Using GUI Features with a Win32 Console Application

- Revised Version-

Written by

Tegdeep Kondal

TABLE OF CONTENTS

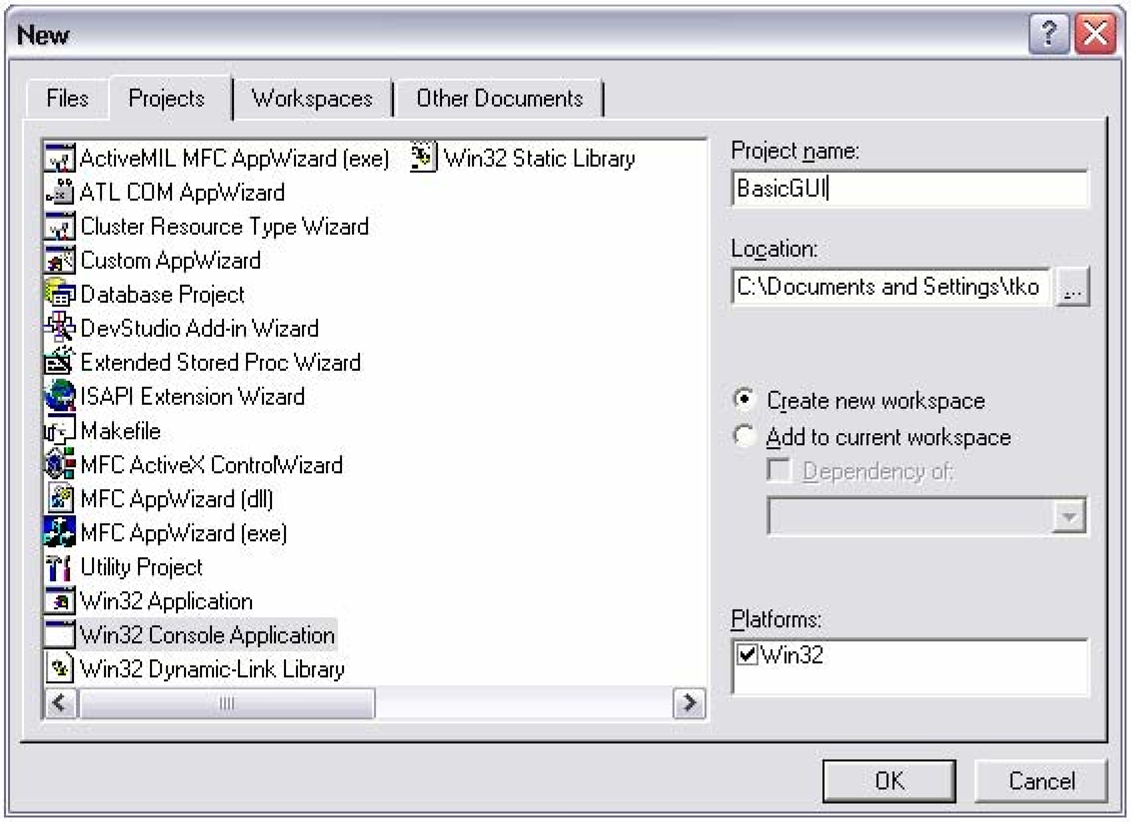
1. PURPOSE

The purpose of this document is to familiarize the reader with basic GUI features that can be used with a console. This guide only teaches the basics of GUI applications; the reader is encouraged to research other features that could be needed.

This document is step-by-step guide to developing nice GUI applications while maintaining a classic console application.

2. A SIMPLE BASIC GUI APPLICATION

In order to start using GUI features in your application, simply create a new Win32 Console Application. This is the same procedure as creating a basic C++ application.



Project name: BasicGUI

Location: C:\Documents and Settings\tko

(● Create new workspace)

Platforms: ✓ Win32

3. BASIC GUI CODE TEMPLATE

Some of these sections can be modified to accommodate other features.

