被减小到要求的数值。如果 `ySize = 0`，控件的 Y 轴尺寸将被设置为来自父窗口的客户区的 Y 轴尺寸。对于参数 `Xsize` 也是同一个道理。

### `LISTBOX_CreateIndirect()`

函数原型的说明在本章开始部分。资源表作为参数传递的元素 `Flags` 和 `Para` 未使用。

### `LISTBOX_DecSel()`

#### 描述

减小列表框选择范围 (把指定列表框的选择条向上移动一个元素)。

#### 函数原型 s

```
void LISTBOX_DecSel (LISTBOX_Handle hObj);
```

参 数	含 意
<code>hObj</code>	列表框的句柄。

### `LISTBOX_GetDefaultFont()`

#### 描述

返回用于创建列表框控件的默认字体。

### 函数原型

```
const GUI_FONT* LISTBOX_GetDefaultFont(void);
```

### 返回数值

默认字体的指针。

## LISTBOX\_GetSel()

### 描述

返回在一个指定列表框中当前选择的单元的数量。

### 函数原型

```
int LISTBOX_GetSel(LISTBOX_Handle hObj);
```

参 数	含 意
<a href="#">hObj</a>	列表框的句柄。

### 返回数值

当前选择的单元的数量。

## LISTBOX\_IncSel()

### 描述

增加列表框选择范围（把指定列表框的选择条向下移动一元素）。

### 函数原型

```
void LISTBOX_IncSel(LISTBOX_Handle hObj);
```

参 数	含 意
<a href="#">hObj</a>	列表框的句柄。



**LISTBOX\_SetBackColor()****描述**

设置列表框背景颜色。

**函数原型**

```
void LISTBOX_SetBackColor(LISTBOX_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<a href="#">hObj</a>	列表框的句柄。
<a href="#">Index</a>	背景颜色的索引（参阅下表）。
<a href="#">Color</a>	设置的颜色。

**参数 [Index](#) 允许的值**

0	为未选择的单元设置颜色。
1	为选择的单元设置颜色。

**LISTBOX\_SetDefaultFont()****描述**

设置用于列表框控件的默认字体。

**函数原型**

```
void LISTBOX_SetDefaultFont(const GUI_FONT* pFont)
```

参 数	含 意
<a href="#">pFont</a>	字体的指针。

**LISTBOX\_SetFont()****描述**

设置列表框字体。

**函数原型**

```
void LISTBOX_SetFont(LISTBOX_Handle hObj, const GUI_FONT* pfont);
```

参 数	含 意
<code>hObj</code>	列表框的句柄。
<code>pFont</code>	字体的指针。

### LISTBOX\_SetSel()

#### 描述

设置指定列表框选择的单元。

#### 函数原型

```
void LISTBOX_SetSel(LISTBOX_Handle hObj, int Sel);
```

参 数	含 意
<code>hObj</code>	列表框的句柄。
<code>Sel</code>	选择的单元。

### LISTBOX\_SetTextColor()

#### 描述

设置列表框文本颜色。

#### 函数原型

```
void LISTBOX_SetTextColor(LISTBOX_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<code>hObj</code>	列表框的句柄。
<code>Index</code>	文本颜色的索引（参考 LISTBOX_SetBackColor()）。
<code>Color</code>	设置的颜色。

## 范例

### 使用列表框控件

下面的范例展示了列表框控件的简单使用。其源代码文件是范例 WIDGET\_SimpleListBox.c:

```

/*-----
文件:      WIDGET_SimpleListBox.c
目的:      展示一个列表框控件使用的例子
-----*/

#include "gui.H" #include "listbox.h" #include <stddef.h>
const GUI_ConstString ListBox[] =
{
"English", "Deutsch", "Français", "Japanese", "Italiano", NULL
};
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

/******
*                      列表框使用的示范                      *
******/

void DemoListBox(void)
{
    int i;
    int Entries = countof(ListBox) - 1;
    LISTBOX_Handle hListBox;
    int ySize = GUI_GetFontSizeYOf(&GUI_Font13B_1) * Entries;
    /* 建立列表框 */
    hListBox = LISTBOX_Create(ListBox, 10, 10, 60, ySize, WM_CF_SHOW);
    /* 改变列表框当前的选择 */
    for(i = 0; i < Entries-1; i++)
    {
        GUI_Delay(500);
        LISTBOX_IncSel(hListBox);
        WM_Exec();
    }
    for(i = 0; i < Entries-1; i++)
    {
        GUI_Delay(500);
        LISTBOX_DecSel(hListBox);
        WM_Exec();
    }
}

```

```

/* 删除列表框控件 */
LISTBOX_Delete(hListBox);
GUI_Clear();
}

/*****
*                               *
*                               *
*****/

static void DemoListbox(void)
{
    GUI_SetBkColor(GUI_BLUE);
    GUI_Clear();
    while(1)
    {
        DemoListBox();
    }
}

/*****
*                               *
*                               *
*****/

void main(void)
{
    GUI_Init();
    DemoListbox();
}

