Adding 3-D Effects to Controls

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

Created: September 23, 1992

Revised: February 1993
Added the **Ctl3dDlgFramePaint** function for 3-D dialog frames.
Added WM_DLGBORDER and WM_DLGSUBCLASS messages.
Added section on using CTL3D with Pascal.
Added section on using CTL3D with Visual Basic™.
Color table is now correctly cleared during **Ctl3dUnregister**.
CTL3D now allows up to 512 characters (used to be 256) in **MessageBox**.
Group box text updates now work correctly when the new text is shorter than the old text.
CTL3D.DLL's **WEP** function is now in a fixed, preloaded code segment.
Added Windows NT™ support. Note that CTL3D32.DLL does not support combo boxes and list boxes at this time. This will be fixed in a later release.

Revised: May 1993
Updated CTL3D to work with Microsoft Foundation Class Library.
Added **Ctl3dSubclassDlgEx** function to subclass all the controls on a dialog box and the dialog box itself.
Common dialog hook functions are no longer required.
All controls on common dialog boxes are handled for Windows NT.
Added CTL3D_NODLGWINDOW constant for **Ctl3dSubclassDlgEx**.

Revised: June 1993
Added CTL3DS.LIB versions of library.

Revised August 1993
Fixed documentation error: CTL3D does subclass static controls with SS_NOPREFIX
Fixed documentation error: Ctl3dUnRegister should be Ctl3dUnregister.
Added CTL3DD.lib versions of library.
Added to section on how to install CTL3D and use static linked versions.
Fixed Disabled text color selection.

Click to open or copy the files in the CTL3D sample application for this technical article.
1406
tech\winman\ctl3d

There are a number of ways to use CTL3D. As a dynamic linked library, or a static linked library for both EXEs and DLLs. Please read the section "Installing CTL3D with Your Application" below for details on how to select the version that is best for your application.

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 describes how an application can add 3-D effects to all controls by using the CTL3D   library..

The 3-D Look

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

Normal
Pushed
OK
OK

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

Normal
Pushed
OK
OK

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 library called CTL3D. This library 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 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 will be updated accordingly.

What CTL3D Controls Look Like

CTL3D 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 paints the entire control.

For edit controls and list boxes, CTL3D paints 3-D effects around the control.

For combo boxes, CTL3D combines both techniques, because combo boxes consist of a combination of other controls.