3.1 PREPROCESSOR DIRECTIVES

```
#define   WIN32_WINNT        0x0400    /* Keep for compatibility issues */
#define   WINVER             0x0400    /* Keep for compatibility issues */

#include  <windows.h>                  /* Basic Window commands     */
#include  <iostream.h>                 /* I/O stream commands       */
#include  "resource.h"                 /* Include your resources    */
#include  "console.h"                  /* Console functions         */

#define   NUM_BTN   2                  /* Number of hover buttons   */
```

The only changes that should be made here are the first two lines and the last line.

The first two lines are used to setup the version of the OS. Only values greater than the ones already there can be used. An attempt of going to a previous version will not permit the hover effect.

The last line contains the number of hover buttons present in the GUI dialog. There should be at least one button, “Exit”, for which hover text should be displayed and that permits the user to close the dialog box. Another button, “ClickMe”, was added to demonstrate how this template works.

3.2 GUI GLOBAL VARIABLES

```
HWND BtnHWND [NUM_BTN];     /* This is an array containing the HWND to the hover buttons. */
BOOL bGUIAppDone = false;   /* This controls the flow of the message loop.                 */
WNDPROC DefaultBtnProc;     /* Default PROC for buttons.                                    */
LPCSTR HoverBtnText;        /* Hover button information string.                             */
HWND hWndDlg;               /* Window Handle of the Dialog                                  */
HINSTANCE hInstance;        /* Hinstance.                                                   */
```

The first line is an array that holds the Window Handle of each hover button. The second line is a very important line; the Boolean bGuiAppDone contains the state of the GUI dialog. If it is set to true, the GUI is terminated. Since it is a global variable, it is possible to modify it in any function of the program. The third line stores the Default Window PROC for the buttons. The fourth line contains the Hover Text to be displayed whenever the mouse hovers. It is meant to be global so that it may be changed in any part of the program. The fifth line is a global handle to the dialog window. The last line simply contains the instance of the program. This is used for many GUI functions.

3.3 BASIC GUI FUNCTION PROTYPES

These are the basic GUI functions that are needed. There is no need to modify these functions unless you really want to.

```
INT CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam);
INT CALLBACK BtnProc(HWND hwndBtn, UINT uMsg, WPARAM wParam, LPARAM lParam);
void SetDialogBkColor(HWND hwndDlg, COLORREF TheColor);
void SetBtnProc(HWND hwnd);
void SetBtnInfo(HWND BtnHwnd);
```

The first two lines are the dialog and button window PROC's. The third line controls the background color of the GUI. The last two functions are used for setting up the hover buttons.

3.4 PROTOTYPE OF MOUSE CLICK FUNCTIONS

Simply add the function prototype of the functions that should be executed when the user clicks on a particular button.

```
void OnClickMe();                    /* When user clicks on "Click Me" button, execute this .*/
```

3.5 MAIN FUNCTION

This is the actual main() function that the compiler will run. It is the same as the one made for basic C++ Console Applications. The only difference is that this one has a message loop and it does not end till the bGUIAppDone is set to true.

3.6 BASIC GUI FUNCTION IMPLEMENTATIONS

The basic GUI function implementations can be modified if need be. The first function is used to set the background color of the GUI dialog. The parameters are a handle to the window to be painted and the color to use.

```
void SetDialogBkColor(HWND hwndDlg, COLORREF TheColor)
  {
    HDC hdc = GetDC(hwndDlg);
    HBRUSH DlgBkBrush = CreateSolidBrush(TheColor);
    RECT DlgRect;
    GetClientRect(hwndDlg, &DlgRect);

    HPEN MyPen = CreatePen(PS_SOLID, 1, TheColor);
    SelectObject(hdc,MyPen);
    SelectObject(hdc,DlgBkBrush);

    Rectangle(hdc,DlgRect.left, DlgRect.top, DlgRect.right, DlgRect.bottom);
    ReleaseDC(hwndDlg, hdc);
    DeleteObject(DlgBkBrush);
    DeleteObject(MyPen);
    UpdateWindow(hwndDlg);
  }
```