```

上面范例执行结果的屏幕截图



使用一个带边框或无边框的列表框控件

下面的范例展示了列表框控件在有框架窗口和没有框架窗口的情况下的使用。其源代码文件是 WIDGET\_ListBox.c:

```

/*-----
文件:      WIDGET_ListBox.c
目的:      展示一个列表框控件使用的例子
-----*/

#include "gui.H"
#include "framewin.h" #include "listbox.h" #include <stddef.h>
const GUI_ConstString ListBox[] =
{
    "English", "Deutsch", "Français", "Japanese", "Italiano", NULL
};
#define countof(Array) (sizeof(Array) / sizeof(Array[0] ) )

/*-----
*                          单独使用列表框控件                          *
*-----*/

void DemoListBox(void)
{
    int i;
    int Entries = countof(ListBox) - 1;
    LISTBOX_Handle hListBox;
    int ySize = GUI_GetFontSizeYOf(&GUI_Font13B_1) * Entries;
    /* 创建列表框 */
    hListBox = LISTBOX_Create(ListBox, 10, 10, 120, ySize, WM_CF_SHOW);
    /* 修改列表框属性 */
    LISTBOX_SetFont(hListBox, &GUI_Font13B_1);
    LISTBOX_SetBackColor(hListBox, 0, GUI_BLUE);
    LISTBOX_SetBackColor(hListBox, 1, GUI_LIGHTBLUE);
    LISTBOX_SetTextColor(hListBox, 0, GUI_WHITE);
    LISTBOX_SetTextColor(hListBox, 1, GUI_BLACK);
    /* 改变列表框当前的选择 */
    for(i = 0; i < Entries - 1; i++)
    {
        GUI_Delay(500);
    }
}

```

```

        LISTBOX_IncSel(hListBox);
    }
    for(i = 0; i < Entries - 1; i++)
    {
        GUI_Delay(500);
        LISTBOX_DecSel(hListBox);
    }
    GUI_Delay(500);
    /* 删除列表框控件 */
    LISTBOX_Delete(hListBox);
    GUI_Clear();
}

/*****
*           把列表框控件当作一个框架窗口控件的子窗口使用           *
*****/

void DemoListBoxAsChild(void)
{
    int i;
    int Entries = countof(ListBox) - 1;
    FRAMEWIN_Handle hFrame;
    LISTBOX_Handle hListBox;
    int ySize = GUI_GetFontSizeYOf(&GUI_Font13B_1)
                * Entries + GUI_GetFontSizeYOf(&GUI_Font16B_ASCII) + 7;
    /* 创建框架窗口 */
    hFrame = FRAMEWIN_Create("List box", NULL, WM_CF_SHOW, 10, 10, 120, ySize);
    /* 修改框架窗口属性 */
    FRAMEWIN_SetFont(hFrame, &GUI_Font16B_ASCII);
    FRAMEWIN_SetTextColor(hFrame, GUI_RED);
    FRAMEWIN_SetBarColor(hFrame, 0, GUI_GREEN);
    FRAMEWIN_SetTextAlign(hFrame, GUI_TA_HCENTER);
    /* 创建列表框 */
    hListBox = LISTBOX_CreateAsChild(ListBox, hFrame, 0, 0, 0, 0, WM_CF_SHOW);
    /* 修改列表框属性 */
    LISTBOX_SetBackColor(hListBox, 0, GUI_BLUE);
    LISTBOX_SetBackColor(hListBox, 1, GUI_LIGHTBLUE);
}

```

```

LISTBOX_SetTextColor(hListBox, 0, GUI_WHITE);
LISTBOX_SetTextColor(hListBox, 1, GUI_BLACK);
LISTBOX_SetFont(hListBox, &GUI_Font13B_1);
/* 改变列表框当前的选择 */
for(i = 0; i < Entries - 1; i++)
{
    GUI_Delay(500);
    LISTBOX_IncSel(hListBox);
}
for(i = 0; i < Entries - 1; i++)
{
    GUI_Delay(500);
    LISTBOX_DecSel(hListBox);
} GUI_Delay(500);
/* 删除列表框控件 */
LISTBOX_Delete(hListBox);
/* 删除框架窗口控件 */
FRAMEWIN_Delete(hFrame);
GUI_Clear();
}

/*****
*                               列表框控件示范                               *
*****/

static void DemoListbox(void)
{
    GUI_SetBkColor(GUI_GRAY);
    GUI_Clear();
    while(1)
    {
        DemoListBox();
        DemoListBoxAsChild();
    }
}

/*****

```

```

*                               主函数                               *
*****/

void main(void)
{
    GUI_Init();
    DemoListbox();
}

```

上面范例执行结果的屏幕截图



## 13.8 PROGBAR: 进度条控件

进度条通常在可视化应用中使用。例如，一个容器的填充容积指示器或一个油压指示器。例子的屏幕截图可以在本章开始部分和本部分末。所有与进度条相关的函数在文件 PROGBAR\*.c, PROGBAR.h 当中。所有的标识符是前缀 PROGBAR。

### 配置选项

类型	宏	默认值	说明
S	PROGBAR_DEFAULT_FONT	GUI_DEFAULT_FONT	使用的字体。
N	PROGBAR_DEFAULT_BARCOLOR0	0x555555 （深灰）	左侧条棒的颜色。
N	PROGBAR_DEFAULT_BARCOLOR1	0xA6A6A6 （浅灰）	右侧条棒的颜色。
N	PROGBAR_DEFAULT_TEXTCOLOR0	0xFFFFFFFF	左侧条棒文字的颜色。
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	右侧条棒文字的颜色。