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

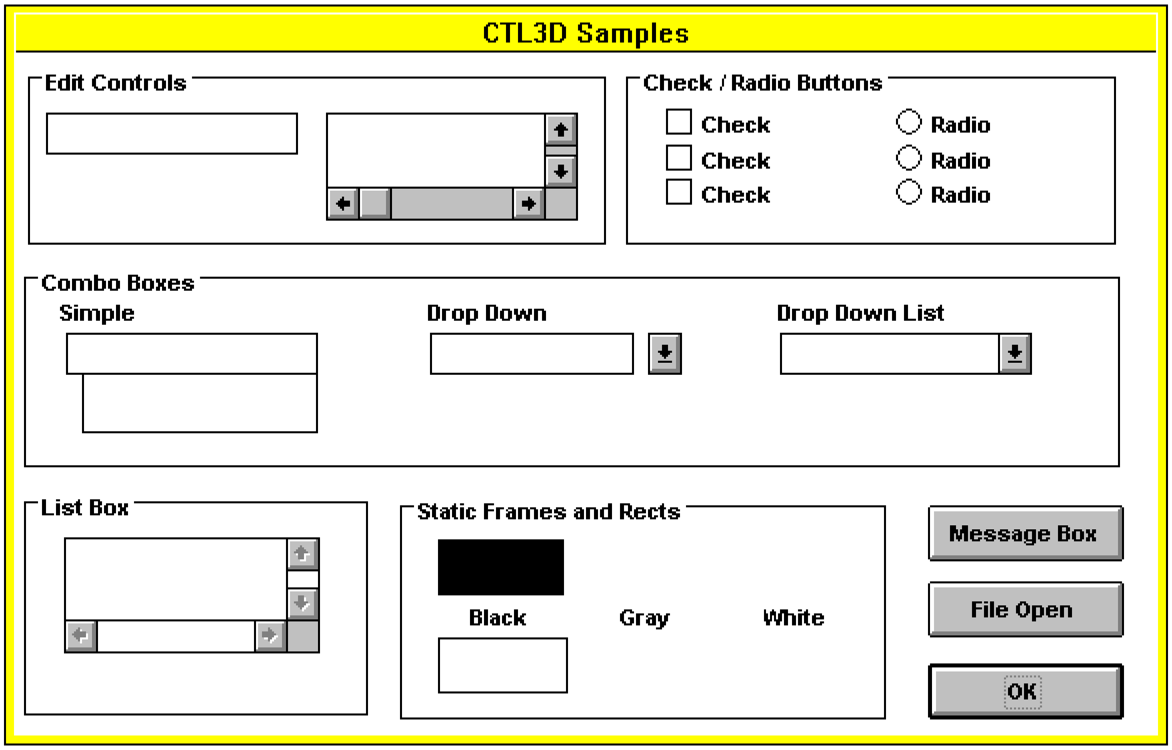


Figure 1. Controls implemented without CTL3D

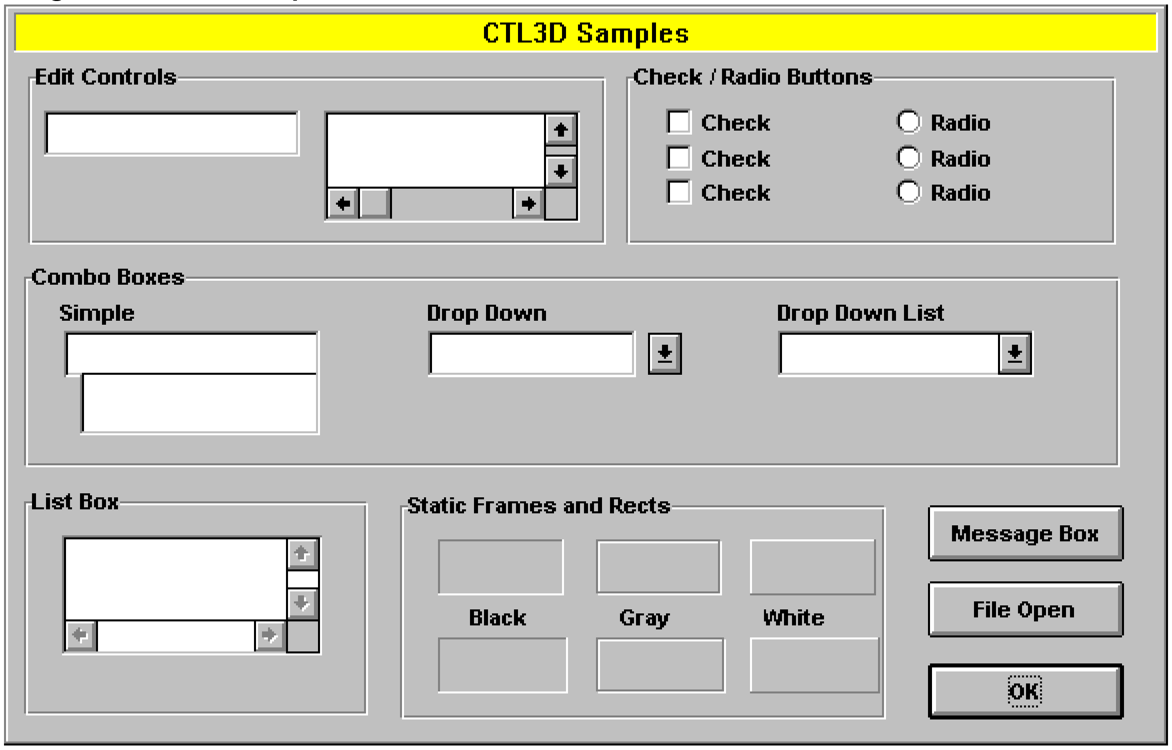
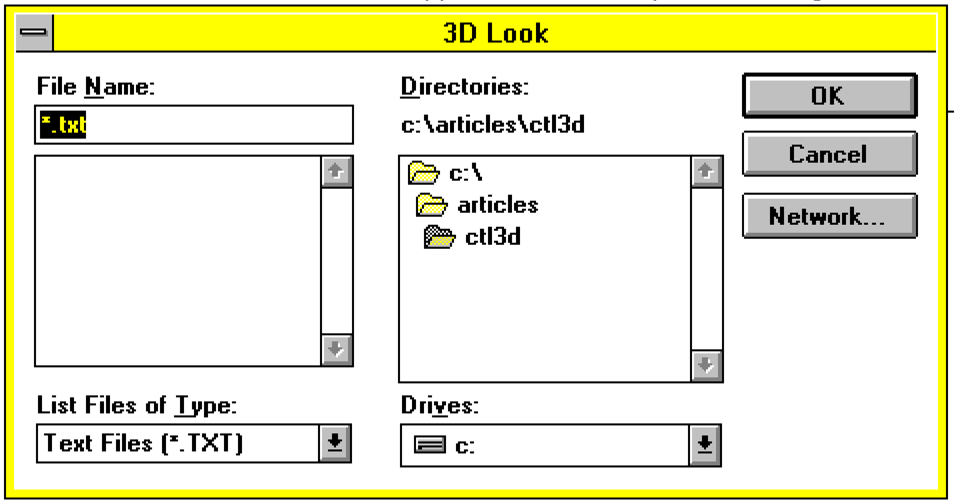