The second function is the most important function of the GUI application; it handles the messages sent to the GUI dialog window. It is also in this function that the buttons are implemented. The code listing below shows an implementation of a button whose resource ID is IDC_BUTTON1. When this button is clicked, the function OnClickMe() will be executed.

```
INT CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
  {

    /* Used to repaint the window. */
    RECT DlgRect;
    HDC hdc = GetDC(hwndDlg);

    switch(uMsg)
      {
        /*Process MouseClicks. */
        case(WM_COMMAND):

          switch(LOWORD(wParam))
```

Defining what function to execute for the button

```
            {
            //ADD BUTTONS HERE

            case IDC_BUTTON1:      /* Button Resource ID.              */
              OnClickMe();         /* Function to execute when clicked. */
              return TRUE;         /* Processed message.               */
            break;

            case IDC_EXIT:
              DestroyWindow(hwndDlg);
              bGUIAppDone=TRUE;
              return TRUE;
            break;
            }

        break;

        case (WM_CTLCOLORSTATIC):
            {
            HBRUSH BgBrush = CreateSolidBrush(RGB(255,255,255));
            return (int)BgBrush;
            }
        break;

        case (WM_ERASEBKGND):

          GetClientRect(hwndDlg, &DlgRect);
          SetDialogBkColor(hwndDlg, RGB(255,255,255));
          InvalidateRect(hwndDlg, &DlgRect, FALSE) ;
          return TRUE;
        break;

        case (WM_CLOSE):
          DestroyWindow(hwndDlg);
          return TRUE;
        break;

        case (WM_DESTROY):
          bGUIAppDone=TRUE;
          return TRUE;
        break;

        }

    return 0;
  }
```

As it can be seen, the WM_CTLCOLORSTATIC and WM_ERASEBKGND messages paint the elements in white. The color can be modified to whatever color is required. Other messages can also be added here however, it is required to return TRUE after the message is processed.

The listing below controls the hover effect. It is possible to modify the code such that another message than “Welcome …” is displayed in the hover text box (whose resource name is IDC_INFO). Simply change “Welcome …” in the WM_MOUSELEAVE message below for the desired message.

```
/* Button PROC. */
INT CALLBACK BtnProc(HWND hwndBtn, UINT uMsg, WPARAM wParam, LPARAM lParam)
  {
    BOOL doTrack = true;

    /* If Mouse moves on button, set the hover text. */
    if(uMsg == WM_MOUSEMOVE)
      {
        doTrack = false;
        SetBtnInfo(hwndBtn);
        SetWindowText(GetDlgItem(GetParent(hwndBtn),IDC_INFO), HoverBtnText);
      }

    /* Track mouse. */
    TRACKMOUSEEVENT TrkMouseEvent;
    TrkMouseEvent.cbSize=sizeof(TrkMouseEvent);
    TrkMouseEvent.dwFlags = TME_LEAVE | TME_HOVER;
    TrkMouseEvent.dwHoverTime = 0;
    TrkMouseEvent.hwndTrack = hwndBtn;
    if(!doTrack)
      TrackMouseEvent(&TrkMouseEvent);

    /* Set the appropriate hover text. */
    SetBtnInfo(hwndBtn);

    /* Set hover text if mouse hovers. */
    if(uMsg == WM_MOUSEHOVER)
      SetWindowText(GetDlgItem(GetParent(hwndBtn),IDC_INFO), HoverBtnText);

    /* Reset default hover text when  mouse leaves.*/
    if(uMsg == WM_MOUSELEAVE)
      {
        SetWindowText(GetDlgItem(GetParent(hwndBtn),IDC_INFO), "Welcome …");
        doTrack=false;
      }

    /* Call default PROC to manage other messages. */
   return CallWindowProc( DefaultBtnProc, hwndBtn, uMsg, wParam, lParam);
  }
```