### 进度条API 函数

下表按字母的顺序列出了  $\mu$ C/GUI 与进度条有关的函数。函数详细的描述在后面给出。



函数	说明
<code>PROGBAR_Create()</code>	创建进度条。
<code>PROGBAR_CreateIndirect()</code>	从资源表条目创建进度条。
<code>PROGBAR_SetBarColor()</code>	设置进度条的颜色。
<code>PROGBAR_SetFont()</code>	选择文字使用的字体。
<code>PROGBAR_SetMinMax()</code>	设置进度条的最小和最大数值。
<code>PROGBAR_SetText()</code>	设置（可选择）条棒图形的文字。bargraph.
<code>PROGBAR_SetTextAlign()</code>	设置对齐方式（默认为居中）。
<code>PROGBAR_SetTextColor()</code>	设置文字的颜色。
<code>PROGBAR_SetTextPos()</code>	设置坐标（默认为 0, 0）。
<code>PROGBAR_SetValue()</code>	设置条棒图形的数值（如果没有文字则显示百分比）。

## PROGBAR\_Create()

### 描述

在一个指定位置，以指定的尺寸追寻一个进度条。

### 函数原型

```
PROGBAR_Handle PROGBAR_Create(int x0, int y0, int xsize, int ysize, int Flags);
```

参 数	含 意
<code>x0</code>	进度条最左边像素（在桌面座标）。
<code>y0</code>	进度条最顶部的像素（在桌面座标）。
<code>xsize</code>	进度条水平方向的尺寸（以像素为单位）。
<code>ysize</code>	进度条垂直方向的尺寸（以像素为单位）。
<code>Flags</code>	窗口创建标志。典型的是 <code>WM_CF_SHOW</code> ，它使控件能立即可见（请参考第 12 章“视窗管理器”中的 <code>WM_CreateWindow()</code> ，其中有一列有效的参数值。）

### 返回数值

所创建的进度条控件的句柄；如果函数执行失败则为 0。

## PROGBAR\_CreateIndirect()

函数原型的说明在本章开始部分。作为参数传递的资源表的元素 `Flags` 和 `Para` 未使用到。

## PROGBAR\_SetBarColor()

### 描述

设置进度条的颜色。

### 函数原型

```
void PROGBAR_SetBarColor(PROGBAR_Handle hObj, int Index, GUI_COLOR Color);
```

参 数	含 意
<a href="#">hObj</a>	进度条的句柄。
<a href="#">Index</a>	参阅下表。其它数值是不允许的。
<a href="#">Color</a>	设置的颜色（24 位 RGB 数值）。

参数 [Index](#) 允许的值

0	进度条左侧部分。
1	进度条右侧部分。

## PROGBAR\_SetFont()

### 描述

选择进度条当中显示文本的字体。

### 函数原型

```
void PROGBAR_SetFont(PROGBAR_Handle hObj, const GUI_FONT* pFont);
```

参 数	含 意
<a href="#">hObj</a>	进度条的句柄。
<a href="#">pFont</a>	字体的指针。

### 附加信息

如果该函数没有被调用，将使用进度条默认字体（GUI 默认字体）。不过，进度默认字体可以在 GUIConf.h 文件中修改。

如下所示，简单地用“#define”定义默认字体：

```
#define PROGBAR_DEFAULT_FONT &GUI_Font13_ASCII
```

## PROGBAR\_SetMinMax()

### 描述

设置用于进度条的最小数值和最大数值。

### 函数原型

```
void PROGBAR_SetMinMax(PROGBAR_Handle hObj, int Min, int Max);
```

参 数	含 意
<code>hObj</code>	进度条的句柄。
<code>Min</code>	最小值, 范围: $-16383 < \text{Min} \leq 16383$ 。
<code>Max</code>	最大值, 范围: $-16383 < \text{Max} \leq 16383$ 。

### 附加信息

如果该函数未被调用, 将使用默认数值 `Min = 0`, `Max = 100`。

## PROGBAR\_SetText()

### 描述

设置进度条当中显示的文本。

### 函数原型

```
void PROGBAR_SetText(PROGBAR_Handle hObj, const char* s);
```

参 数	含 意
<code>hObj</code>	进度条的句柄。
<code>s</code>	显示的文本。允许一个空指针; 在这种情况下显示一个百分数。

### 附加信息

如果该函数未被调用, 一个百分比数值会作为默认值显示。如果你不打算显示任何文本, 你应当设置一个空字符串。

## PROGBAR\_SetTextAlign()

### 描述

设置文本对齐方式。

### 函数原型

```
void PROGBAR_SetTextAlign(PROGBAR_Handle hObj, int Align);
```

参 数	含 意
<a href="#">hObj</a>	进度条的句柄。
<a href="#">Align</a>	文本的水平对齐属性（参阅下表）。

参数 [Align](#) 允许的值

GUI_TA_HCENTER	标题对中（默认）。
GUI_TA_LEFT	在左侧显示标题。
GUI_TA_RIGHT	在右侧显示标题。

### 附加信息

如果该函数未被调用，默认的行为是文本对中显示。

## PROGBAR\_SetTextPos()

### 描述

以像素为单位设置文本的坐标。

### 函数原型

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

参 数	含 意
<a href="#">hObj</a>	进度条的句柄。
<a href="#">XOff</a>	水平方向文本移动的像素数。该值为正值则文本向右侧移动。
<a href="#">YOff</a>	垂直方向文本移动的像素数。该值为正值则文本向下移动。

