Adding 3-D Effects to Controls

Wes Cherry, Microsoft Excel Development Group
Kyle Marsh, Microsoft Developer Network Technology Group

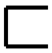
Created: September 23, 1992

Abstract

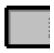
Microsoft® Windows™ version 3.*x* adds three-dimensional (3-D) support for pushbuttons, but all other controls appear two-dimensional (2-D) by default. This article discusses how an application can add 3-D effects to all controls by using the CTL3D dynamic-link library (CTL3D.DLL).

The 3-D Look

Before Microsoft® Windows™ version 3.0, pushbutton controls looked something like this:

Normal	Pushed
	

Windows version 3.0 introduced three-dimensional (3-D) pushbuttons that look like this:

Normal	Pushed
	

The 3-D effect gives the Windows interface a more sophisticated appearance. It also gives the user a clearer indication of which actions are taking place. A 3-D button that looks pressed provides more information than a two-dimensional (2-D) button that simply changes color.

Windows version 3.*x* uses 3-D for pushbuttons only; all other standard controls are in 2-D. However, many applications have started adopting 3-D for other controls as well, so the complete 3-D look is now associated with leading-edge applications and is quickly becoming a standard.

For example, Microsoft Excel version 4.0 uses 3-D for all of its controls. Microsoft Excel developers have placed all of the functionality required for 3-D controls into a dynamic-link library (DLL) called CTL3D.DLL. This DLL is now available to other applications and will help standardize the 3-D look.

Important

Future versions of Windows will implement all controls in 3-D. Therefore, any method you use to add 3-D to current controls today must either work with future versions of Windows, or must automatically disable itself when running future versions of Windows. CTL3D.DLL disables itself when running Windows version 4.0 or later. Future versions of Windows are not known at this point. If interim Windows releases (such as Windows version 3.11 or 3.20) become available without 3-D functionality, CTL3D.DLL will be updated accordingly.

What CTL3D.DLL Controls Look Like

CTL3D.DLL gives all standard controls a 3-D look, but uses different techniques for different types of controls:

- For group boxes, check boxes, option buttons (radio buttons), and static controls, CTL3D.DLL paints the entire control.
- For edit controls and list boxes, CTL3D.DLL paints 3-D effects around the control.
- For combo boxes, CTL3D.DLL combines both techniques, because combo boxes consist of a combination of other controls.

Figures 1 and 2 illustrate how CTL3D.DLL changes the appearance of these Windows controls.