It is time to set the hover messages. First, the SetBtnProc() function will take care of setting the HWND of the hover buttons specified in it. To add buttons simply copy the procedure for the IDC_BUTTON1 button below.

```
void SetBtnProc(HWND hwnd)
  {
    /* Associate Exit Button and set PROC. */
    BtnHWND[0] = GetDlgItem(hwnd,IDC_EXIT);
    SetWindowLong(GetDlgItem(hwnd, IDC_EXIT), GWL_WNDPROC, (LONG) BtnProc);

    /* Associate ClickMe button and set PROC. */
    BtnHWND[1] = GetDlgItem(hwnd,IDC_BUTTON1);
    SetWindowLong(GetDlgItem(hwnd, IDC_BUTTON1), GWL_WNDPROC, (LONG) BtnProc);
  }
```

Now, set the hover messages for the corresponding hover button. Note that the indexing depends on the programmer. Just make sure that each hover button has a unique index.

```
void SetBtnInfo(HWND BtnHwnd)
  {
    /* Exit button. */
    if(BtnHwnd == BtnHWND[0])
      HoverBtnText = "Click here if you wish to exit from the Edge Module Demo Launcher.";

    /* ClickMe button. */
    if(BtnHwnd == BtnHWND[1])
      HoverBtnText ="Click Me && Find out what I do. \n Come on click ... click ... click...";

  }
```

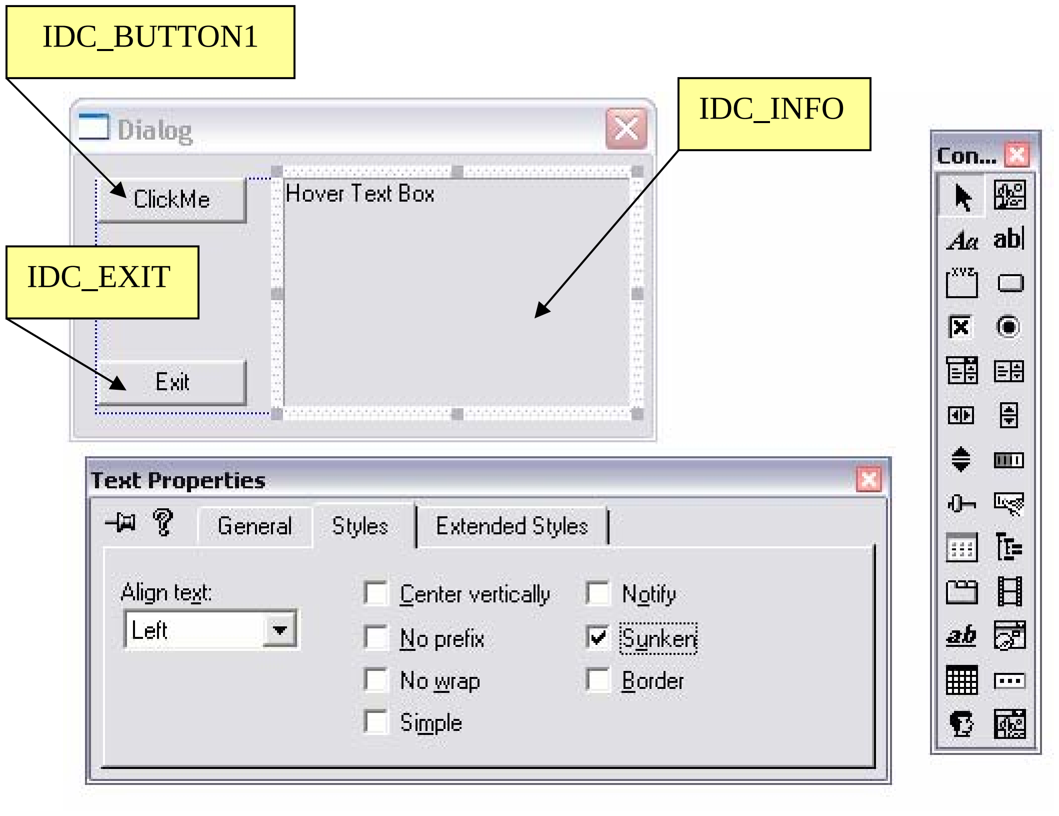
3.7 MOUSE CLICK FUNCTION IMPLEMENTATIONS