### 附加信息

该数值在控件内移动文本到指定像素数处。通常，默认值 (0, 0) 应该足够了。

## PROGBAR\_SetValue()

### 描述

设置进度条的数值。

### 函数原型

```
void PROGBAR_SetValue(PROGBAR_Handle hObj, int v);
```

参 数	含 意
<code>hObj</code>	进度条的句柄。
<code>v</code>	设置的数值。

### 附加信息

进度条指示由最大/最小值的关系计算得来。如果自动显示一个百分数，该百分数同样可以由给出的最大/最小值算出，如下所示：

$$p = 100\% * (v - \text{Min}) / (\text{Max} - \text{Min})$$

控件创建后的默认数值为 0。

## 范例

### 使用进度条控件

下面的简单范例展示进度条控件的使用。其源代码文件是范例中的 WIDGET\_SimpleProgbar.c:

```

/*-----
文件:      WIDGET_SimpleProgbar.c
目的:      展示一个进度条控件使用的例子
-----*/

#include "gui.h"
#include "progbar.h"

```

```

/*****
*
*                               展示进度条的使用
*
*****/

static void DemoProgBar(void)
{
    int i;
    PROGBAR_Handle ahProgBar;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringAt("Progress bar", 100, 80);
    /* 创建进度条 */
    ahProgBar = PROGBAR_Create(100, 100, 100, 20, WM_CF_SHOW);
    GUI_Delay(500);
    /* 修改进度条 */
    for(i = 0; i <= 100; i++)
    {
        PROGBAR_SetValue(ahProgBar, i);
        GUI_Delay(10);
    }
    GUI_Delay(500);
    /* 删除进度条 */
    PROGBAR_Delete(ahProgBar);
}

/*****
*
*                               主函数
*
*****/

void main(void)
{
    GUI_Init();
    while(1)
    {
        DemoProgBar();
    }
}

```

上面范例执行结果的屏幕截图



### 进度条控件的高级应用

这个更高级的范例创建一个容器的填充级别指示器。其源代码文件是范例中的 WIDGET\_Progbar.c:

```

/*-----*/
文件:      WIDGET_Progbar.c
目的:      进度条控件使用的简单演示
-----*/

#include "gui.h"
#include "progbar.h"

/*****
*                  展示进度条的使用                  *
*****/

static void DemoProgBar(void)
{
    int i;
    PROGBAR_Handle ahProgBar[2] ;
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringAt("Progress bar", 100, 80);
    /* 创建全身 */
    ahProgBar[0] = PROGBAR_Create(100, 100, 100, 20, WM_CF_SHOW);
    ahProgBar[1] = PROGBAR_Create( 80, 150, 140, 10, WM_CF_SHOW);
    /* 使用存储设备 (可选, 为了看起来更好一些) */
    PROGBAR_EnableMemdev(ahProgBar[0]);
    PROGBAR_EnableMemdev(ahProgBar[1]);
}

```

```
GUI_Delay(1000);
PROGBAR_SetMinMax(ahProgBar[1], 0, 500);
while(1)
{
    PROGBAR_SetFont(ahProgBar[0], &GUI_Font8x16);
    #if(LCD_BITSPERPIXEL <= 4)
        PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_DARKGRAY);
        PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_LIGHTGRAY);
    #else
        PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_GREEN);
        PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_RED);
    #endif
    for(i=0; i<=100; i++)
    {
        PROGBAR_SetValue(ahProgBar[0], i);
        PROGBAR_SetValue(ahProgBar[1], i);
        GUI_Delay(5);
    }
    PROGBAR_SetText(ahProgBar[0], "Tank empty");
    for(; i>=0; i--)
    {
        PROGBAR_SetValue(ahProgBar[0], i);
        PROGBAR_SetValue(ahProgBar[1], 200-i);
        GUI_Delay(5);
    }
    PROGBAR_SetText(ahProgBar[0], "Any text ...");
    PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_LEFT);
    for(; i<=100; i++)
    {
        PROGBAR_SetValue(ahProgBar[0], i);
        PROGBAR_SetValue(ahProgBar[1], 200+i);
        GUI_Delay(5);
    }
    PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_RIGHT);
    for(; i>=0; i--)
    {
        PROGBAR_SetValue(ahProgBar[0], i);
```



```

        PROGBAR_SetValue(ahProgBar[1], 400-i); GUI_Delay(5);
    }
    PROGBAR_SetFont(ahProgBar[0], &GUI_FontComic18B_1);
    PROGBAR_SetText(ahProgBar[0], "Any font ...");
    for(; i<=100; i++)
    {
        PROGBAR_SetValue(ahProgBar[0], i);
        PROGBAR_SetValue(ahProgBar[1], 400+i);
        GUI_Delay(5);
    }
    GUI_Delay(1000);
}

/*****
*                               *
*                               *
*****/

void main(void)
{
    GUI_Init() ;
    DemoProgBar();
}

```

上面范例执行结果的屏幕截图



## 13.9 RADIO: 单选按钮控件

单选按钮，类似于复选框，用于选项的选择。当一个单选按钮开启或被选中后，在它上

面会显示一个点。与复选框不同的是在同一时间，用户只能选择一个单选按钮。当一个按钮被选择时，控件中其它的按钮被关闭，如下图所示。一个单选按钮控件可以包括几个按钮，它们总是垂直排列的。



与单选框相关的函数位于文件 RADIO\*.c，RADIO.h 中。所有的标识符是前缀 RADIO。

### 配置选项

类型	宏	默认值	说明
	RADIO_BKCOLOR0_DEFAULT	0xc0c0c0	非活动状态按钮的背景颜色。
	RADIO_BKCOLOR1_DEFAULT	GUI_WHITE	活动状态按钮的背景颜色。

