***Using Visual Studio Editor:***

Building all the controls of Win32 screen(as described below) can be very time consuming.

Define a Dialog in the resource file (\*.rc), and load it.

***A - Open Editor Steps***

A screenshot of a computer

Description automatically generated with medium confidence

1. Locate the rc file of your project, and double click (will open the Resource View)
2. Select the Dialog item in the list of resources
3. Right Click and select ***Insert Dialog***
4. ***The editior will open and you can build a screen***

***B – Modify you Win32 Window Instantiation***

In the Win32 main or init function replace the call to theCreateWindow or CreateWindowW

By

HWND hWnd = CreateDialog(hInstance, MAKEINTRESOURCE(IDD\_MAIN\_WINDOW), NULL, (DLGPROC)WndProc);

**Note**: Note the ID resource of your dialog resource in the editor (here IDD\_MAIN\_WINDOW)

***C – Connect Controls to Global handle***

Since all the control dialog are created when the dialog resource is loaded, there is no HANDLE for all the controls of your windows. And this is required often for dynamic action

(i.e Enable or Disable a control)

HWND hPlayButton = GetDlgItem(hWnd, IDC\_PLAY\_BUTTON);

***Basic Win32 Controls:***

***Note:*** In all the following example, make sure to:

***1 - Define a constant in*** Resource.h

#define IDM\_BUTTON\_ID\_ RESOURCE something

***2 – Add a Case in the main Window’s procedure***

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_ IDM\_BUTTON\_ID\_ RESOURCE:

DoSomething();

break;

***}***

***1 - Label:***

In main program, Window Procedure, inside WM\_CREATE

HWND hLabel = CreateWindow(L"Static", L"220",

WS\_CHILD | WS\_VISIBLE,

10, 85,

DEFAULT\_LABEL\_WIDTH, DEFAULT\_LABEL\_HEIGHT,

hWnd, (HMENU)1, NULL, NULL);

***2 - Push Button:***

In main program, Window Procedure:

HWND hButton = CreateWindow(

L"BUTTON", // Predefined class; Unicode assumed

L"Your Text Here", // Button text

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, // Styles

10, // x position

10, // y position

DEFAULT\_COMMAND\_BUTTON\_WIDTH, // Button width

DEFAULT\_COMMAND\_BUTTON\_HEIGHT, // Button height

hWnd, // Parent window

(HMENU)IDM\_BUTTON\_ID\_ RESOURCE, // No menu.

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL); // Pointer not needed.

***Enabling Control:***

Few goods macros are discussed here

<https://docs.microsoft.com/en-us/windows/win32/api/windowsx/nf-windowsx-button_enable?redirectedfrom=MSDN>

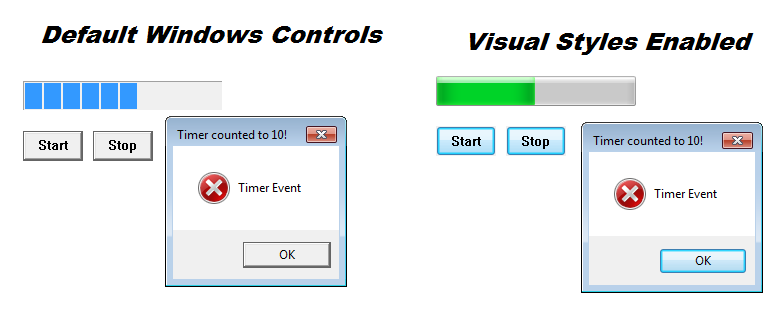
void Button\_Enable(handleToContro, FALSE);

*Note*: add windowsx.h in the include (Won’t compile otherwise)

#include <windowsx.h>

***Visual Styles***

Modern Look and Feel can be available if the Visual Styles are enable. See difference below



The following line have to be at the beginning of the main cpp file of you application

#pragma comment( linker, "/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

Aug 2021: It doesn’t appear to work when you create dialogs from a RC file

A direct call to CreateWindow(X,Y,Z) appears to work dough…

***Create Transparent Win32 label...***

Create Transparent Win32 label and have control of the color of their font…

Using the WM\_PAINT message of the window is too cumbersome…

Add the follow code in your Win32 WndProc method:

case  WM\_CREATE:

   {

     //  Make sure to assign an ID to the static label, when you create it

    // here IDS\_YPOS\_LABEL  is used.

    CreateWindow(L"STATIC",

                      L"Y",      // Button text

                      WS\_CHILD | WS\_VISIBLE,

                      120, 23,

                      10, 20,

                      hWnd,

                      (HMENU)IDS\_YPOS\_LABEL,

                      (HINSTANCE)GetWindowLong(hWnd, GWL\_HINSTANCE),

                      NULL);

  }

  case WM\_CTLCOLORSTATIC:

        {

            //  Since there is an ID associated with the label, the Win32

            //function GetDlgItem can find it HANDLE easily.

    //that way, you don’t need to keep a HANDLE variable in your code, to keep track…

            if (GetDlgItem(hWnd, IDS\_YPOS\_LABEL) == (HWND)lParam)

            {

                SetBkMode((HDC)wParam, TRANSPARENT);

                SetTextColor((HDC)wParam, RGB(255, 0, 0));

                return (BOOL)GetStockObject(HOLLOW\_BRUSH);

            }

        }

        break;

***Modify Color of text in Win32 TextBox***

The message WM\_CTLCOLOREDIT is sent to the main Window, and a Device Context is passed via the wParam parameter. Here some explanation from MicroSoft

<https://docs.microsoft.com/en-us/windows/win32/controls/wm-ctlcolorstatic#examples>

So, you need to add an handle to the WM\_CTLCOLOREDIT message

Add the follow code in your Win32 WndProc method:

case  WM\_CREATE:

   {

      //  Same thing as previous topic (transparent label)

//make sure to assign an ID to the edit control in the

Menu param (IDC\_EDIT\_MOUSE\_XPOS,)

       CreateWindow(L"EDIT",

                         L"",

                        WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT | ES\_AUTOHSCROLL | ES\_WANTRETURN,

                       30, 20, 80, 25,

                        hWnd,

                       (HMENU)IDC\_EDIT\_MOUSE\_XPOS,

                        hInst,

                        0);

  case WM\_CTLCOLOREDIT:

        {

            if (GetDlgItem(hWnd, IDC\_EDIT\_MOUSE\_XPOS) == (HWND)lParam)

            {

                SetTextColor((HDC)wParam, RGB(255, 0, 0));

            }

        }

***Writing String***

To have correct String format in a Win32 application is a bit tedious: A classic

char myVar[Mydimensin];

TextOut(hDC, 0, 0, myVar, strlen(myVar));

Will have at best, horrible result (a Chinese Text).

You need a LPCWSTR (Windows’s Long Pointer to a string) as following:

char text[] = "something";

wchar\_t wtext[20];

mbstowcs(wtext, szMsg, strlen(szMsg) + 1);//Plus null

LPWSTR ptr = wtext;

TextOut(hDC, 0, 0, ptr, strlen(szMsg) + 1 );

***Writing String to Console***

Using a straight printf won’t work in a Win32 application. Use the function OutputDebugString instead.

OutputDebugString(\_T("My output string."));