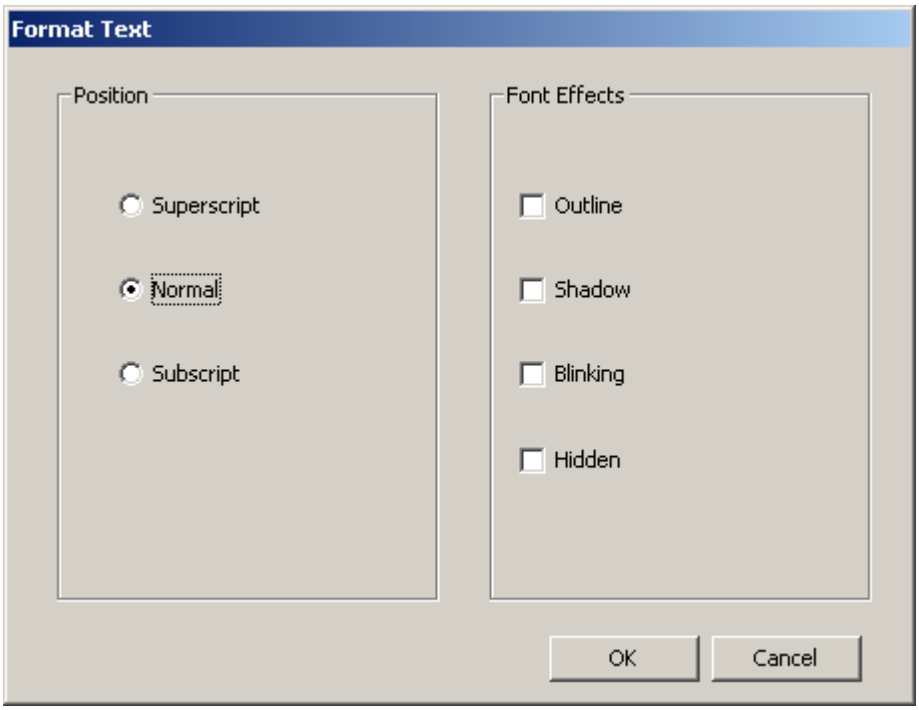
Programming Windows

Terry Marris January 2013

5 Dialog Boxes

A dialog box contains controls such as radio buttons, check boxes and drop down lists. They allow users to select preferences and provide information. A simple dialog box is shown below.



To begin, we take a first look at *resource files*.

5.1 Resource Files

A resource file contains the specifications for objects such as dialog boxes, buttons, drop-down lists, text and menus. You can create the resource file with your favourite text editor, compile it to object code with a resource compiler, and link it into your program. Resource files have their own syntax. You could code dialog boxes, buttons, etc entirely in C. So why bother with resource files?

1. to manage size and complexity. As Windows programs become larger, understanding and managing the code becomes exponentially harder. We split a large program up into a set of coherent program modules. In coding a simple text editor for example, you might have individual modules for file handling, search and replace functions, choosing a font, and printing, and a separate file for all the resources.

2. to separate code from data. The code is the function calls that you make. And the data refers to the message boxes and their controls. In Model-View-Controller architecture, the view component is largely implemented as a resource file.

The header file defines some constants that are shared by both the resource file and the C program.

```
/* dlgbox.h */

#define IDB_OK 101
#define IDB_SUPERSCRIPT 102
#define IDB_NORMAL 103
#define IDB_SUBSCRIPT 104
#define IDB_OUTLINE 105
#define IDB_SHADOW 106
#define IDB_BLINKING 107
#define IDB_HIDDEN 108
#define IDB_CANCEL 109
```

*IDB_* is what I have chosen to represent a button id. Now let's have a look at the resource file.

```
/* dlgboxres.rc - Resource Script for a Dialog Box with
   radio buttons and check boxes */
#include <windows.h>
#include <afxres.h>
#include "dlgbox.h"

#define BTN_WIDTH 50
#define BTN_HEIGHT 14
#define BOX_WIDTH 127
#define BOX_HEIGHT 160

EffectsBoxDlg DIALOG 120, 100, 300, 200
   STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
   FONT 8, "Tahoma"
   CAPTION "Format Text"
{
   GROUPBOX "Position", IDC_STATIC, 16, 10, BOX_WIDTH, BOX_HEIGHT
   AUTORADIOBUTTON "Superscript", IDB_SUPERSCRIPT, 36, 38, 88, 20,
                   WS_GROUP
   AUTORADIOBUTTON "Normal", IDB_NORMAL, 36, 64, 88, 20
   AUTORADIOBUTTON "Subscript", IDB_SUBSCRIPT, 36, 90, 88, 20

   GROUPBOX "Font Effects", IDC_STATIC, 160, 10, BOX_WIDTH, BOX_HEIGHT,
            WS_GROUP
   CHECKBOX "Outline", IDB_OUTLINE, 170, 38, 88, 20, WS_GROUP |
            WS_TABSTOP
   CHECKBOX "Shadow", IDB_SHADOW, 170, 64, 88, 20
   CHECKBOX "Blinking", IDB_BLINKING, 170, 90, 88, 20
   CHECKBOX "Hidden", IDB_HIDDEN, 170, 116, 88, 20

   PUSHBUTTON "OK", IDB_OK, 180, 180, BTN_WIDTH, BTN_HEIGHT, WS_GROUP
   PUSHBUTTON "Cancel", IDB_CANCEL, 234, 180, BTN_WIDTH, BTN_HEIGHT,
              WS_GROUP
}
```

The comments are just like C comments. And the *#include* and *#define* directives are just like they are in C.

afxres.h contains the definition of *IDC_STATIC* - which is discussed in context below.