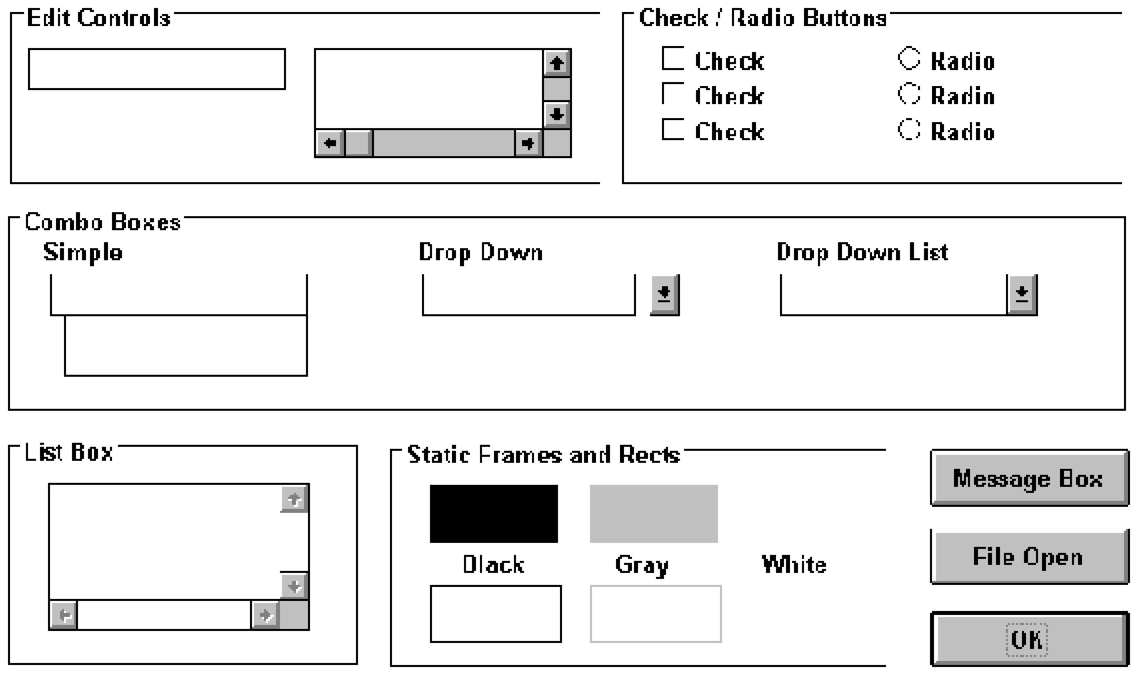


Figure 1. Controls implemented without CTL3D.DLL

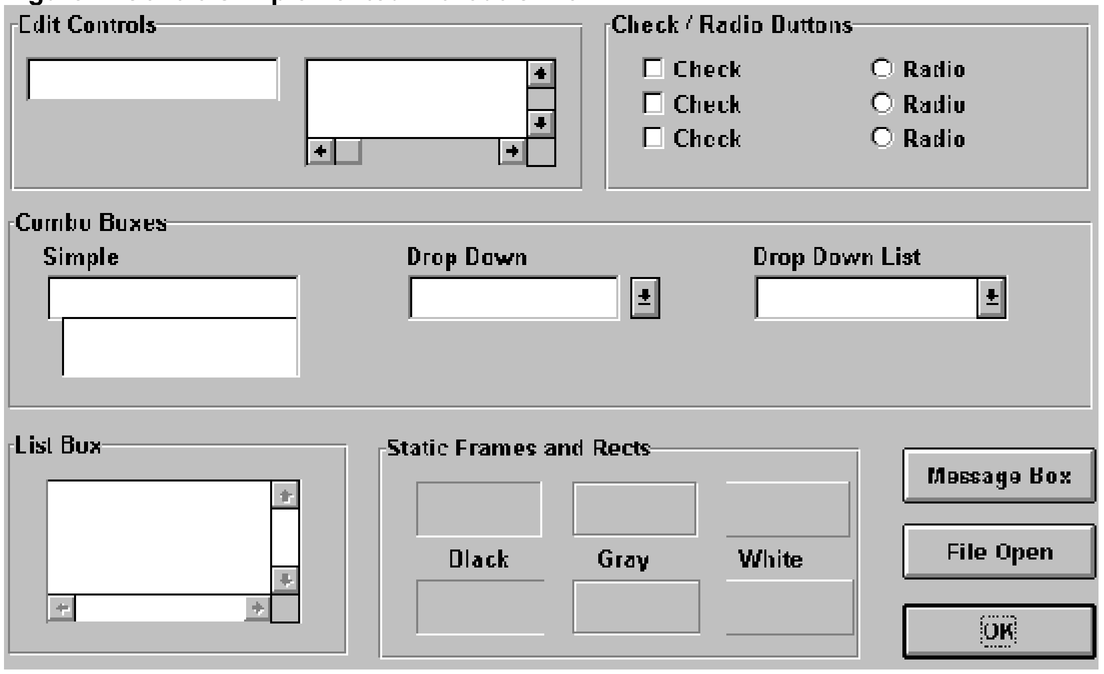
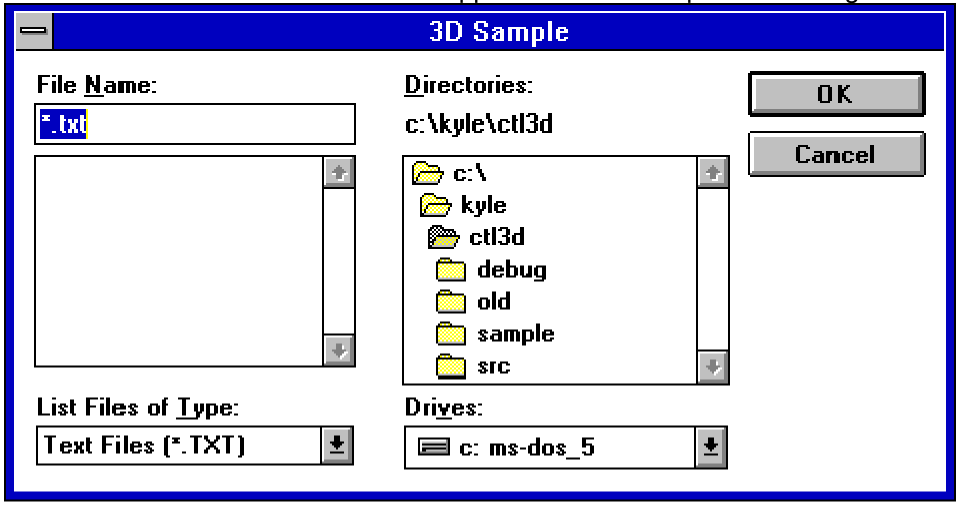


