Customing the standard color-picker dialog

devblogs.microsoft.com/oldnewthing/20140707-00

July 7, 2014

Raymond Chen

Today's Little Program does a little bit of customization of the `ChooseColor` dialog. We do this by, um, doing what the documentation says.

For the color dialog, we take the template in `color.dlg` and modify it. Just to get our feet wet, we won't customize anything at all! This ensures that we have the basics down before we start trying anything fancy.

Handy tip: Before trying to customize something, first write code that does it uncustomized. That way, you have a known working starting point.

```
#include <windows.h>
#include <colordlg.h>
1 DIALOG LOADONCALL MOVEABLE DISCARDABLE 2, 0, 298, 184
STYLE WS_BORDER | DS_MODALFRAME | WS_CAPTION | WS_POPUP | WS_SYSMENU |
      DS_3DLOOK
CAPTION "Color"
FONT 8 "MS Shell Dlg"
BEGIN
    LTEXT           "&Basic colors:", -1, 4, 4, 140, 9
    CONTROL         "", COLOR_BOX1, "static",
                    SS_SIMPLE | WS_CHILD | WS_TABSTOP | WS_GROUP,
                    4, 14, 140, 86
    LTEXT           "&Custom colors:", -1, 4, 106, 140, 9
    CONTROL         "",  COLOR_CUSTOM1, "static",
                    SS_SIMPLE | WS_CHILD | WS_TABSTOP | WS_GROUP,
                    4, 116, 140, 28
    PUSHBUTTON      "&Define Custom Colors >>" COLOR_MIX, 4, 150, 138, 14,
                    WS_TABSTOP | WS_GROUP
    DEFPUSHBUTTON   "OK", IDOK, 4, 166, 44, 14, WS_GROUP | WS_TABSTOP
    PUSHBUTTON      "Cancel", IDCANCEL, 52, 166, 44, 14, WS_GROUP | WS_TABSTOP
    PUSHBUTTON      "&Help", pshHelp, 100, 166, 44, 14, WS_GROUP | WS_TABSTOP
    CONTROL         "", COLOR_RAINBOW, "static",
                    SS_SUNKEN | SS_SIMPLE | WS_CHILD, 152, 4, 118, 116
    CONTROL         "", COLOR_LUMSCROLL, "static",
                    SS_SUNKEN | SS_SIMPLE | WS_CHILD, 280, 4, 8, 116
    CONTROL         "", COLOR_CURRENT, "static",
                    SS_SUNKEN | SS_SIMPLE | WS_CHILD, 152, 124, 40, 26
    PUSHBUTTON      "&o", COLOR_SOLID, 300, 200, 4, 14, WS_GROUP
    RTEXT           "Color", COLOR_SOLID_LEFT, 152, 151, 20, 9
    LTEXT           "|S&olid", COLOR_SOLID_RIGHT, 172, 151, 20, 9
    RTEXT           "Hu&e:", COLOR_HUEACCEL, 194, 126, 20, 9
    EDITTEXT,       COLOR_HUE, 216, 124, 18, 12, WS_GROUP | WS_TABSTOP
    RTEXT           "&Sat:", COLOR_SATACCEL, 194, 140, 20, 9
    EDITTEXT,       COLOR_SAT, 216, 138, 18, 12, WS_GROUP | WS_TABSTOP
    RTEXT           "&Lum:", COLOR_LUMACCEL, 194, 154, 20, 9
    EDITTEXT,       COLOR_LUM, 216, 152, 18, 12, WS_GROUP | WS_TABSTOP
    RTEXT           "&Red:", COLOR_REDACCEL, 243, 126, 24, 9
    EDITTEXT,       COLOR_RED, 269, 124, 18, 12, WS_GROUP | WS_TABSTOP
    RTEXT           "&Green:", COLOR_GREENACCEL, 243, 140, 24, 9
    EDITTEXT,       COLOR_GREEN, 269, 138, 18, 12, WS_GROUP | WS_TABSTOP
    RTEXT           "Bl&ue:", COLOR_BLUEACCEL, 243, 154, 24, 9
    EDITTEXT,       COLOR_BLUE, 269, 152, 18, 12, WS_GROUP | WS_TABSTOP
    PUSHBUTTON      "&Add to Custom Colors", COLOR_ADD, 152, 166, 142, 14,
                    WS_GROUP | WS_TABSTOP
END
```