### 单选按钮控件 API 函数

下表按字母顺序列出了  $\mu$ C/GUI 与单选按钮有关的函数。函数详细的描述在稍后给出。

函数	说明
<a href="#">RADIO_Create()</a>	建立一组单选按钮。
<a href="#">RADIO_CreateIndirect()</a>	从资源表条目创建一组单选按钮。
<a href="#">RADIO_Dec()</a>	将按钮选择序号减 1。
<a href="#">RADIO_GetValue()</a>	返回当前选择的按钮。
<a href="#">RADIO_Inc()</a>	将按钮选择序号加 1。
<a href="#">RADIO_SetValue()</a>	设置选择的按钮。

### RADIO\_Create()

#### 描述

在一个指定位置，以指定的大小创建一个单选按钮控件。

#### 函数原型

```
RADIO_Handle RADIO_Create ( int x0, int y0,
                             int xsize, int ysize,
                             WM_HWIN hParent,
```

```
int Id, int Flags, unsigned Para);
```

参 数	含 意
x0	单选按钮最左侧像素 (在桌面坐标)。
y0	单选按钮最顶部像素 (在桌面坐标)。
xsize	单选按钮水平尺寸 (以像素为单位)。
ysize	单选按钮垂直尺寸 (以像素为单位)。
hParent	父窗口句柄。
Id	返回的 ID 号。
Flags	窗口创建标志。典型的是 WM_CF_SHOW 用于使控件立即可见 (请参阅第 12 章“视窗管理器”中的 WM_CreateWindow() 所列出的参数值)。
Para	按钮在组当中的数值。

### 返回数值

所创建的单选按钮控件的句柄；如果函数执行失败则为 0。

## RADIO\_CreateIndirect()

函数原型的说明在本章的开始部分。资源表作为参数传递的元素 Flags 未使用，但是所使用的单选按钮号码指定为 Para 元素。

## RADIO\_Dec()

### 描述

把选择数值减 1。

### 函数原型

```
void RADIO_Dec(RADIO_Handle hObj);
```

参 数	含 意
hObj	单选按钮控件的的句柄。

### 附加信息

请注意，按钮的编号总是从顶部以 0 开始；因此，选择按钮的序号减 1 实际上将选择的按钮往上移一个。

## RADIO\_GetValue()

### 描述

返回当前选择的按钮。

### 函数原型

```
void RADIO_GetValue(RADIO_Handle hObj);
```

参 数	含 意
<code>hObj</code>	单选按钮控件的的句柄。

### 返回数值

当前选择按钮的值。

## RADIO\_Inc()

### 描述

选择数值加 1。

### 函数原型

```
void RADIO_Inc(RADIO_Handle hObj);
```

参 数	含 意
<code>hObj</code>	单选按钮控件的的句柄。

### 附加信息

请注意，按钮的编号总是从顶部以 0 开始；因此，选择按钮的序号加 1 实际上将选择的按钮往下移一个。

## RADIO\_SetValue()

### 描述

设置当前选择的按钮。

## 函数原型

```
void RADIO_SetValue(RADIO_Handle hObj, int v);
```

参 数	含 意
<code>hObj</code>	单选按钮控件的的句柄。
<code>v</code>	设置的数值。

## 附加信息

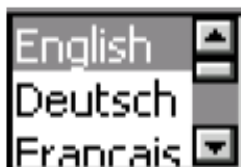
单选按钮控件最顶部的单选按钮的序号总是 0，往下一个按钮总是 1，再往下一个是 2，等等。

## 13.10 SCROLLBAR: 滚动条控件

滚动条用于滚动一个列表框或任何没有完全适合它们框架的窗口。它们可以以水平或垂直方式创建。



典型的，一个滚动条依附在一个存在的窗口，如下面展示列表框例子：



所有与滚动条相关的函数位于文件 `SCROLLBAR*.c`, `SCROLLBAR.h` 当中。所有的标识符是前缀名 `SCROLLBAR`。

## 配置选项

类型	宏	默认值	说明
N	<code>SCROLLBAR_BKCOLOR0_DEFA</code>	<code>0x808080</code>	背景（thumb 区域）颜色。
N	<code>SCROLLBAR_BKCOLOR1_DEFA</code>	<code>GUI_BLACK</code>	滚动条（thumb）颜色。
N	<code>SCROLLBAR_COLOR0_DEFAULT</code>	<code>0xc0c0c0</code>	箭头按钮的颜色。
B	<code>SCROLLBAR_USE_3D</code>	1	启用 3D 支持。

## 滚动条 API

下表列出了  $\mu\text{C}/\text{GUI}$  与滚动条相关函数的，函数详细的描述在稍后给出。

函数	说明
<code>SCROLLBAR_AddValue()</code>	滚动条的数值增加或减少一个指定的值。
<code>SCROLLBAR_Create()</code>	创建滚动条。
<code>SCROLLBAR_CreateAttached()</code>	创建一个吸附到一个窗口的滚动条。
<code>SCROLLBAR_CreateIndirect()</code>	从资源表条目中创建滚动条。
<code>SCROLLBAR_Dec()</code>	滚动条数值减 1。
<code>SCROLLBAR_GetValue()</code>	返回当前条目的数值。
<code>SCROLLBAR_Inc()</code>	滚动条数值加 1。
<code>SCROLLBAR_SetNumItems()</code>	设置滚动条目的值。
<code>SCROLLBAR_SetPageSize()</code>	设置页尺寸（按条目的数量）。
<code>SCROLLBAR_SetValue()</code>	设置滚动条当前数值。
<code>SCROLLBAR_SetWidth()</code>	设置滚动条的宽度。