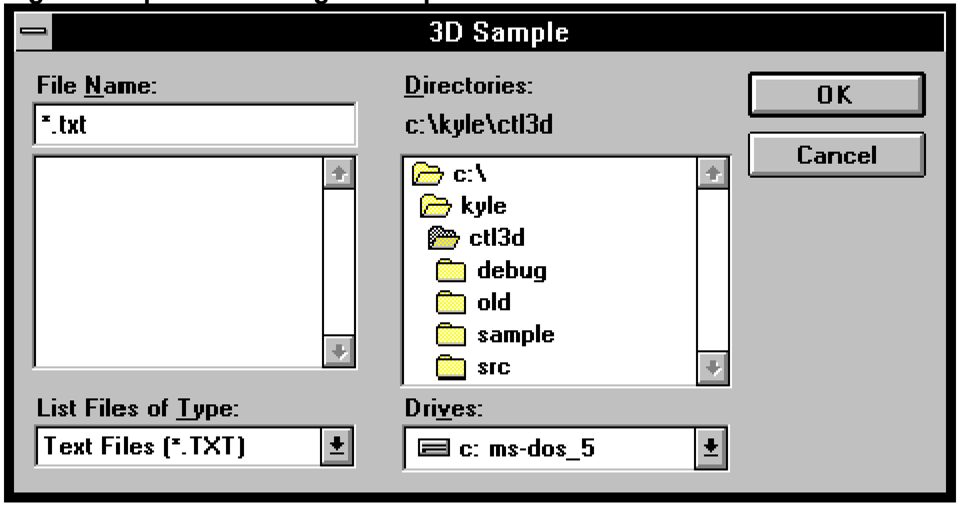
Figure 2. Controls implemented with CTL3D.DLL

CTL3D.DLL also provides a 3-D look for common dialog boxes supplied by Windows. Figures 3 and 4 illustrate how CTL3D.DLL affects the appearance of the Open File dialog box.



File Name: *.txt
Directories: c:\kyle\ctl3d
List Files of Type: Text Files (*.TXT)
Drives: c: ms-dos_5

Figure 3. Open File dialog box implemented without CTL3D.DLL



File Name: *.txt
Directories: c:\kyle\ctl3d
List Files of Type: Text Files (*.TXT)
Drives: c: ms-dos_5

Figure 4. Open File dialog box implemented with CTL3D.DLL

Using CTL3D.DLL

To use CTL3D.DLL, follow the steps listed below. See the "Function Reference" section later in this article for descriptions of functions mentioned in this section.