Figure 2. Controls implemented with CTL3D

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



3D Look

File Name: *.txt

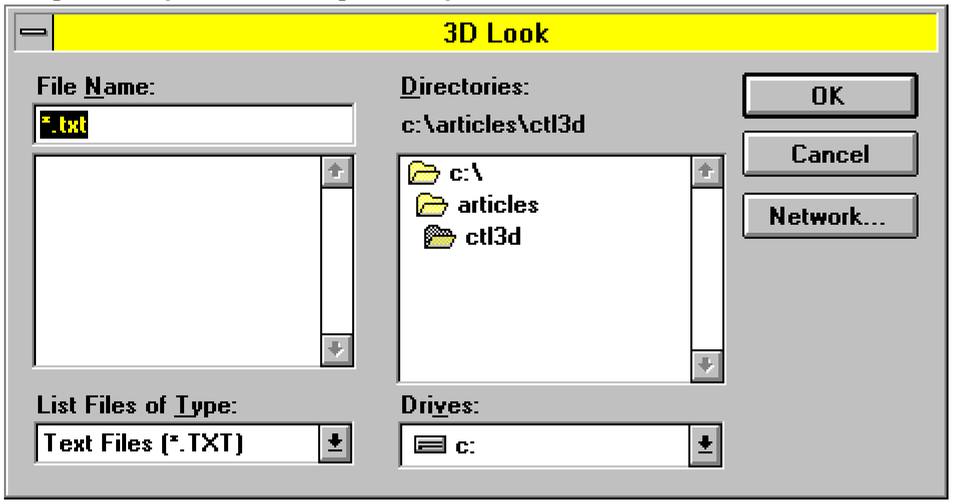
Directories: c:\articles\ctl3d

c:\
articles
ctl3d

List Files of Type: Text Files (*.TXT)

Drives: c:

Figure 3. Open File dialog box implemented without CTL3D



3D Look

File Name: *.txt

Directories: c:\articles\ctl3d

c:\
articles
ctl3d

List Files of Type: Text Files (*.TXT)

Drives: c:

Figure 4. Open File dialog box implemented with CTL3D

Using CTL3D from C

To use CTL3D, 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 appropriate CTL3D library. There are different ways to link CTL3D. Read

the section on Installing CTL3D with your application below for a discussion on which CTL3D library you should use.