BTN_WIDTH and *BTN_HEIGHT* defines the width and height of the two buttons, *OK* and *Cancel*. The units are DLU - Dialog Box Unit - and are based on the chosen dialog box font. A horizontal DLU is the average size of a font character divided by four. A vertical DLU is the average height of a font character divided by eight.

```
EffectsBoxDlg DIALOG 120, 100, 300, 200
   STYLE DS_MODALFRAME |WS_POPUP | WS_CAPTION
   FONT 8, "Tahoma"
   CAPTION "Format Text"
{
   ...
}
```

The dialog box is named *EffectsBoxDlg*. *DIALOG* is the keyword that identifies the type of object. *120, 100* specify the co-ordinates, (x, y) of the top left corner of the box, relative to the client area of its parent window. *300, 200* define the box's width and height.

The *STYLE* statement is similar to the style field of *CreateWindow*. A *modal* dialog box requires the user to either supply the requested information or cancel the box before proceeding any further. With modal dialog boxes you usually specify

WS_POPUP - a popup window
WS_SYSMENU - a window menu
WS_CAPTION - a title bar
DS_MODALFRAME - a thick border

The font is set to *Tahoma 8* point. And the caption in the title bar is set to *Format Text*.

The radio buttons and the check boxes are organised into two group boxes.

In

```
   GROUPBOX "Position", IDC_STATIC, 16, 10, BOX_WIDTH, BOX_HEIGHT
```

GROUPBOX introduces a group box. *Position* is the text that appears in the top left corner of the box. *IDC_STATIC* means the box does not send messages to its parent window. *16, 10* refer to the box's top left corner, relative to the top left corner of the dialog box. *BOX_WIDTH* and *BOX_HEIGHT* are both defined near the top of *dlgboxres.rc* shown above.

Next, we come to

```
AUTORADIOBUTTON "Superscript", IDB_SUPERSCRIPT, 36, 38, 88, 20, WS_GROUP
```

AUTORADIOBUTTON defines the type of control, a radio button. *WS_GROUP* marks the beginning of the group of radio buttons. The group extends to immediately before the next occurrence of a *WS_GROUP*. Just one button in an *AUTORADIOBUTTON* group can be selected at any one time. *Superscript* is the text that appears immediately to the right of the

button. *IDB_SUPERSCRIPT*, defined in *dlgbox.h* and shown above, uniquely identifies the button. *36, 38* defines the position of the button. *88, 20* defines its width and height.

In the second group box, *Font Effects*, we find

```
CHECKBOX "Outline", IDB_OUTLINE, 170, 38, 88, 20, WS_GROUP | WS_TABSTOP
```

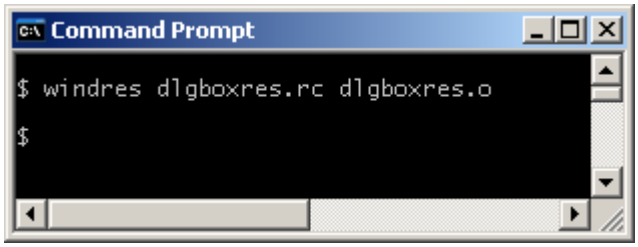
With an *AUTORADIO* button group, just one button may be selected. With a *CHECKBOX* button group, zero, one or many buttons may be selected. *Outline* is the text that appears alongside the button. *IDB_OUTLINE* is its identifier, defined in *dlgbox.h*. *170, 38* is the check box's location, and 88, 20 is its width and height. *WS_GROUP* starts a new group. and *WS_TABSTOP* means the user can reach the group by pressing the tab key.

Finally, we come to the push buttons, the first of which is

```
   PUSHBUTTON "OK", IDB_OK, 180, 180, BTN_WIDTH, BTN_HEIGHT, WS_GROUP
```

PUSHBUTTON is the regular push button. *OK* is the text that appears on the button. *IDB_OK* is its unique identifier defined in *dlgbox.h* shown above. *180, 180* is its position. And *BTN_HEIGHT* and *BTN_WIDTH* are both defined near the beginning of *dlgboxres.rc* shown above.

Having written the resource file, how do you compile it? Very simply if you are using the GNU Compiler collection.



The resource compiler is named *windres*. The extension *.rc* identifies the input resource file, and the *.o* extension identifies the output object file.

We shall see later how to link this object file with the program's object file to create the executable.