1. Include CTL3D.H in each source file that contains a dialog procedure or **WinMain**, or that uses a standard control.
2. Link the application with CTL3D.LIB.
3. Add a call to the **Ctl3dRegister** function when the application initializes. **WinMain** is usually a good place to add this function call.
4. Add a call to **Ctl3dColorChange** in your application's WM_SYSCOLORCHANGE message handler. **WinMain** is the best place to add this function call.
5. Add a call to **Ctl3dUnRegister** at application destroy time. Add this call at WM_DESTROY time if you subclass only dialog-box controls. If you subclass controls in your main window, you may want to call **Ctl3dUnRegister** just before returning from **WinMain**.
6. Subclass the controls. You can do this in three ways:

A. Use automatic subclassing.

If your application runs under Windows version 3.1 or later, it can use the automatic subclassing feature of CTL3D.DLL. To use this feature, call **Ctl3dAutoSubclass** after **Ctl3dRegister**. This automatically subclasses all controls in all dialog boxes, including message boxes and the Windows common dialog boxes.

Most of the common dialog boxes are handled automatically. However, because of several problems in the common dialog box libraries, you must call **Ctl3dSubclassDlg** in the hook procedures for the File Open and Page Setup dialog boxes. For example:

```
BOOL FAR PASCAL OpenHook(HWND hwnd, int wm, int wParam,
         long lParam)
{
   switch (wm) {
   case WM_INITDIALOG:
      Ctl3dSubclassDlg(hwnd, CTL3D_ALL);
      break;
 }
   return FALSE;
}

void FileOpen(HWND hwndOwner)
{
   OPENFILENAME ofn;

   ofn.lStructSize       = sizeof(OPENFILENAME);
   .
   .
   .
   ofn.lpfnHook = MakeProcInstance(OpenHook,
                                 hinstApp);
   GetOpenFileName ((LPOPENFILENAME)&ofn);
   FreeProcInstance(ofn.lpfnHook);
}
```

Automatic subclassing sets a task-specific WH_CBT hook to locate dialog boxes for the application. If your application sets a WH_CBT hook, it should do so before calling **Ctl3dAutoSubclass**. Your application's CBT hook procedure should also call **CallNextHookEx** to ensure that CTL3D.DLL's CBT hook procedure is called whenever the application's CBT hook is called.

B. Subclass each dialog box.

If your application runs under Windows version 3.0, or if you want to change only specific dialog boxes to 3-D, add a call to **Ctl3dSubclassDlg** in the WM_INITDIALOG message handler for each dialog box. You must do this before your application subclasses any controls itself:

```
case WM_INITDIALOG:
   Ctl3dSubclassDlg(hdlg, CTL3D_ALL);
break;
```

You must also call the **Ctl3dCtlColorEx** function in the WM_CTLCOLOR message handler for each dialog box, for example, by using the following code:

```
case WM_CTLCOLOR:
   return Ctl3dCtlColorEx(wm, wParam, lParam);
```

C. Subclass any control that does not appear in a dialog box.

If your application uses controls outside of dialog boxes, it must use **Ctl3dSubclassCtl** to subclass these controls individually. When an application calls **Ctl3dCtlColorEx** from a window procedure instead of a dialog procedure, it should use the following code:

```
case WM_CTLCOLOR:
   hbrush = Ctl3dCtlColorEx(wm, wParam, lParam)
   if (hbrush != (HBRUSH) fFalse)
      return hbrush;
   else
      return DefWindowProc(hwnd, wm, wParam, lParam);
```

Considerations

■ CTL3D.DLL creates 3-D effects for list boxes and edit controls outside the control's client area. For this reason, the controls appear larger than specified in the dialog definition or **CreateWindow** for the control. The application designer may need to tweak the size of controls to account for this increase.

■ CTL3D.DLL does not support the BS_LEFTTEXT style for check boxes, option buttons, and group boxes.

■ CTL3D.DLL handles static frame controls differently than Windows does. In fact, CTL3D.DLL implements its own static rectangles and frames:

■ Static controls with the SS_BLACKRECT or SS_BLACKFRAME style are drawn as inset 3-D rectangles.

■ Static controls with the SS_GRAYRECT or SS_GRAYFRAME style are drawn as bas-relief 3-D rectangles.

■ Static controls with the SS_WHITERECT or SS_WHITEFRAME style are drawn as outset 3-D rectangles.

