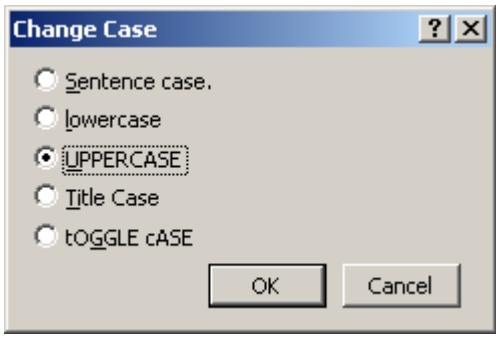
Programming Windows

Terry Marris January 2013

4 Buttons

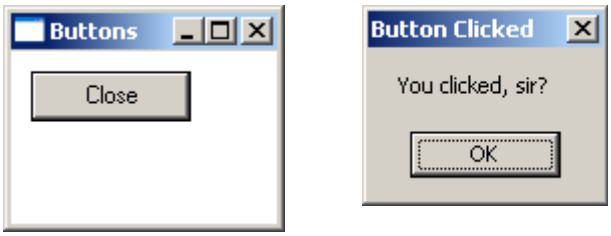
Buttons in some format or other are found in many windows programs. For example, these radio buttons and push buttons are found in the Microsoft Word 2000 Format menu, Change Case option.



We take a first look at push buttons.

4.1 Push Button

A push button usually provides an immediate response when clicked by the user. The next program, shown below, displays a *Close* button which when clicked, results in a message box with the text *You clicked, sir?* being shown.



We create the button, and then check the response to button clicks.

A button is created using *CreateWindow*.

```
#define IDB_CLOSEBUTTON 101
...
HWND hwndButton;
...
hwndButton = CreateWindow(
                  "BUTTON", "Close",
                  WS_TABSTOP | WS_CHILD | WS_VISIBLE |
                  BS_PUSHBUTTON,
                  10, 10, 75, 23,
                  hwnd,
                  (HMENU)IDB_CLOSEBUTTON,
                  (HINSTANCE)GetWindowLongPtr(hwnd, GWL_HINSTANCE),
                  NULL));
```

BUTTON is a pre-defined class name. *Close* is the button's caption (A button's caption is sometimes called the button's label). WS_TABSTOP specifies the user can use the tab key to reach the button control. *BS_PUSHBUTTON* defines the button style. 10, 10 defines the position of the top left corner of the button in its window. 75 represents the button's width. 23 represents the button's height. The units are in pixels. *IDB_CLOSEBUTTON* is the identifier that represents the button when clicked.

"During a *WM_CREATE* message, *lParam* is actually a pointer to a structure of type *CREATESTRUCT* ("creation structure") that has a member *hInstance*. So we cast *lParam* into a pointer to a *CREATESTRUCT* structure and get *hInstance out*." [Petzold C Programming Windows 95 page 366.]

The default font for the button's caption is awful. We set its font to the default GUI font (even though the MSDN documentation says we should not, and that we should create and use a font, such as Tahoma - but that makes the coding a little more cluttered - see below)

```
.
HFONT hFont;
...
hFont = GetStockObject(DEFAULT_GUI_FONT);
```

GetStockObject returns a handle to the given font.

Then we send a set font message to the button.

```
SendMessage(hwndButton, WM_SETFONT, (WPARAM)hFont, MAKELPARAM(TRUE, 0));
```

SendMessage has the format

SendMessage(hwnd, message, wParam, lParam)

MAKELPARAM creates a value for use as an *LPARAM* argument from the given arguments *TRUE* and *0*. Why *TRUE* and *0* and not something else? I haven't a clue!

GetWindowLongPtr retrieves information about the given window. *GWL_HINSTANCE* retrieves a handle to the program's instance. *GetWindowLongPtr* returns zero on error.

Having created a button, we might like to know when it is clicked.