5.2 Dialog Box Procedure

The dialog box procedure handles messages to the dialog box. Here, the procedure is named *EffectBoxProc*. The parameters are the same as for a regular windows procedure.

```
LRESULT CALLBACK EffectBoxProc(HWND hDlg, UINT msg,
                               WPARAM wParam, LPARAM lParam)
```

The *WM_INITDIALOG* message deals with initialising the dialog box. The only initialisation we need to do here is to set the default radio button.

```
   switch(msg) {
   case WM_INITDIALOG:
      CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT, IDB_NORMAL);
      return TRUE;
```

The format of *CheckRadioButton* is

CheckRadioButton(hwnd, firstRadioButton, lastRadioButton, defaultRadioButton)

The *WM_COMMAND* messages deal with button clicks. A button's ID is in the *LOWORD* of *wParam*.

When the OK button is clicked we close the dialog box and return to the parent window.

```
case WM_COMMAND:
   switch (LOWORD(wParam)) {
   case IDB_OK:
      EndDialog(hDlg, TRUE);
      break;
```

EndDialog(hDlg, TRUE) returns *TRUE* in the parameter to the application parent window that created the dialog box - see later. *EndDialog* returns zero on error.

If the *Superscript* radio button is selected by the user,

```
case IDB_SUPERSCRIPT:
   CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT,
                                           IDB_SUPERSCRIPT);
```

its button is selected and any other radio button in the group is automatically cleared.

A check box works like a toggle. You click it and it is selected. You click it again and it is deselected. We handle the logic to achieve this.

```
      case IDB_OUTLINE:
         if (BST_CHECKED == IsDlgButtonChecked(hDlg, IDB_OUTLINE))
            CheckDlgButton(hDlg, IDB_OUTLINE, BST_UNCHECKED);
         else
            CheckDlgButton(hDlg, IDB_OUTLINE, BST_CHECKED);
```

IsDlgButtonChecked returns *BST_CHECKED* if the given check box (*IDB_OUTLINE*) is selected. If the box is selected, we uncheck it.

```
   CheckDlgButton(hDlg, IDB_OUTLINE, BST_UNCHECKED);
```

If the given box has not been selected, we check it.

```
   CheckDlgButton(hDlg, IDB_OUTLINE, BST_CHECKED);
```

IsDlgButtonChecked return values include *BST_CHECKED* and *BST_UNCHECKED*.

CheckDlgButton either places a check mark (*BST_CHECKED*) or removes a check mark (*BST_UNCHECKED*) from the given button.

The entire procedure is shown below in section 5.3.

5.3 Calling the Dialog Box

The dialog box procedure is called in response to the *WM_CREATE* message in *WndProc*.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                   WPARAM wParam, LPARAM lParam)
{
   static HINSTANCE hInstance;

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      DialogBox(hInstance, "EffectsBoxDlg", hwnd,
                  (DLGPROC)EffectBoxProc);
      return 0;
```

DialogBox creates a modal dialog box from the given dialog box template, *EffectsBoxDlg*, defined in the resource file, and connects it with the dialog box procedure, *EffectBoxProc*. *DialogBox* returns control only when the dialog box calls *EndDialog*.

Shown below is the entire program.

```
/* dlgbox.c - Radio Buttons and Check Boxes in a dialog box */

#include <windows.h>
#include <stdarg.h>
#include <stdio.h>

#include "dlgbox.h"

#define REP_FATAL 1
#define REP_WARNING 2
#define REP_DIAGNOSTIC 3

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK EffectBoxProc(HWND, UINT, WPARAM, LPARAM);


int report(int repType, PSTR caption, PSTR format, ...)
{
   CHAR string[256];
   va_list args;

   va_start(args, format);
   vsprintf(string, format, args);
   va_end(args);

   MessageBox(NULL, string, caption, MB_OK);

   if (repType == REP_FATAL)
     PostQuitMessage(0);
   return 0;
}
```

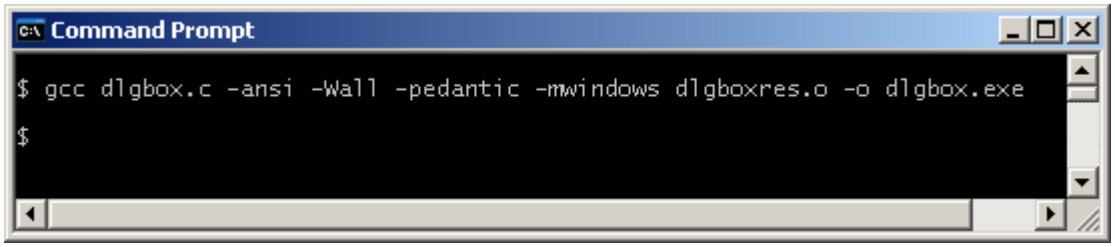
```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   PSTR cmdLine, int cmdShow)
{
   static char appName[] = "dlgbox";
   HWND hwnd;
   MSG msg;
   WNDCLASS wc;
   BOOL ret;