To stop CTL3D.DLL from using its own static controls, do not pass the CTL3D_STATICFRAMES flag to the **Ctl3dSubclassDlg** function. (See the "Function Reference" section later in this article for a description of **Ctl3dSubclassDlg**.)

■ If static controls are not disabled (grayed), there is no need to pass CTL_3DSTATICTEXTS to the **Ctl3dSubclassDlg** function.

■ For future compatibility, an application should not rely on subtle changes in the way CTL3D.DLL draws controls.

Using CTL3D.DLL with Microsoft Foundation Classes (MFC)*

Using CTL3D.DLL in an MFC (C/C++ version 7.0) application requires several additional steps. To add 3-D controls to an MFC dialog box, you must:

1. Call **Ctl3dRegister** in the application's **CWinApp::InitInstance** function.

```
Ctl3dRegister(AfxGetInstanceHandle())
```

2. Add an **OnCtlColor** member function to the application's dialog class, and add ON_WM_CTLCOLOR to its message map. The **OnCtlColor** member function should call the base class **OnCtlColor** first, to color the text correctly, then it should call **Ctl3dCtlColor**:

```
HBRUSH CAboutBox::OnCtlColor( CDC*  pDC, CWnd*  pWnd,
                                        UINT nCtlColor  )
 {
    CModalDialog::OnCtlColor(pDC, pWnd, nCtlColor);
    return Ctl3dCtlColor((HDC)pDC,
                  (LONG)MAKELONG((HWND)pWnd, nCtlColor));
 }
```

3. In the application's **OnInitDialog** function for the dialog class, call **Ctl3dSubclassDlg**:

```
 BOOL CAboutBox::OnInitDialog()
{
   Ctl3dSubclassDlg(m_hWnd, CTL3D_ALL);
   return TRUE;
}
```

4. Add an **OnSysColorChange** member to the application's CWinApp class, and add ON_WM_SYSCOLORCHANGE to its message map. Call **Ctl3dColorChange** from **OnSysColorChange**:

```
void CMainWindow::OnSysColorChange()
   {
    Ctl3dColorChange();
   }
```

5. Add **ExitInstance** to the application's CWinApp class, and call **Ctl3dUnRegister** from this function:

```
int CTheApp::ExitInstance()
{
   Ctl3dUnRegister(AfxGetInstanceHandle());
   return CWinApp::ExitInstance();
}
```

6. Link with CTL3D.LIB.

Installing CTL3D.DLL with Your Application

An application's installation program should install CTL3D.DLL in the Windows SYSTEM directory, not in the application's own directory. CTL3D.DLL includes standard version control stamping to help the application ensure that it does not overwrite a later version of the DLL during the installation process.

Bugs

When an application that uses CTL3D.DLL terminates under the debugging version of Windows version 3.1, Windows displays a warning about a brush and a bitmap not being freed. Ignore this warning; the brush and bitmap are freed in CTL3D.DLL's WEP.

Important

The Microsoft Developer Network CD contains source code for CTL3D.DLL. **Please do not make any changes to this code or release private versions of CTL3D.DLL.** Any changes you make to CTL3D.DLL may cause problems for other applications that use this code, and will make you very unpopular. If you need a special version of the DLL, rename CTL3D.DLL and change the NAME= statement in the DEF file to avoid conflicts.

Function Reference

Ctl3dRegister

```
BOOL Ctl3dRegister(HANDLE hinstApp)
```

The **Ctl3dRegister** function registers an application as a client of CTL3D.DLL. An application that uses CTL3D.DLL should call this function in **WinMain**. **Ctl3dRegister** returns TRUE if 3-D effects are active, and FALSE if they are not. 3-D effects are not available under Windows version 4.0 or on computers that have less than VGA resolution.

Ctl3dUnRegister

```
BOOL Ctl3dUnRegister(HANDLE hinstApp)
```

An application calls the **Ctl3dUnRegister** function (usually in the application's **WinMain** function) to stop using CTL3D.DLL. **Ctl3dUnRegister** returns TRUE if no controls are currently using CTL3D.DLL; otherwise, it returns FALSE.

