# 第14章 对话框

控件可以创建和独立使用，因为它们自己天生就是窗口。可是，使用对话框常常是很需要的，它是一种包含一个或多个控件的窗口。

一个对话框通常是一个窗口，它的出现要求使用者输入信息。它可能包括多个控件，通过对这些不同的控件的选择向使用者发出请求信息，或者它采用一个提供简单信息（例如提醒使消息框用者注意或警告）和一个“OK”按钮的形式。

# 14.1 对话框基础

### 1. 输入焦点

视察管理器能记住一个窗口或窗口物体最终被选择是通过用户使用触摸屏，鼠标，键盘或者其它的什么。一个窗口收到键盘输入消息了，我们说它有了输入焦点。

记住输入焦点的主要原因是为了确定键盘命令发送到哪去了。具有输入焦点的窗口会接收由键盘产生的事件。

### 2. 模块化和非模块化对话框

对话窗口分为模块化和非模块化两种。一个模块化对话框会阻塞执行的线程。默认情况下，它有输入焦点，必须在线程继续执行前被用户关闭。另一方面，一个非模块化对话框不会阻塞调用的线程--在它显示的时候，它允许任务继续运行。

请注意在其它的一些环境中，模块化和非模块化对话框分别被称为模态和非模态。

### 3. 对话框消息

一个对话框就是一个窗口，它接收消息，就象系统中其它所有窗口做的一们。大多数消息被对话框自动处理了，其它传给了在建立对话框上指定的回调函数（也为对话框程序所知）。

有两种类型的附加消息被送到对话窗口处理：WM_INIT_DIALOG 和 WM_NOTIFY_PARENT。在一个对话框显示前，WM_INIT_DIALOG 消息会立即被发送到对话框处理，而对话框程序对它典型的用法是用这个消息初始化控件，及实现其它影响对话框显示的初始化任务。WM_NOTIFY_PARENT 消息是通过对话框的子窗口发送到对话框的，通知父窗口一些为了保证同步的事件。

# 14.2 建立对话框

建立对话框需要做两件基本的事情：一个资源表，定义包括的控件，和一个对话框程序，定义控件的初始值，与它们的行为一样。一旦两个条件都存在，你只需进行单个函数调用（GUI_CreateDialogBox()或 GUI_ExecDialogBox()）实际建立对话框。

### 1. 资源表

对话框可以基于阻塞（使用 GUI_ExecDialogBox()）或非阻塞（使用

GUI_CreateDialogBox()）方式建立。

必须首先定义一个资源表，以指定包括在对话框中的所有控件。下面的范例展示如何建立一个资源表。这个特定的范例是手工建立的，尽管它也能通过一个 GUI-builder 做到。

```
static const GUI_WIDGET_CREATE_INFO_aDialogCreate[] =
{
    { FRAMEWIN_CreateIndirect,"Dialog",0,10,10,180,230,0,0 },
    { BUTTON_CreateIndirect,"OK",GUI_ID_OK,100,5,60,20,0,0 },
    { BUTTON_CreateIndirect,"Cancel",GUI_ID_CANCEL,100,30,60,20,0,0 },
    { TEXT_CreateIndirect, "LText",0,10,55,48,15,0,GUI_TA_LEFT },
    { TEXT CreateIndirect, "RText",0,10,80,48,15,0,GUI_TA_RIGHT },
    { EDIT_CreateIndirect,NULL,GUI_ID_EDIT0,60,55,100,15,0,50 },
    { EDIT_CreateIndirect,NULL,GUI_ID_EDIT1,60,80,100,15,0,50 },
    { TEXT_CreateIndirect,"Hex",0,10,100,48,15,0,GUI_TA_RIGHT },
    { EDIT_CreateIndirect,NULL,GUI_ID_EDIT2,60,100,100,15,0,6 },
    { TEXT_CreateIndirect,"Bin",0,10,120,48,15,0,GUI_TA_RIGHT },
    { EDIT_CreateIndirect,NULL,GUI_ID_EDIT3,60,120,100,15,0,0 },
    { LISTBOX_CreateIndirect,NULL,GUI_ID_LISTBOX0,10,20,48,40,0,0 },
    { CHECKBOX_CreateIndirect,NULL,GUI_ID_CHECK0,10,5,0,0,0,0 },
    { CHECKBOX_CreateIndirect,NULL,GUI_ID_CHECK1,30,5,0,0,0,0 },
    { SLIDER_CreateIndirect,NULL,GUI_ID_SLIDER0,60,140,100,20,0,0 },
    { SLIDER_CreateIndirect,NULL,GUI_ID_SLIDER1,10,170,150,30,0,0 }
};
```