   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInst;
   wc.hIcon         = NULL;
   wc.hCursor       = LoadImage(NULL, IDC_ARROW, IMAGE_CURSOR,
                        0, 0, LR_SHARED);
   wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName  = NULL;
   wc.lpszClassName = appName;

   if (0 == (RegisterClass(&wc)))
      report(REP_FATAL, "WinMain", "%s", "RegisterClass failed");
   if (NULL == (hwnd = CreateWindow(appName,
                       "Dialog Box",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,
                       NULL,
                       hInst,
                       NULL)))
      report(REP_FATAL, "WinMain", "%s", "CreateWindow failed");
   ShowWindow(hwnd, cmdShow);
   UpdateWindow(hwnd);

   while ((ret = GetMessage(&msg, NULL, 0, 0)) != 0) {
      if (ret < 0)
         report(REP_FATAL, "WinMain", "%s", "GetMessage failed");
      else {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
   return msg.wParam ;
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                   WPARAM wParam, LPARAM lParam)
{
   static HINSTANCE hInstance;

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      DialogBox(hInstance, "EffectsBoxDlg", hwnd,
                   (DLGPROC)EffectBoxProc);

      return 0;
```

```
   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;
   }
   return DefWindowProc(hwnd, msg, wParam, lParam);
}


LRESULT CALLBACK EffectBoxProc(HWND hDlg, UINT msg,
                          WPARAM wParam, LPARAM lParam)
{
   switch(msg) {
   case WM_INITDIALOG:
      CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT, IDB_NORMAL);
      return TRUE;

   case WM_COMMAND:
      switch (LOWORD(wParam)) {
      case IDB_OK:
         EndDialog(hDlg, TRUE);
         break;

      case IDB_CANCEL:
         EndDialog(hDlg, TRUE);
         break;

      case IDB_SUPERSCRIPT:
         CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT,
                                                 IDB_SUPERSCRIPT);
         break;

      case IDB_NORMAL:
         CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT,
                                                 IDB_NORMAL);
         break;

      case IDB_SUBSCRIPT:
         CheckRadioButton(hDlg, IDB_SUPERSCRIPT, IDB_SUBSCRIPT,
                                                 IDB_SUBSCRIPT);
         break;

      case IDB_OUTLINE:
         if (BST_CHECKED == IsDlgButtonChecked(hDlg, IDB_OUTLINE))
            CheckDlgButton(hDlg, IDB_OUTLINE, BST_UNCHECKED);
         else
            CheckDlgButton(hDlg, IDB_OUTLINE, BST_CHECKED);
         break;

      case IDB_SHADOW:
         if (BST_CHECKED == IsDlgButtonChecked(hDlg, IDB_SHADOW))
            CheckDlgButton(hDlg, IDB_SHADOW, BST_UNCHECKED);
         else
            CheckDlgButton(hDlg, IDB_SHADOW, BST_CHECKED);
         break;
```

```
      case IDB_BLINKING:
         if (BST_CHECKED == IsDlgButtonChecked(hDlg, IDB_BLINKING))
            CheckDlgButton(hDlg, IDB_BLINKING, BST_UNCHECKED);
         else
            CheckDlgButton(hDlg, IDB_BLINKING, BST_CHECKED);
         break;

      case IDB_HIDDEN:
         if (BST_CHECKED == IsDlgButtonChecked(hDlg, IDB_HIDDEN))
            CheckDlgButton(hDlg, IDB_HIDDEN, BST_UNCHECKED);
         else
            CheckDlgButton(hDlg, IDB_HIDDEN, BST_CHECKED);
         break;
      }

      return TRUE;
   }
   return FALSE;
}
```

5.4 Linking Resource File with Source File

Linking the resource file with the C object file to create the executable is straightforward.



The object file, *dlgboxres.o*, previously created with *windres*, is listed after *-mwindows* but before the output flag *-o*.

5.5 List Box

You might use a simple list box if there are more than, say, five entries in a radio button group.

In this simple list box the user can make a selection and confirm it by pressing the OK button.

The header file defines identifiers for both the push button and the list box.

```
/* lbox.h */

#define IDB_OK 101
#define IDL_BOX 102
```

*IDL_* is the code I have chosen to identify a list box.

The resource script for the list and its dialog box is shown below.

```
/* lboxres.rc - Resource Script for a Dialog Box with a list box */

#include <windows.h>
#include "lbox.h"

ListBoxDlg DIALOG 120, 64, 75, 85
   STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
   FONT 8, "Tahoma"
   CAPTION "Title"
{
   LISTBOX IDL_BOX, 12, 12, 50, 50,
                  LBS_NOTIFY | WS_VSCROLL | WS_BORDER
   PUSHBUTTON "OK", IDB_OK, 12, 60, 50, 14, WS_GROUP
}
```

Look at the *LISTBOX* specification. *IDL_BOX* is its identifier, defined in *lbox.h* shown above, and used in the program shown below. *12, 12* defines its position within the dialog frame. *50, 50* define its width and height. *LBS_NOTIFY* allows the parent window to receive *WM_COMMAND* messages from the list box.

Values are assigned, and the default value is set, in the list box procedure.