### SCROLLBAR\_AddValue()

#### 定义

滚动条的数值增加或减少一个指定的值。

#### 函数原型

```
void SCROLLBAR_AddValue(SCROLLBAR_Handle hObj, int Add);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。
<code>Add</code>	在一个时间条目增加或减少的数值。

#### 附加信息

滚动条的数值不能超过函数 `SCROLLBAR_SetNumItems()` 当中设置的值。例如，如果窗口包括 200 个条目，当前滚动条数值是 195，滚动条增加 3 个条目，它将移动到数值 198。然而，增加 10 个条目最多也只能移动到数值 200，即该特定窗口的最大数值。

**SCROLLBAR\_Create()****描述**

在指定位置，以指定大小创建一个滚动条控件。

**函数原型**

```
SCROLLBAR_Handle SCROLLBAR_Create ( int x0, int y0,
                                     int xsize, int ysize,
                                     WM_HWIN hParent, int Id,
                                     int WinFlags, int SpecialFlags);
```

参 数	含 意
<code>x0</code>	滚动条最左侧像素（在桌面座标）。
<code>y0</code>	滚动条最顶部像素（在桌面座标）。
<code>xsize</code>	滚动条水平尺寸（以像素为单位）。
<code>ysize</code>	滚动条垂直尺寸（以像素为单位）。
<code>hParent</code>	父窗口句柄。
<code>Id</code>	返回的 ID 号。
<code>WinFlags</code>	窗口创建标志。典型的是 <code>WM_CF_SHOW</code> 用于使控件立即可见（请参阅第 12 章“视窗管理器”中的 <code>WM_CreateWindow()</code> 所列出的参数值）。
<code>SpecialFlags</code>	指定的创建标志（参阅 <code>SCROLLBAR_CreateIndirect()</code> 下的间接创建标志）。

**返回数值**

所创建的滚动条的句柄；如果函数执行失败，则为 0。

**SCROLLBAR\_CreateAttached()****描述**

依附一个已存在的窗口创建一个滚动条。

**函数原型**

```
SCROLLBAR_Handle SCROLLBAR_CreateAttached(WM_HWIN hParent, int SpecialFlags);
```

参 数	含 意
<code>hParent</code>	父窗口的句柄。

SpecialFlags	特定的创建标志(参阅在 SCROLLBAR_CreateIndirect() 下的间接创建标志)。
--------------	---

### 返回数值

所创建滚动条控件的句柄；如果函数执行失败则为 0。

### 附加信息

一个依附的滚动条实质上是一个把它自己放置在父窗口上并因而起作用的子窗口。垂直依附滚动条将自动放置在父窗口的右侧；水平滚动条放置在底部。

### 范例

创建一个带有滚动条的列表框：

```
LISTBOX_Handle hListBox;
hListBox = LISTBOX_Create(ListBox, 50, 50, 100, 100, WM_CF_SHOW);
SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
```

### 上面范例执行结果的屏幕截图

左图展示的是创建后显示的列表框样子；右图展示依附有垂直滚动条的列表框。



## SCROLLBAR\_CreateIndirect()

函数原型的说明在本章开始部分。下面的标志当作作为参数传递的资源表的元素 Flags 使用。

### 允许的间接创建标志（以“OR”组合）

SCROLLBAR_CF_VERTICAL	创建一个垂直滚动条（默认值是水平）。
SCROLLBAR_CF_FOCUSSABLE	给滚动条输入焦点。



在资源表中 Para 元素未使用。

## SCROLLBAR\_Dec()

### 描述

滚动条当前数值减 1。

### 函数原型

```
void SCROLLBAR_Dec(SCROLLBAR_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。

### 附加信息

一个“条目”的定义是特定的应用，尽管在大部分情况下它等于一行。条目编号顺序从上到下或从左到右，以数值 0 开始。

## SCROLLBAR\_GetValue()

### 描述

返回当前条目的数值。

### 函数原型

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。

### 返回数值

当前条目的数值。

## SCROLLBAR\_Inc()

### 描述

滚动条当前数值加 1。

### 函数原型

```
void SCROLLBAR_Inc(SCROLLBAR_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。

### 附加信息

一个“条目”的定义是特定的应用，尽管在大部分情况下它等于一行。条目编号顺序从上到下或从左到右，以数值 0 开始。

## SCROLLBAR\_SetNumItems()

### 描述

设置滚动的条目的数量。

### 函数原型

```
void SCROLLBAR_SetNumItems(SCROLLBAR_Handle hObj, int NumItems);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。
<code>NumItems</code>	设置条目的数量。

### 附加信息

一个“条目”的定义是特定的应用，尽管在大部分情况下它等于一行。

指定的条目数是最大值；滚动条不能超过该值。

**SCROLLBAR\_SetPageSize()****描述**

设置页尺寸。

**函数原型**

```
void SCROLLBAR_SetPageSize(SCROLLBAR_Handle hObj, int PageSize);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。
<code>PageSize</code>	页尺寸 (条目的数量)。

**附加信息**

页尺寸由一页条目的数量指定。如果用户需要向上或向下翻页, 可以使用键盘或通过滚动条区域点击鼠标, 窗口将会按一页指定的条目数量向上或向下滚动。

**SCROLLBAR\_SetValue()****描述**

设置滚动条当前的值。

**函数原型**