## 2. 对话框程序

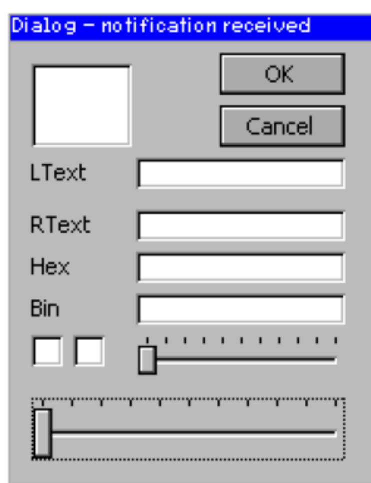
```
/***************************************************************************
*                              对话框程序                                  *
***************************************************************************/

static void_cbCallback(WM_MESSAGE * pMsg)
{
    switch(pMsg->MsgId)
    {
        default: WM_DefaultProc(pMsg);
    }
}
```

对于这个范例，随着下面一行代码的执行，对话框会显示出来。

```
GUI_ExecDialogBox(  _aDialogCreate,
                    GUI_COUNTOF(_aDialogCreate),
                    &_cbCallback,
                    0, 0, 0);
```

结果的对话框看起来如下图一样，或者与之相似（实际外观依赖于你的配置和默认设置）：




建立对话框后，所有包括在资源表中的控件将可见，尽管看起来与上面的屏幕截图差不多，这样控件是以“空”的形式出现。这是因为对话框程序依旧没有包括初始化单个元素的代码。控件的初始化数值，由它们引起的行为，以及它们之间的交互作用都需要在对话框程序中定义。

## 3. 初始化对话框

最典型的下一步是使用它们各自的初始化数值对控件进行初始化。在对话框程序中，这是对 WM_INIT_DIALOG 消息做出反应时的通常做法。下面的程序说明这个事情：

```
/*************************************************************************
*                              对话框程序                                *
*************************************************************************/

static void_cbCallback(WM_MESSAGE * pMsg)
{
    int  NCode, Id;
    WM_HWIN  hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
```

```
    WM_HWIN  hWin = pMsg->hWin;
    switch(pMsg->MsgId)
    {
        case WM_INIT_DIALOG:
        /* 获得所有控件的窗口句柄 */
            hEdit0 = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
            hEdit1 = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
            hEdit2 = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
            hEdit3 = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
            hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
        /* 初始化所有控件 */
            EDIT_SetText(hEdit0, "EDIT widget 0");
            EDIT_SetText(hEdit1, "EDIT widget 1");
            EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
            EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
            EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
            LISTBOX_SetText(hListBox,_apListBox);
            WM_DisableWindow(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK0));
            CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            SLIDER_SetWidth(WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
            SLIDER_SetValue(WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
            break;
        default:
            WM_DefaultProc(pMsg);
    }
}
```

初始化后的对话现在看起来象下面的一样了，所有的控件都有了初始数值：

## 4. 定义对话框行为

对话框一旦被初始化，剩下的所有要向对话框程序添加的代码将定义控件的行为， 使其可充分的操作。继续同一个范例，最终的对话框程序如下所示：

```
/*****************************************************************************
*                              对话框程序                                    *
*****************************************************************************/

static void_cbCallback(WM_MESSAGE * pMsg)
{
    int NCode, Id;
    WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
    WM_HWIN hWin = pMsg->hWin;
    switch(pMsg->MsgId)
    {
        case WM_INIT_DIALOG:
    /* 获得所有控件的窗口句柄 */
            hEdit0 = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
            hEdit1 = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
            hEdit2 = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
            hEdit3 = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
            hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
    /* 初始化所有控件 */
            EDIT_SetText(hEdit0, “EDIT widget 0”);
            EDIT_SetText(hEdit1, “EDIT widget 1”);
            EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
```

```
        EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
        EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
        LISTBOX_SetText(hListBox,_apListBox);
        WM_DisableWindow(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
        CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK0));
        CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
        SLIDER_SetWidth(WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
        SLIDER_SetValue(WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 0);
        break;
    case WM_KEY:
        switch(((WM_KEY_INFO*)(pMsg->Data.p))->Key)
        {
            case GUI_ID_ESCAPE:
                GUI_EndDialog(hWin, 1); break;
            case GUI_ID_ENTER:
                GUI_EndDialog(hWin, 0); break;
        }
        break;
    case WM_NOTIFY_PARENT:
        Id = WM_GetId(pMsg->hWinSrc);          /* 控件的 Id */
        NCode = pMsg->Data.v;                  /* 通知代码 */
        switch(NCode)
        {
            case WM_NOTIFICATION_RELEASED:     /* 如果释放则起作用 */
                if(Id == GUI_ID_OK)
                {
                /* “OK”按钮 */
                    GUI_EndDialog(hWin, 0);
                }
                if(Id == GUI_ID_CANCEL)
                {
                /* “Cancel”按钮 */
                    GUI_EndDialog(hWin, 1);
                }
                break;
            case WM_NOTIFICATION_SEL_CHANGED:  /* 改变选择 */
                FRAMEWIN_SetText(hWin, “Dialog - sel changed”);
```

```
                break;
            default:
                FRAMEWIN_SetText(   hWin,
                                    "Dialog-notification received");
        }
        break;
    default:
        WM_DefaultProc(pMsg);
    }
}
```