Finally, the application is almost complete. The only thing remaining to do is to implement our OnClickMe() function and set up our dialog box. Let the function warn the user that it is going to write to the console (using a message box) and then, write to the console.

```
void OnClickMe()
  {
    MessageBox(NULL, "Click Me", "I have written CLICK on the console 10 times!", MB_OK);
    for(int i=0;i<10; i++)
      cout<<"CLICK"<<endl;
  }
```

4. SETTING UP THE DIALOG BOX

To setup the dialog box, simply open the resource file ***"BasicGUI.rc"*** and add the ClickMe Button. Now, arrange the layout of the dialog box to whatever you like; you may also change the styles of the resources.



Align text: Left

[ ] Center vertically
[ ] No prefix
[ ] No wrap
[ ] Simple
[ ] Notify
[✓] Sunken
[ ] Border

7. CONSOLE FUNCTIONS

The following are useful functions that may be needed in a GUI application. These are all located in the ***"console.h"*** header file. Other functions may be added, or if need be, these ones can be modified. Use these functions whenever you want a colorful console, or if you want to control the console size. It even detects mouse clicks within the console window.

void GetMouseClick() Gets a mouse click in the console.
void SetConsolePosition(int x, int y, int width, int height) Sets the console window’s position and size to the specified values in the parameters.
HWND GetConsoleHwnd() Retrieve the Console’s window handle.
void WriteOffset(int x, int y, const char* str) Write a string at an offset position in the console.
void ClearScreen(int BGColor) Clear the console’s buffer with a specified background color. Black is the default color.
void SetConsoleTextPosition(int x, int y) Set the text cursor’s position in the console.
void SetConsoleTextColor(int FGColor, int BGColor) Set the console’s text color. The default foreground is white, and the background is black
HANDLE GetInputHandle () Retrieve the input handle of the console.
HANDLE GetOutputHandle () Retrieve the output handle of the console.
void ShowConsole() Show Console window.

void HideConsole()

Hide Console window.

8. WHY USE CONSOLE FUNCTIONS?

Remember, the objective of this guide was to be able to use the console as a basic element in the GUI application. This means that we want to be able to input from the console and output to the console. The console functions can be used to set the color of the text, the position of the text. Even a mouse click can be implemented in the console, instead of using getchar(). The user may implement other functions. For example, if one wishes to create a “Text based button”, a function can be implemented to verify the coordinates of the mouse cursor and use the GetMouseClick() function.