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. 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.

      Earlier versions of CTL3D required hook procedures for some of the common dialog boxes. This is no longer required.

      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'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** or **Ctl3dSubclassDlgEx** 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;
      ```

      Or:

      ```
      case WM_INITDIALOG:
         Ctl3dSubclassDlgEx(hdlg, CTL3D_ALL);
      break;
      ```

      If your application uses **Ctl3dSubclassDlg**, the following steps are required. If your application uses **Ctl3dSubclassDlgEx**, these steps are not required.

      The application 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);
      ```

      For applications running on Windows NT, the application must call **Ctl3dCtlColorEx** for the

WM_CTLCOLORBTN, WM_CTLCOLORDLG, WM_CTLCOLOREDIT, WM_CTLCOLORLISTBOX, WM_CTLCOLORMSGBOX, WM_CTLCOLORSCROLLBAR, and WM_CTLCOLORSTATIC messages. For example:

```
case WM_CTLCOLORBTN:
case WM_CTLCOLORDLG:
case WM_CTLCOLOREDIT:
case WM_CTLCOLORLISTBOX:
case WM_CTLCOLORMSGBOX:
case WM_CTLCOLORSCROLLBAR:
case WM_CTLCOLORSTATIC:
   return Ctl3dCtlColorEx(wm, wParam, lParam);
```

If you want CTL3D to draw a 3-D border for the dialog box, add the following call to **Ctl3dDlgFramePaint** in the WM_SETTEXT, WM_NCPAINT, and WM_NCACTIVATE message handlers for the dialog box (the dialog box must have the WS_DLGFRAME and DS_MODALFRAME styles):

```
case WM_SETTEXT
case WM_NCPAINT:
case WM_NCACTIVATE:
   SetWindowLong(hdlg, DWL_MSGRESULT,
                  Ctl3dDlgFramePaint(hdlg, wm, wParam, Param);
return TRUE;
```

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:

```
#ifdef WIN32
case WM_CTLCOLORBTN:
case WM_CTLCOLORDLG:
case WM_CTLCOLOREDIT:
case WM_CTLCOLORLISTBOX:
case WM_CTLCOLORMSGBOX:
case WM_CTLCOLORSCROLLBAR:
case WM_CTLCOLORSTATIC:
#else
case WM_CTLCOLOR:
#endif
   hbrush = Ctl3dCtlColorEx(wm, wParam, lParam)
   if (hbrush != (HBRUSH) fFalse)
      return hbrush;
   else
      return DefWindowProc(hwnd, wm, wParam, lParam);
```

Considerations

Ctl3dAutoSubClass only subclasses windows that have the standard WC_DIALOG dialog window class.

CTL3D adds 3-D effects only to the following controls:

Standard Windows controls (list box, edit control, static control, button, combo box).

Controls that are not superclassed.

Non-dialog box controls (edit controls and list boxes) whose parents do not have the WS_CLIPCHILDREN style.

CTL3D does not subclass buttons that have the BS_OWNERDRAW or BS_LEFTTEXT style.

CTL3D does not subclass static controls that have the SS_ICON, SS_LEFT, SS_CENTER, SS_LEFTNOWORDWRAP, SS_SIMPLE style.

CTL3D 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 handles static frame controls differently than Windows does. In fact, CTL3D 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 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), you need not pass CTL_3DSTATICTEXTS to the **Ctl3dSubclassDlg** or **Ctl3dSubclassDlgEx** function.

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

CTL3D draws a 3-D border for dialogs that have the WS_DLGFRAME and DS_MODALFRAME styles. To disable this feature, see the WM_DLGBORDER message in the "Messages" section later in this article.

When using automatic subclassing, CTL3D sends a WM_CTLCOLOR message to the application's dialog boxes. The application can respond to the message and return a handle to a brush, or it can return FALSE and have CTL3D respond to the message. Dialog procedures usually return FALSE when they do not handle a particular message, so additional programming is necessary only if the application wants to handle WM_CTLCOLOR messages for a particular dialog box.