```
void SCROLLBAR_SetValue(SCROLLBAR_Handle hObj, int v) ;
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。
<code>v</code>	设置的数值。

**SCROLLBAR\_SetWidth()****描述**

设置滚动条的宽度。

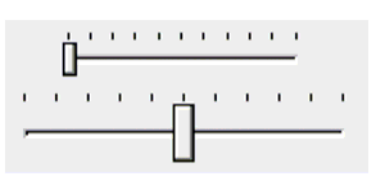
**函数原型**

```
void SCROLLBAR_SetWidth(SCROLLBAR_Handle hObj, int Width);
```

参 数	含 意
<code>hObj</code>	滚动条控件的句柄。
<code>Width</code>	设置的宽度。

## 13.11 SLIDER: 滑动条控件

滑动条控件通常通过使用一个滑动条来改变数值。



所有与滑动条有关的函数位于文件 SLIDER\*.c, SLIDER.h 中。所有的标识符是前缀 SLIDER。

### 配置选项

类型	宏	默认值	说明
N	SLIDER_BKCOLOR0_DEFAULT	0xc0c0c0	背景颜色。
N	SLIDER_COLOR0_DEFAULT	0xc0c0c0	滑动条（滑块 thumb）颜色。
B	SLIDER_USE_3D	1	启用 3D 支持。

### SLIDER API 函数

下表依字母顺序列出了  $\mu$ C/GUI 与滑动条有关的函数。函数详细的描述在下面给出。

函 数	说 明
<code>SLIDER_Create()</code>	创建滑动条。
<code>SLIDER_CreateIndirect()</code>	从资源表条目创建滑动条。
<code>SLIDER_Dec()</code>	滑动条数值减 1。
<code>SLIDER_GetValue()</code>	返回滑动条当前数值。
<code>SLIDER_Inc()</code>	滑动条数值增 1。
<code>SLIDER_SetRange()</code>	设置滑动条数值范围。
<code>SLIDER_SetValue()</code>	设置滑动条当前数值。
<code>SLIDER_SetWidth()</code>	设置滑动条的宽度。

**SLIDER\_Create()****描述**

在指定位置，以指定尺寸创建一个滑动条控件。

**函数原型**

```
SLIDER_Handle SLIDER_Create (  int x0, int y0,
                               int xsize, int ysize,
                               WM_HWIN hParent,
                               int Id,
                               int WinFlags,
                               int SpecialFlags);
```

参 数	含 意
x0	滑动条最左侧像素 (在桌面坐标)。
y0	滑动条最顶部像素 (在桌面坐标)。
xsize	滑动条水平尺寸 (以像素为单位)。
ysize	滑动条垂直尺寸 (以像素为单位)。
hParent	父窗口的句柄。
Id	返回的 ID 号。
WinFlags	窗口创建标志。典型的是 WM_CF_SHOW 用于使控件立即可见 (请参阅第 12 章“视窗管理器”中的 WM_CreateWindow() 所列出的参数值)。
SpecialFlags	特定的创建标志 (参阅 SLIDER_CreateIndirect() 下的间接创建标志)。

**返回数值**

创建的滑动条的控件；如果函数执行失败，则为 0。

**SLIDER\_CreateIndirect()**

函数原型的说明在本章开始部分。下面的标志可以作为当作参数传递的资源表元素 Flags 使用：

**允许的间接标志**

SLIDER\_CF\_VERTICAL      创建一个垂直滑动条 (默认值是水平)。

在资源表中 Para 元素未使用。

## SLIDER\_Dec()

### 描述

滑动条当前条目数减 1。

### 函数原型

```
void SLIDER_Dec(SLIDER_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。

## SLIDER\_GetValue()

### 描述

返回滑动条当前的数值。

### 函数原型

```
int SLIDER_GetValue(SLIDER_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。

### 返回数值

滑动条当前的数值。

## SLIDER\_Inc()

### 描述

滑动条当前条目数加 1。

### 函数原型

```
void SLIDER_Inc(SLIDER_Handle hObj);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。

### **SLIDER\_SetRange()**

#### 描述

设置滑动条的范围。

#### 函数原型

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。
<code>Min</code>	最小值
<code>Max</code>	最大值

#### 附加信息

建立一个滑动条后，其默认值为 0~100。

### **SLIDER\_SetValue()**

#### 描述

设置滑动条当前的数值。

#### 函数原型

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。
<code>v</code>	设置的数值。

### **SLIDER\_SetWidth()**

#### 描述

设置滑动条的宽度。

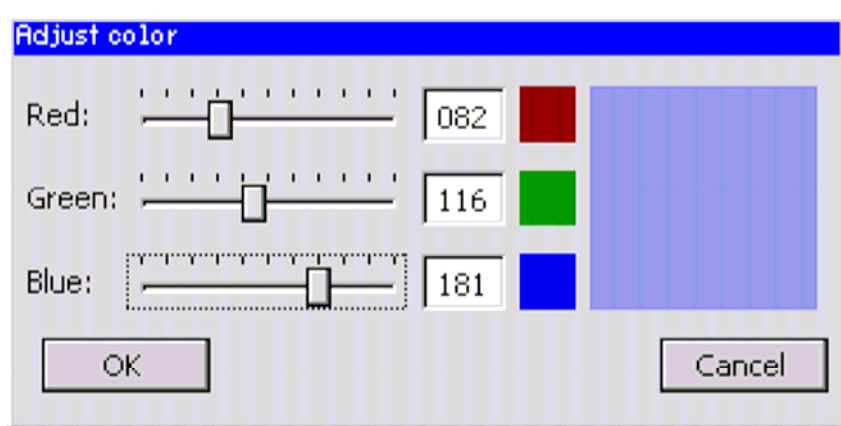
### 函数原型