```
LRESULT CALLBACK ListBoxProc(HWND hDlg, UINT msg,
                             WPARAM wParam, LPARAM lParam)
{
   ...

   switch(msg) {
   case WM_INITDIALOG:
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Mr");
      ...
      SendDlgItemMessage(hDlg, IDL_BOX, LB_SETCURSEL, 0, 0);
      return TRUE;
```

The *LB_ADDSTRING* message is used to add text entries to the list box, *IDL_BOX*. In the absence of the *LBS_SORT* style (see *lboxres.rc* above) the entries appear in the order they are added. LB_SETCURSEL is used to set the default value. Which item is selected as the default value? The values are indexed from zero upwards starting with the first item. And the zero shown emboldened identifies the first value in the list.

```
      SendDlgItemMessage(hDlg, IDL_BOX, LB_SETCURSEL, 0, 0);
```

SendDlgItemMessage sends a message (*LB_SETCURSEL*) to the given control (*IDL_BOX*) in the dialog box (*hDlg*).

Having initialised the dialog box and its list box control, we need to process its messages.

```
...
   HWND hwndList;
   int index;
   char title[10];
...
case WM_COMMAND:
      ...

      case IDL_BOX:
         switch (HIWORD(wParam)) {
         case LBN_SELCHANGE:
            hwndList = GetDlgItem(hDlg, IDL_BOX);
            index = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
            SendMessage(hwndList, LB_GETTEXT, index, (LPARAM)title);
            report(REP_DIAGNOSTIC, "ListBox", "%s selected", title);
            break;
         }
         ...
```

Messages from the list box are contained in the *HIWORD* component of *wParam*. The *LBN_SELCHANGE* message indicates that the user has made a change to the list box.

A handle to the given list box is retrieved by *GetDlgItem*. The *index* of the current selection is retrieved by a call to *SendMessage*, and its text is copied into *title* in response to the message *LB_GETTEXT*. We assume *title* is large enough. And, finally, the value of *title* is displayed by *report*.

GetDlgItem returns *NULL* on error.

Here is the entire program listing.

```
/* lbox.c - list box */

#include <windows.h>
#include <stdarg.h>
#include <stdio.h>

#include "lbox.h"

#define REP_FATAL 1
#define REP_WARNING 2
#define REP_DIAGNOSTIC 3

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ListBoxProc(HWND, UINT, WPARAM, LPARAM);


int report(int repType, PSTR caption, PSTR format, ...)
{
   CHAR string[256];
   va_list args;

   va_start(args, format);
   vsprintf(string, format, args);
   va_end(args);

   MessageBox(NULL, string, caption, MB_OK);

   if (repType == REP_FATAL)
     PostQuitMessage(0);
   return 0;
}


int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   PSTR cmdLine, int cmdShow)
{
   static char appName[] = "lbox";
   HWND hwnd;
   MSG msg;
   WNDCLASS wc;
   BOOL ret;

   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInst;
   wc.hIcon         = NULL;
   wc.hCursor       = LoadImage(NULL, IDC_ARROW, IMAGE_CURSOR,
                        0, 0, LR_SHARED);
   wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName  = NULL;
   wc.lpszClassName = appName;

   if (0 == (RegisterClass(&wc)))
      report(REP_FATAL, "WinMain", "%s", "RegisterClass failed");
```

```
   if (NULL == (hwnd = CreateWindow(appName,
                       "List Box",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,
                       NULL,
                       hInst,
                       NULL)))
      report(REP_FATAL, "WinMain", "%s", "CreateWindow failed");


   ShowWindow(hwnd, cmdShow);
   UpdateWindow(hwnd);

   while ((ret = GetMessage(&msg, NULL, 0, 0)) != 0) {
      if (ret < 0)
         report(REP_FATAL, "WinMain", "%s", "GetMessage failed");
      else {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
   return msg.wParam ;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                   WPARAM wParam, LPARAM lParam)
{
   static HINSTANCE hInstance;

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      DialogBox(hInstance, "ListBoxDlg", hwnd,
                   (DLGPROC)ListBoxProc);
      return 0;

   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;
   }
   return DefWindowProc(hwnd, msg, wParam, lParam);
}


LRESULT CALLBACK ListBoxProc(HWND hDlg, UINT msg,
                           WPARAM wParam, LPARAM lParam)
{
   HWND hwndList;
   int index;
   char title[10];
```

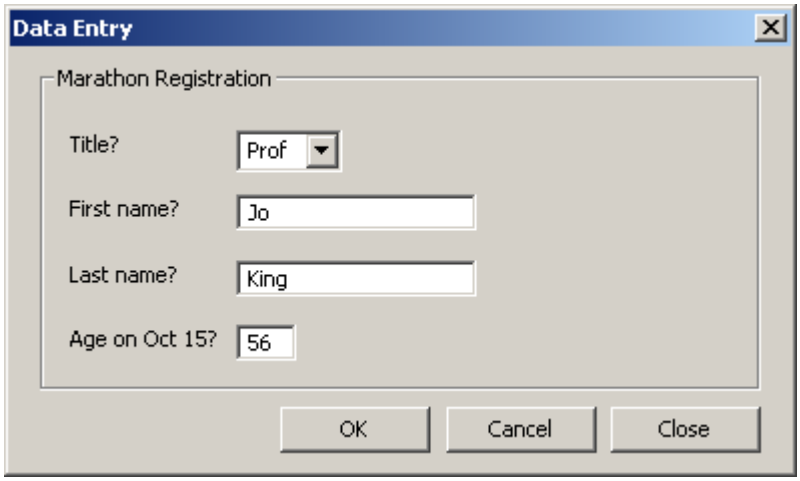
```
   switch(msg) {
   case WM_INITDIALOG:
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Mr");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Mrs");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Ms");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Dr");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Rev");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_ADDSTRING,
                         0, (LPARAM)"Prof");
      SendDlgItemMessage(hDlg, IDL_BOX, LB_SETCURSEL, 0, 0);
      return TRUE;