The high order bits of *wParam* will tell you if a button has been clicked, and the low order bits will tell you which button.

```
case WM_COMMAND:
   if (HIWORD(wParam) == BN_CLICKED) {
      btnId = LOWORD(wParam);

      switch (btnId) {
      case IDB_CLOSEBUTTON:
         report(REP_FATAL, "Button Clicked", "%s", "You clicked, sir?");
         break;
      }
   }
   return 0;
```

If the button with id *IDB_CLOSEBUTTON* has been clicked (*IDB_CLOSEBUTTON* was *#defined* at the top of the program and used in the button's creation) a call to *report* is made. *report* terminates program execution if its first argument is *REP_FATAL*.

Shown below is the entire program.

```
/* buttons.c - Displays buttons */

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

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

#define IDB_CLOSEBUTTON 101

LRESULT CALLBACK WndProc(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[] = "buttons";
   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,
                       "Buttons",
                       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)
{
   HFONT hFont;
   HWND hwndButton;
   int btnId;

   switch (msg) {
   case WM_CREATE:
      if (NULL == (hwndButton = CreateWindow(
                  "BUTTON", "Close",
                  WS_TABSTOP | WS_CHILD | WS_VISIBLE |
                  BS_PUSHBUTTON,
                  10, 10, 75, 23,
                  hwnd,
                  (HMENU)IDB_CLOSEBUTTON,
                  (HINSTANCE)GetWindowLongPtr(hwnd, GWL_HINSTANCE),
                  NULL)))
         report(REP_FATAL, "WndProc", "%s", "Create button failure");

      hFont = GetStockObject(DEFAULT_GUI_FONT);
      SendMessage(hwndButton, WM_SETFONT, (WPARAM)hFont,
                  MAKELPARAM(TRUE, 0));

      return 0;
```

```
   case WM_COMMAND:
      if (HIWORD(wParam) == BN_CLICKED) {
         btnId = LOWORD(wParam);

         switch (btnId) {
         case IDB_CLOSEBUTTON:
            report(REP_FATAL, "Button Clicked", "%s",
                   "You clicked, sir?");
            break;
         }
      }
      return 0;

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

If you want to use a different font, such as Tahoma ...

```
HFONT hFont;
HDC hdc;
TEXTMETRIC tm;
LOGFONT logFont;
int yChar;
int pointSize = 8;
char *faceName = "Tahoma";

HWND hwndButton;
int btnId;

switch (msg) {
case WM_CREATE:
   hdc = GetDC(hwnd);
   ZeroMemory(&logFont, sizeof(logFont));
   logFont.lfHeight = -MulDiv(pointSize,
                        GetDeviceCaps(hdc, LOGPIXELSY), 72);
   strcpy(logFont.lfFaceName, faceName);
   hFont = CreateFontIndirect(&logFont);

   GetTextMetrics(hdc, &tm);
   yChar = tm.tmHeight + tm.tmExternalLeading;
   ReleaseDC(hwnd, hdc);

   if (NULL == (hwndButton = CreateWindow(
                  "BUTTON", "OK",
                  WS_TABSTOP | WS_CHILD | WS_VISIBLE |
                  BS_PUSHBUTTON,
                  10, 10, 75, 7 * yChar / 4,
                  hwnd,
                  (HMENU)IDB_CLOSEBUTTON,
                  (HINSTANCE)GetWindowLongPtr(hwnd, GWL_HINSTANCE),
                  NULL)))
         report(REP_FATAL, "WndProc", "%s", "Create button failure");

   SendMessage(hwndButton, WM_SETFONT, (WPARAM)hFont, MAKELPARAM(TRUE,0));
   return 0;
```

A *TEXTMETRIC* structure contains the information about a physical font. *tmExternaLeading* is the space between rows.

A *LOGFONT* structure defines the attributes of a font, characteristics such as height, width and face name e.g. Arial, Tahoma and Times New Roman.

ZeroMemory clears a block of memory and fills it with zeros. The MSDN documentation says it would be safer to use *SecureZeroMemory* instead.

MulDiv multiplies the first two arguments, and then divides the result with the third argument. The result is rounded up to the nearest integer if the result is positive, rounded down if the result is negative.

CreateFontIndirect creates a logical font with the given characteristics.

In the next chapter on dialog boxes we look at radio buttons and check boxes.

Resources

Microsoft have provided an excellent discussion on button styles in

http://msdn.microsoft.com/en-us/library/windows/desktop/aa511482.aspx