```
void SLIDER_SetWidth(SLIDER_Handle hObj, int Width);
```

参 数	含 意
<code>hObj</code>	滑动条控件的句柄。
<code>Width</code>	设置的宽度。

### 范例

下面范例的源代码是随 $\mu$ C/GUI 一同提供的范例中的 `DIALOG_SliderColor.c` 文件：



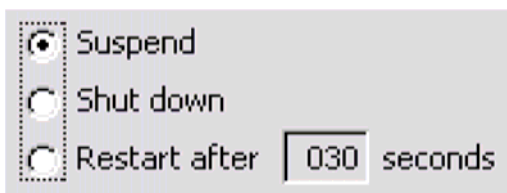
## 13.12 TEXT: 文本控件

文本控件典型用于显示一个对话框的文本区域，如下面所示的消息框：



当然，文本区域也可以用作其它控件的标签，如下所示：





所有与文本有关的函数位于文件 TEXT\*.c, TEXT.h 当中。所有的标识符是前缀 TEXT。

## 配置选项

类型	宏	默认值	说明
S	TEXT_FONT_DEFAULT	&GUI_Font13_1	使用的字体。

## 文本 API 函数

下表依字母顺序列出了  $\mu$ C/GUI 与文本有关的函数。函数详细的描述在下面给出。

函数	说明
TEXT_Create()	创建文本。
TEXT_CreateIndirect()	从资源表条目中创建文本。
TEXT_GetDefaultFont()	返回用于文本的默认字体。
TEXT_SetDefaultFont()	设置用于文本的默认字体。
TEXT_SetFont()	设置用于一个指定的文本控件的字体。
TEXT_SetText()	设置一个指定的文本控件的文本。
TEXT_SetTextAlign()	设置一个指定的文本控件的文本对齐方式。
TEXT_SetTextPos()	设置在一个指定的文本控件当中文本的位置。

## TEXT\_Create()

### 描述

在一个指定位置，指定大小创建一个文本控件。

### 函数原型

```
TEXT_Handle TEXT_Create ( int x0, int y0,
                          int xsize, int ysize,
                          int Id, int Flags,
                          const char* s, int Align);
```

参 数	含 意
-----	-----

x0	滑动条最左侧像素（在桌面坐标）。
y0	滑动条最顶部像素（在桌面坐标）。
xsize	滑动条水平尺寸（以像素为单位）。
ysize	滑动条垂直尺寸（以像素为单位）。
Id	返回的 ID 号。
Flags	窗口创建标志。典型的是 WM_CF_SHOW 用于使控件立即可见（请参阅第 12 章“视窗管理器”中的 WM_CreateWindow() 所列出的参数值）。
s	显示文本的指针。
Align	文本的对齐属性（参阅 TEXT_CreateIndirect() 下的间接创建标志）。

### 返回数值

创建的文件控件的句柄；如果函数执行失败则为 0。

## TEXT\_CreateIndirect()

函数原型的说明在本章开始部分。下面的标志可以用作资源表中当作参数传递的 Flags 元素：

### 允许的间接创建标志（以“OR”组合）

TEXT_CF_LEFT	水平对齐：左
TEXT_CF_RIGHT	水平对齐：右
TEXT_CF_HCENTER	水平对齐：中
TEXT_CF_TOP	垂直对齐：顶
TEXT_CF_BOTTOM	垂直对齐：底
TEXT_CF_VCENTER	垂直对齐：中

资源表中 Para 元素未使用。

## TEXT\_GetDefaultFont()

### 描述

返回用于文本控件的默认字体。

### 函数原型

```
const GUI_FONT* TEXT_GetDefaultFont(void);
```

**返回数值**

用于文本控件的默认字体的指针。

**TEXT\_SetDefaultFont()****描述**

设置用于文本控件的默认字体。

**函数原型**

```
void TEXT_SetDefaultFont(const GUI_FONT* pFont) ;
```

参 数	含 意
pFont	设置为默认值的字体的指针。

**TEXT\_SetFont()****描述**

设置用于指定文本控件的字体。

**函数原型**

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT* pFont);
```

参 数	含 意
HObj	文本控件的句柄。
PFont	使用的字体的指针。

**TEXT\_SetText()****描述**

设置用于指定文本控件的文本。

**函数原型**

```
void TEXT_SetText(TEXT_Handle hObj, const char* s);
```

参 数	含 意
-----	-----

<code>hObj</code>	文本控件的句柄。
<code>s</code>	显示的文本。

**TEXT\_SetTextAlign()****描述**

设置指定文本控件的文本对齐方式。

**函数原型**

```
void TEXT_SetTextAlign(TEXT_Handle hObj, int Align);
```

参 数	含 意
<code>hObj</code>	文本控件的句柄。
<code>s</code>	文本对齐方式（参阅 TEXT_Create()）。

**TEXT\_SetTextPos()****描述**

设置一个指定文本控件中的文本位置。

**函数原型**

```
void TEXT_SetTextPos(TEXT_Handle hObj, int XOff, int YOff);
```

参 数	含 意
<code>hObj</code>	文本控件的句柄。
<code>XOff</code>	文本相对于控件原点 X 轴的偏移距离。
<code>YOff</code>	文本相对于控件原点 Y 轴的偏移距离。