   case WM_COMMAND:
      switch (LOWORD(wParam)) {
      case IDB_OK:
         EndDialog(hDlg, LOWORD(wParam));
         break;

      case IDL_BOX:
         switch (HIWORD(wParam)) {
         case LBN_SELCHANGE:
            hwndList = GetDlgItem(hDlg, IDL_BOX);
            index = (int)SendMessage(hwndList, LB_GETCURSEL, 0, 0);
            SendMessage(hwndList, LB_GETTEXT, index, (LPARAM)title);
            report(REP_DIAGNOSTIC, "ListBox", "%s selected", title);
            break;
         }
      }
      return TRUE;
   }
   return FALSE;
}
```

6.6 Combo Box

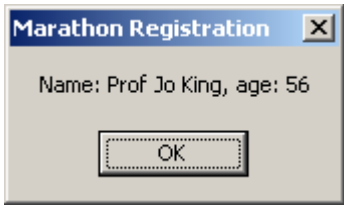
A combo box is a combination of an editable string and a drop down list. Clicking on the down arrow reveals the list, and the user is able to make a selection. The selected item appears in the editable text box.

The editable text box allows the user to enter text. If the first few letters match any of the list entries, it appears in the text box.

We see how to create a simple screen data entry form.



The title, Mr, Ms, Dr, Prof, ... is selected from the drop down list. The user enters text for a name, and for an age in years. The *Close* button closes the dialog box. The *Cancel* button clears the entries made by the user. Clicking on the *OK* button results in the entries being echoed in a message box.



We define some constants to be used in both the resource file and in the C source code.

```
/* dataentry.h */

#define IDB_OK 101
#define IDB_CANCEL 102
#define IDB_CLOSE 103

#define IDC_TITLE 201
#define IDE_FIRSTNAME 202
#define IDE_LASTNAME 203
#define IDE_AGE 204
```

*IDB_* identifies a button. *IDC_* identifies a combo box. *IDE_* identifies an editable input field.

Now we come to the resource file.

```
/* dataentryres.rc - Data entry Dialog Box */
#include <windows.h>
#include <afxres.h>
#include "dataentry.h"

DataEntryBoxDlg DIALOG 70, 70, 260, 130
   STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
   FONT 8, "Tahoma"
   CAPTION "Data Entry"
{
   GROUPBOX "Marathon Registration", IDC_STATIC, 10, 5, 240, 100
   LTEXT "Title?", 0, 20, 25, 60, 8
   COMBOBOX IDC_TITLE, 75, 25, 35, 80, WS_TABSTOP | WS_VSCROLL |
                                                    CBS_DROPDOWN