APPENDIX A: LISTING OF CONSOLE.H

```
#ifndef CONSOLE_H
#define CONSOLE_H

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

/* The following defines the available colors for the console */
#define C_BLACK           0
#define C_DARK_BLUE       1
#define C_DARK_GREEN      2
#define C_DARK_CYAN       3
#define C_DARK_RED        4
#define C_PURPLE          5
#define C_OLIVE           6
#define C_LIGHT_GREY      7
#define C_DARK_GREY       8
#define C_BLUE            9
#define C_GREEN          10
#define C_CYAN           11
#define C_RED            12
#define C_PINK           13
#define C_YELLOW         14
#define C_WHITE          15

// ButtonState flags
#define FROM_LEFT_1ST_BUTTON_PRESSED    0x0001
#define RIGHTMOST_BUTTON_PRESSED        0x0002
#define FROM_LEFT_2ND_BUTTON_PRESSED    0x0004
#define FROM_LEFT_3RD_BUTTON_PRESSED    0x0008
#define FROM_LEFT_4TH_BUTTON_PRESSED    0x0010

// EventFlags
#define MOUSE_MOVED   0x0001
#define DOUBLE_CLICK  0x0002
#define MOUSE_WHEELED 0x0004

void GetMouseClick();
void SetConsolePosition(int x, int y, int width, int height);
HWND GetConsoleHwnd();
void WriteOffset(int x, int y, const char* str);
void ClearScreen(int BGColor);
void SetConsoleTextPosition(int x, int y);
void SetConsoleTextColor(int FGColor, int BGColor);
HANDLE GetInputHandle();
HANDLE GetOutputHandle();
void ShowConsole();
void HideConsole();

int g_BGColor = C_BLACK;
int g_FGColor = C_WHITE;

/* Show Console. */
void ShowConsole()
  {
    SetWindowPos(GetConsoleHwnd(), FindWindow(NULL,"Demo
Launcher"),0,0,0,0,SWP_NOSIZE|SWP_NOMOVE|SWP_NOZORDER);
    ShowWindow(GetConsoleHwnd(), SW_SHOW);

  }

/* Hide Console. */
void HideConsole()
  {
    ShowWindow(GetConsoleHwnd(), SW_HIDE);
  }

HANDLE GetOutputHandle()
  {
    return GetStdHandle(STD_OUTPUT_HANDLE);
  }

HANDLE GetInputHandle()
  {
    return GetStdHandle(STD_INPUT_HANDLE);
  }

void SetConsoleTextColor(int FGColor, int BGColor)
  {
    SetConsoleTextAttribute(GetOutputHandle(), FGColor+(BGColor*16));
  }

void SetConsoleTextPosition(int x, int y)
  {
    COORD pos={x,y};
    SetConsoleCursorPosition(GetOutputHandle(), pos);
  }

void ClearScreen(int BGColor)
  {
    g_BGColor = BGColor;
    SetConsoleTextColor(g_FGColor, BGColor);

    COORD coordScreen = { 0, 0 };
    DWORD cCharsWritten;
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    DWORD dwConSize;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    GetConsoleScreenBufferInfo(hConsole, &csbi);
    dwConSize = csbi.dwSize.X * csbi.dwSize.Y;
    FillConsoleOutputCharacter(hConsole, TEXT(' '), dwConSize,
      coordScreen, &cCharsWritten);
    GetConsoleScreenBufferInfo(hConsole, &csbi);
    FillConsoleOutputAttribute(hConsole, csbi.wAttributes, dwConSize,
      coordScreen, &cCharsWritten);
    SetConsoleCursorPosition(hConsole, coordScreen);
  }

void WriteOffset(int x, int y, const char* str)
  {
      SetConsoleTextPosition(x, y);
      printf(str);
   }

HWND GetConsoleHwnd()
   {
       HWND hwndFound;
       char TempWindowTitle[1024];
       char WindowTitle[1024];

       GetConsoleTitle(WindowTitle, 1024);

       wsprintf(TempWindowTitle,"%d/%d",
                   GetTickCount(),
                   GetCurrentProcessId());

       SetConsoleTitle(TempWindowTitle);
       Sleep(40);
       hwndFound=FindWindow(NULL, TempWindowTitle);
       SetConsoleTitle(WindowTitle);
       return(hwndFound);
   }

void SetConsolePosition(int x, int y, int width, int height)
   {
      SetWindowPos(GetConsoleHwnd(), NULL, x, y, width, height, SWP_SHOWWINDOW);
   }


void GetMouseClick()
   {
   INPUT_RECORD InputRecord;
   DWORD Events;

   while(1)
      {
      ReadConsoleInput(GetStdHandle(STD_INPUT_HANDLE), &InputRecord, 1, &Events);
      //printf("%d\n", InputRecord.Event.MouseEvent.dwMousePosition.X);
      //printf("%d\n", InputRecord.Event.MouseEvent.dwMousePosition.Y);
      if(InputRecord.Event.MouseEvent.dwButtonState == FROM_LEFT_1ST_BUTTON_PRESSED)
         break;

      }
   }

#endif
```