想了解更详细的资料，请参考随μC/GUI 一起发布的范例中的 Dialog.c 这个范例的全部代码。

## 14.3 对话框API 参考函数

下表列出了与对话框相关的函数，在各自的类型中按字母顺序进行排列。函数的详细描述后面列出。

| 函　数 | 说　明 |
|---|---|
| 对话框 | |
| GUI_CreateDialogBox | 建立一个非阻塞式的对话框。 |
| GUI_ExecDialogBox | 建立一个阻塞式的对话框。 |
| GUI_EndDialog | 结束一个对话框。 |
| 消息框 | |
| GUI_MessageBox | 建立一个消息框。 |

## 14.4 对话框

### GUI_CreateDialogBox

**描述**

建立一个非阻塞式的对话框。

**函数原型**

```
WM_HWIN GUI_CreateDialogBox (   const GUI_WIDGET_CREATE_INFO* paWidget,
                                int NumWidgets,
```

```
WM_CALLBACK* cb,
WM_HWIN hParent,
int x0, int y0);
```

| 参　数 | 含　意 |
|---|---|
| paWidget | 定义包含在对话框中所有控件的资源表的指针。 |
| NumWidgets | 包含在对话框中所有控件的数量。 |
| cb | 一个具体应用的回调函数的指针（对话框程序）。 |
| hParent | 父窗口的句柄（0 表示没有父窗口）。 |
| x0 | 对话框相对于父窗口的 X 轴坐标。 |
| y0 | 对话框相对于父窗口的 Y 轴坐标。 |

## GUI_ExecDialogBox

### 描述

建立一个阻塞式的对话框。

### 函数原型

```
int GUI_ExecDialogBox ( const GUI_WIDGET_CREATE_INFO* paWidget,
                        int NumWidgets,
                        WM_CALLBACK* cb,
                        WM_HWIN hParent,
                        int x0, int y0);
```

| 参　数 | 含　意 |
|---|---|
| paWidget | 定义包含在对话框中所有控件的资源表的指针。 |
| NumWidgets | 包含在对话框中所有控件的数量。 |
| cb | 一个具体应用的回调函数的指针（对话框程序）。 |
| hParent | 父窗口的句柄（0 表示没有父窗口）。 |
| x0 | 对话框相对于父窗口的 X 轴坐标。 |
| y0 | 对话框相对于父窗口的 Y 轴坐标。 |

## GUI_EndDialog

### 描述

结束（关闭）一个对话框。

**函数原型**

```
void GUI_EndDialog(WM_HWIN hDialog, int r);
```

| 参　数 | 含　意 |
|---|---|
| hDialog | 对话框的句柄。 |

**返回数值**

从建立对话框的函数（只是与 GUI_ExecDialogBox 相关）指定返回调用任务的数值。对于非阻塞方式的对话框，没有等待的应用任务，则返回值被忽略。

# 14.5 消息框

消息框实际上是一种对话框，只不过是它的默认属性被指定了。它只需要一行代码就能建立。一条消息显示在一个带标题栏的窗框内，同时带有一个“OK”按钮，要关闭窗口必需按下它。

## GUI_MessageBox

**描述**

建立及显示一个消息框。

**函数原型**

```
void GUI_MessageBox(const char* sMessage, const char* sCaption, int Flags);
```

| 参　数 | 含　意 |
|---|---|
| sMessage | 显示的消息。 |
| sCaption | 窗口框标题栏的标题内容。 |
| Flags | 为以后的扩展而保留，数值无关紧要。 |

**附加信息**

消息框的默认特性可以通过修改 GUIConf.h 文件中相关内容而改变。下表显示了所有可用的配置宏：

| 类型 | 宏 | 默认值 | 说明 |
|---|---|---|---|
| N | MESSAGEBOX_BORDER | 4 | 消息框元素和客户窗框元素之间的距离。 |
| N | MESSAGEBOX_XSIZEOK | 50 | “OK”按钮的 X 轴尺寸。 |

| N | MESSAGEBOX_YSIZEOK | 20 | “OK”按钮的 Y 轴尺寸。 |
|---|---|---|---|
| S | MESSAGEBOX_BKCOLOR | GUI_WHITE | 客户窗口的背景颜色。 |

消息框显示的内容多于一行是可以的。下面的范例展示如何做到这一点。

**范例**

下面的例子展示了一个简单消息框的用法。

```
/*---------------------------------------------------------------------------
文件:        DIALOG_MessageBox.c
目的:        展示 GUI_MessageBox 的例子
---------------------------------------------------------------------------*/

#include “GUI.h”

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

void main(void)
{
    GUI_Init();
    WM_SetBkWindowColor(GUI_RED);
/* 建立消息框，并等待至其关闭 */
    GUI_MessageBox( “This text is shown\nin a message box”,
                    “Caption/Title”,
                    GUI_MESSAGEBOX_CF_MOVEABLE);
    GUI_Delay(500);
/* 等待一小段时间…… */
    GUI_MessageBox( “New message !”,
                    “Caption/Title”,
                    GUI_MESSAGEBOX_CF_MOVEABLE);
}
```

**上面范例程序执行结果的屏幕截图**

Caption/Title
This text is shown
in a message box
OK