   LTEXT "First name?", 0, 20, 45, 60, 8
   EDITTEXT IDE_FIRSTNAME, 75, 45, 80, 11, WS_TABSTOP, WS_EX_WINDOWEDGE

   LTEXT "Last name?", 0, 20, 65, 60, 8
   EDITTEXT IDE_LASTNAME, 75, 65, 80, 11, WS_TABSTOP, WS_EX_WINDOWEDGE

   LTEXT "Age on Oct 15?", 0, 20, 85, 60, 8
   EDITTEXT IDE_AGE, 75, 85, 20, 11, WS_TABSTOP, WS_EX_WINDOWEDGE

   PUSHBUTTON "OK", IDB_OK, 90, 110, 50, 14, WS_GROUP
   PUSHBUTTON "Cancel", IDB_CANCEL, 145, 110, 50, 14, WS_GROUP
   PUSHBUTTON "Close", IDB_CLOSE, 200, 110, 50, 14, WS_GROUP
}
```

WS_CAPTION means the dialog box window has a title bar, and *WS_SYSMENU* says put a window close icon on the title bar.

LTEXT displays text left justified in the defined rectangle. The *0* in *0, 20, 25, 60, 8* represents it's id. *CTEXT* centres text and *RTEXT* shows text right justified in its defined rectangle.

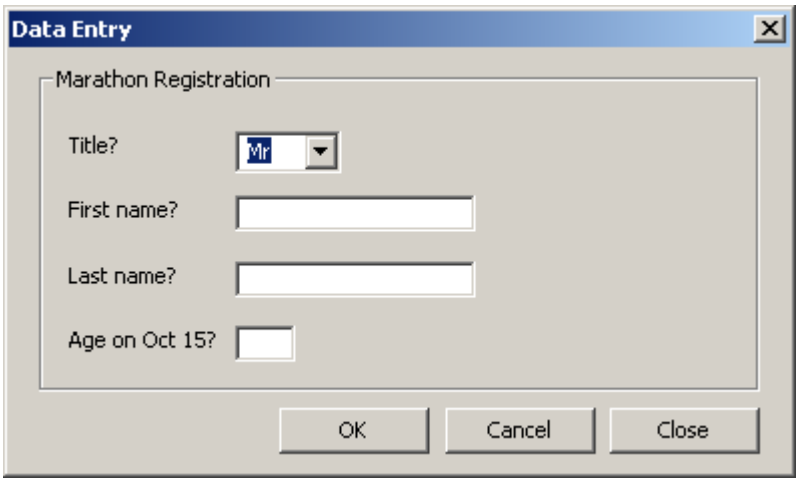
COMBOBOX introduces a combo box. *CBS_DROPDOWN* is a drop down list shown only when the user clicks on its down arrow.

EDITTEXT introduces a an editable single line text box. *WS_EX_WINDOWEDGE*, an extended style, specifies that the window has a border with a raised edge.

Now we come to the procedure that processes the combo box. The first task is to set up the combo box along with its drop down list.

```
   static HWND hwndTitleComBox;
   const char *title[] = { "Mr","Ms","Mrs","Dr","Rev","Prof" };
   int i;
   static char pTitle[10];
      ...
   case WM_INITDIALOG:
      hwndTitleComBox = GetDlgItem(hDlg, IDC_TITLE);
      for (i = 0; i < sizeof(title) / sizeof(title[0]); i++)
         SendMessage(hwndTitleComBox, CB_ADDSTRING, 0,
                     (LPARAM)title[i]);
      SendMessage(hwndTitleComBox, CB_SETCURSEL, 0, 0);
      strcpy(pTitle, "Mr");
```

SendMessage(hwndTitleComBox, CB_SETCURSEL, 0, 0) sets the first item in the drop down list as the default item. It is shown with a blue background.



You may remember *SendMessage* has the format

SendMessage(hwnd, message, wParam, lParam)

The default index, *0*, is set in *wParam*. *lParam* is not used. *CB_SETCURSEL* selects a string in the combo box list.

The initial value is preserved in *pTitle*.

```
strcpy(pTitle, "Mr");
```

We assume *pTitle* is sufficiently large to accommodate any title.

A change in the title selected by the user is signalled in *HIWORD(wParam)*.

```
int i;
...
case WM_COMMAND:
   if (HIWORD(wParam) == CBN_SELCHANGE) {
      i = (int)SendMessage(hwndTitleComBox, CB_GETCURSEL, 0, 0);
      SendMessage(hwndTitleComBox, CB_GETLBTEXT, (WPARAM)i,
                                                 (LPARAM)pTitle);
```

CB_GETCURSEL retrieves the index of the currently selected item, and this is stored in *i*. *CB_GETLBTEXT* retrieves the string indexed by *i* from a list box, and stores it in *pTitle*. *pTitle* must be large enough.

On clicking the *OK* button, the data entered by the user is retrieved.

```
HWND hwndFirstName,
PTSTR pFirstName,
int len;
...
```

```
      case IDB_OK:
         hwndFirstName = GetDlgItem(hDlg, IDE_FIRSTNAME);
         ...
         len = GetWindowTextLength(hwndFirstName);
         pFirstName = (PTSTR)malloc((len + 1) * sizeof(char));
         GetWindowText(hwndFirstName, pFirstName, len + 1);
```

Finally, the data collected is displayed for inspection.

```
report(REP_DIAGNOSTIC, "Marathon Registration",
          "Name: %s %s %s, age: %s", pTitle, pFirstName, pLastName, pAge);
```

The complete program is shown below.

```
/* dataentry.c - data entry */

#include <windows.h>
#include <stdarg.h>
#include <stdio.h>

#include "dataentry.h"

#define REP_FATAL 1
#define REP_WARNING 2
#define REP_DIAGNOSTIC 3

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK DataEntryBoxProc(HWND, UINT, WPARAM, LPARAM);


int report(int repType, PSTR caption, PSTR format, ...)
{
   CHAR string[256];
   va_list args;

   va_start(args, format);
   vsprintf(string, format, args);
   va_end(args);

   MessageBox(NULL, string, caption, MB_OK);

   if (repType == REP_FATAL)
     PostQuitMessage(0);
   return 0;
}


int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   PSTR cmdLine, int cmdShow)
{
   static char appName[] = "lbox";
   HWND hwnd;
   MSG msg;
   WNDCLASS wc;
   BOOL ret;
```

```
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInst;
   wc.hIcon         = NULL;
   wc.hCursor       = LoadImage(NULL, IDC_ARROW, IMAGE_CURSOR,
                        0, 0, LR_SHARED);
   wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName  = NULL;
   wc.lpszClassName = appName;

   if (0 == (RegisterClass(&wc)))
      report(REP_FATAL, "WinMain", "%s", "RegisterClass failed");

   if (NULL == (hwnd = CreateWindow(appName,
                       "List Box",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,
                       NULL,
                       hInst,
                       NULL)))
      report(REP_FATAL, "WinMain", "%s", "CreateWindow failed");