Our resource file is just a copy of the original `color.dlg` file with no customizations. We stick a `windows.h` in front, and assign it a custom resource ID of 1. Let's see if we can display it.

```
#define UNICODE
#define _UNICODE
#define STRICT
#include <windows.h>
#include <commdlg.h>
int WINAPI wWinMain(
    HINSTANCE hinst, HINSTANCE hinstPrev,
    LPWSTR lpCmdLine, int nCmdShow)
{
 COLORREF rgCustColors[16] = { 0 };
 CHOOSECOLOR cc = { sizeof(cc) };
 cc.hInstance = reinterpret_cast<HWND>(hinst);
 cc.lpTemplateName = MAKEINTRESOURCE(1);
 cc.Flags = CC_ENABLETEMPLATE;
 cc.lpCustColors = rgCustColors;
 if (ChooseColor(&cc)) {
  MessageBox(nullptr, TEXT("Thank you"), TEXT("Sample"), MB_OK);
 }
 return 0;
}
```

The `hInstance` member of the `CHOOSECOLOR` structure is incorrectly declared as `HWND`, so we need to stick in a cast to keep everybody happy.

Run this program, and... everything looks perfectly normal. Good! Now we can customize it.

First, let's just add a message to the dialog.

```
1 DIALOG LOADONCALL MOVEABLE DISCARDABLE 2, 0, 298, 198
...
    LTEXT           "Don't forget to smile!",
                    -1, 4, 166, 138, 14
    DEFPUSHBUTTON   "OK", IDOK, 4, 180, 44, 14, WS_GROUP | WS_TABSTOP
    PUSHBUTTON      "Cancel", IDCANCEL, 52, 180, 44, 14, WS_GROUP | WS_TABSTOP
    PUSHBUTTON      "&Help", pshHelp, 100, 180, 44, 14, WS_GROUP | WS_TABSTOP
...
```

Rebuild the program and run it. Hey look, a happy message! Note that in order to fit the message in the dialog box, we had to make the dialog box taller and move some buttons out of the way.

Just adding static text is nice, but it's not particularly interesting. So let's add a check box to the dialog too.

```
    AUTOCHECKBOX    "I remembered to s&mile",
                    1000, 4, 166, 138, 14, WS_TABSTOP
```

In addition to remembering the color the user chose, we also want to remember whether they checked the box that says that they smiled. The documentation says that when the hook procedure receives a `WM_INITDIALOG`, the `lParam` points to the `CHOOSECOLOR` dialog. We

therefore have two options for passing extra data to the hook procedure.

- We can pass a pointer to extra data in the `lCustData` member. This is the traditional method.
- We can append our custom data to the `CHOOSECOLOR` structure. This is the technique used by the `OVERLAPPED` structure.

I'll use the traditional method for now. The `lCustData` is a pointer to a `BOOL` that receives the checkbox state on exit.

```
UINT_PTR CALLBACK CCHookProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg)
  {
  case WM_INITDIALOG:
    {
      auto pcc = reinterpret_cast<CHOOSECOLOR*>(lParam);
      auto pfSmiled = reinterpret_cast<BOOL*>(pcc->lCustData);
      SetProp(hdlg, TEXT("SmileResult"), pfSmiled);
    }
    break;
  case WM_DESTROY:
    {
      auto pfSmiled = reinterpret_cast<BOOL*>(
           GetProp(hdlg, TEXT("SmileResult")));
      if (pfSmiled) {
       *pfSmiled = IsDlgButtonChecked(hdlg, 1000);
      }
      RemoveProp(hdlg, TEXT("SmileResult"));
    }
  }
  return 0;
}
int WINAPI wWinMain(
    HINSTANCE hinst, HINSTANCE hinstPrev,
    LPWSTR lpCmdLine, int nCmdShow)
{
 COLORREF rgCustColors[16] = { 0 };
 BOOL fSmiled = FALSE;
 CHOOSECOLOR cc = { sizeof(cc) };
 cc.hInstance = reinterpret_cast<HWND>(hinst);
 cc.lpTemplateName = MAKEINTRESOURCE(1);
 cc.Flags = CC_ENABLETEMPLATE  | CC_ENABLEHOOK;
 cc.lpCustColors = rgCustColors;
 cc.lCustData = reinterpret_cast<LPARAM>(&fSmiled);
 cc.lpfnHook = CCHookProc;
 if (ChooseColor(&cc) && !fSmiled) {
  MessageBox(nullptr, TEXT("You forgot to smile."),
             TEXT("Sample"), MB_OK);
 }
 return 0;
}
```

Now, the program displays a message if you selected a color but did not check the *I remembered to smile* box.

Raymond Chen

Follow