Ctl3dAutoSubclass

```
PUBLIC BOOL FAR PASCAL Ctl3dAutoSubclass(HANDLE hinstApp)
```

Ctl3dAutoSubclass automatically subclasses and adds 3-D effects to all dialog boxes in the application. (As mentioned previously, an application must still call **Ctl3dSubclassDlg** for the Open File and Page Setup common dialog boxes.) **Ctl3dAutoSubclass** returns FALSE if CTL3D.DLL:

- Is running under Windows version 3.0 or earlier.
- Does not have space available in its tables for the current application. CTL3D.DLL can service up to 32 applications at the same time.
- Cannot install its CBT hook.

Otherwise, the function returns TRUE.

Ctl3dGetVer

```
WORD Ctl3dGetVer(void)
```

Ctl3dGetVer indicates the version of CTL3D.DLL that is currently running. It returns a value that contains the major version number in the high-order byte and the minor version number in the low-order byte.

Ctl3dEnabled

```
BOOL Ctl3dEnabled(void)
```

Ctl3dEnabled checks to see whether controls can use 3-D effects. The function returns TRUE if they can, or FALSE if they cannot. **Ctl3dEnabled** and **Ctl3dRegister** return FALSE in Windows version 4.0 or later.

Ctl3dSubclassCtl

```
BOOL Ctl3dSubclassCtl(HWND hwnd)
```

Ctl3dSubclassCtl subclasses an individual control. This function is used only for controls that do not appear in dialog boxes. **Ctl3dSubclassCtl** returns TRUE if the control is successfully subclassed, or FALSE if it is not.

Ctl3dSubclassDlg

```
PUBLIC BOOL FAR PASCAL Ctl3dSubclassDlg(HWND hwndDlg, WORD grbit)
```

Ctl3dSubclassDlg subclasses all controls in a dialog box. An application must call this function in the

WM_INITDIALOG message handler. The *grbit* parameter determines the specific control types to be subclassed. The CTL3D.H file defines the following values:

- CTL3D_BUTTONS—Subclass buttons.
- CTL3D_LISTBOXES—Subclass list boxes.
- CTL3D_EDITS—Subclass edit controls.
- CTL3D_COMBOS—Subclass combo boxes.
- CTL3D_STATICTEXTS—Subclass static text controls.
- CTL3D_STATICFRAMES—Subclass static frames.
- CTL3D_ALL—Subclass all controls.

Ctl3dCtlColor

Ctl3dCtlColor is provided for compatibility with previous versions of CTL3D.DLL. It should not be used in applications that started implementing 3-D with the current version of CTL3D.DLL.

Ctl3dCtlColorEx

```
HBRUSH Ctl3dCtlColorEx(UINT message, WPARAM wParam, LPARAM lParam)
```

Ctl3dCtlColorEx handles the WM_CTLCOLOR message for applications that use CTL3D.DLL. The function returns a handle to the appropriate brush or (HBRUSH)(0) if an error occurred. The following code fragment illustrates the use of **Ctl3dCtlColorEx** from a dialog procedure:

```
case WM_CTLCOLOR:
   return Ctl3dCtlColorEx(wm, wParam, lParam);
```

The following code fragment illustrates the use of **Ctl3dCtlColorEx** from a window procedure:

```
case WM_CTLCOLOR:
   hbrush = Ctl3dCtlColorEx(wm, wParam, lParam)
   if (hbrush != (HBRUSH) fFalse)
      return hbrush;
   else
      return DefWindowProc(hwnd, wm, wParam, lParam);
```

Ctl3dColorChange

```
BOOL Ctl3dColorChange(VOID)
```

Ctl3dColorChange handles system color changes for applications that use CTL3D.DLL. This function should be called in the application's main window procedure for the WM_SYSCOLORCHANGE message; for example:

```
case WM_SYSCOLORCHANGE:
   Ctl3dColorChange();
break;
```

Ctl3dColorChange returns TRUE if it is successful; otherwise, it returns FALSE.

---

* Thanks to Mark Bader for providing the information in this section.