   ShowWindow(hwnd, cmdShow);
   UpdateWindow(hwnd);

   while ((ret = GetMessage(&msg, NULL, 0, 0)) != 0) {
      if (ret < 0)
         report(REP_FATAL, "WinMain", "%s", "GetMessage failed");
      else {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
   return msg.wParam ;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                   WPARAM wParam, LPARAM lParam)
{
   static HINSTANCE hInstance;

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      DialogBox(hInstance, "DataEntryBoxDlg", hwnd,
                  (DLGPROC)DataEntryBoxProc);
      return 0;

   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;
   }
   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

```
LRESULT CALLBACK DataEntryBoxProc(HWND hDlg, UINT msg,
                          WPARAM wParam, LPARAM lParam)
{
   static HWND hwndTitleComBox;
   const char *title[] = { "Mr","Ms","Mrs","Dr","Rev","Prof" };
   int i;

   HWND hwndFirstName, hwndLastName, hwndAge;
   PTSTR pFirstName, pLastName, pAge;

   static char pTitle[10];
   int len;

   switch(msg) {
   case WM_INITDIALOG:
      hwndTitleComBox = GetDlgItem(hDlg, IDC_TITLE);
      for (i = 0; i < sizeof(title) / sizeof(title[0]); i++)
         SendMessage(hwndTitleComBox, CB_ADDSTRING, 0,
                     (LPARAM)title[i]);
      SendMessage(hwndTitleComBox, CB_SETCURSEL, 0, 0);
      strcpy(pTitle, "Mr");
      return TRUE;

   case WM_CLOSE:
      EndDialog(hDlg, LOWORD(wParam));
      return TRUE;

   case WM_COMMAND:
      if (HIWORD(wParam) == CBN_SELCHANGE) {
         i = (int)SendMessage(hwndTitleComBox, CB_GETCURSEL, 0, 0);
         SendMessage(hwndTitleComBox, CB_GETLBTEXT, (WPARAM)i,
                                                    (LPARAM)pTitle);
         return TRUE;
      }

      switch (LOWORD(wParam)) {
      case IDB_CANCEL:
         SendMessage(hwndTitleComBox, CB_SETCURSEL, 0, 0);
         SetWindowText(GetDlgItem(hDlg, IDE_FIRSTNAME), "");
         SetWindowText(GetDlgItem(hDlg, IDE_LASTNAME), "");
         SetWindowText(GetDlgItem(hDlg, IDE_AGE), "");
         break;

      case IDB_CLOSE:
         EndDialog(hDlg, LOWORD(wParam));
         break;

      case IDB_OK:
         hwndFirstName = GetDlgItem(hDlg, IDE_FIRSTNAME);
         hwndLastName = GetDlgItem(hDlg, IDE_LASTNAME);
         hwndAge = GetDlgItem(hDlg, IDE_AGE);

         len = GetWindowTextLength(hwndFirstName);
         pFirstName = (PTSTR)malloc((len + 1) * sizeof(char));
         GetWindowText(hwndFirstName, pFirstName, len + 1);

         len = GetWindowTextLength(hwndLastName);
         pLastName = (PTSTR)malloc((len + 1) * sizeof(char));
         GetWindowText(hwndLastName, pLastName, len + 1);
```

```
         len = GetWindowTextLength(hwndAge);
         pAge = (PTSTR)malloc((len + 1) * sizeof(char));
         GetWindowText(hwndAge, pAge, len + 1);

        report(REP_DIAGNOSTIC, "Marathon Registration",
          "Name: %s %s %s, age: %s", pTitle, pFirstName, pLastName, pAge);

        free(pFirstName);
        free(pLastName);
        free(pAge);
        pFirstName = NULL;
        pLastName = NULL;
        pAge = NULL;
        break;
      }
      return TRUE;
   }
   return FALSE;
}
```

6.7 Values Returned By Window Procedures

We know that *WndProc*, called by *WinMain*, returns either 0 or the result returned by *DefWindowProc*. But what about other Window procedures?

Notice that *DataEntryBoxProc* does not return a call to *DefWindowProc*. Apparently, unwanted messages are automatically processed. Typically, a dialog box procedure, *DataEntryBoxProc* processes a dialog box, returns *TRUE* if it procesess a message, and *FALSE* if it does not. *TRUE* and *FALSE* are perversly returned even though the procedure is defined as returning an integer.

7.8 CALLBACK

When should we specify *CALLBACK* in the functions we write? When Windows needs to call our function in response to an event generated by an API call. You can determine whether a function requires *CALLBACK* by looking at its parameters. If you look at

```
LRESULT CALLBACK DataEntryBoxProc(HWND hDlg, UINT msg,
                                  WPARAM wParam, LPARAM lParam)
```

its parameter types include *WPARAM* and *LPARAM*, the message parameters. Windows sends messages in these parameters to your function to process (or ignore).

Next, we take a look at menus.