Using CTL3D with the Microsoft Foundation Class Library

Using CTL3D in a Microsoft Foundation Class Library (C/C++ version 7.0 or Visual C++™ version 1.0) application requires additional steps. To add 3-D controls to a Foundation class dialog box, you must:

1. Call **Ctl3dRegister** and **Ctl3dAutoSubclass** in the application's **CWinApp::InitInstance** function:

   ```
   Ctl3dRegister(AfxGetInstanceHandle());
   Ctl3dAutoSubclass(AfxGetInstanceHandle());
   ```

   You can also remove, or comment out, the call to **SetDialogBkColor** from the **InitInstance** function. CTL3D and **SetDialogBkColor** can coexist, but they perform similar functions (so it's overkill to have both).

2. Add an **OnSysColorChange** member to the application's main frame window, and add ON_WM_SYSCOLORCHANGE to its message map. Call **Ctl3dColorChange** from **OnSysColorChange**:

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

3. Add **ExitInstance** to the application's CWinApp class, and call **Ctl3dUnregister** from this function:

```
int CTheApp::ExitInstance()
{
   Ctl3dUnregister(AfxGetInstanceHandle());
   return CWinApp::ExitInstance();
}
```

4. Link the application with appropriate CTL3D library. There are different ways to link CTL3D. Read the section on Installing CTL3D with your application below for a discussion on which CTL3D library you should use.

We suggest using the automatic subclassing of CTL3D with the Foundation classes. It simplifies the effort required to add 3-D effects to an application, and it is easily removed when the application is updated to versions of Windows that no longer benefit from CTL3D.

If automatic subclassing does not suit your application, our second suggestion is to use the **Ctl3dSubclassDlgEx** function for each dialog box to which you want to add 3-D effects. Do steps 1 thorugh 4 above, but leave out the call to Ctl3dAutoSubclass. Then in the application's **OnInitDialog** function for the dialog class, call **Ctl3dSubclassDlgEx**:

```
 BOOL CAboutBox::OnInitDialog()
{
   Ctl3dSubclassDlgEx(m_hWnd, CTL3D_ALL);
   return TRUE;
}
```

If you are using automatic subclassing and a FormView class from the Foundation classes, and you want to add 3-D effects for the form, you must call **Ctl3dSubclassDlg** from the class's **OnInitialUpdate** function:

```
 VOID CCheckView::OnInitialUpdate()
{
   Ctl3dSubclassDlg(m_hWnd, CTL3D_ALL);
   return TRUE;
}
```

Applications that are not using automatic subclassing and want a FormView to have 3-D effects should call **Ctl3dSubclassDlgEx** from the **OnInitialUpdate** function:

```
 VOID CCheckView::OnInitialUpdate()
{
   Ctl3dSubclassDlgEx(m_hWnd, CTL3D_ALL);
   return TRUE;
}
```

Do not use CTL3D.DLL with the DLL version of the Foundation Class Library.

There is a possible conflict between these DLLs that does not occur with the statically linked Foundation Class Library. If you must use a DLL version of MFC then you must statically link in the DLL static link version of CTL3D, see "Installing CTL3D with Your Application" below, with the MFC DLL.

Using CTL3D with Pascal

To use CTL3D.DLL from Turbo Pascal® for Windows, compile the CTL3D.PAS unit. This allows you to use the CTL3D functions as you would in C. A port of the SAMPLE3D.C file to Pascal is available in the SAMPLE3D.PAS file.

Using CTL3D with Visual Basic

You can use CTL3D.DLL to add 3-D effects to the common dialog boxes and message boxes for a Visual Basic™ application. CTL3D.DLL does not currently work for the controls on a Visual Basic form. To use CTL3D.DLL to enhance the common dialog boxes and message boxes from Visual Basic, follow the steps below.

To the form's general area, add:

```
Declare Function GetModuleHandle Lib "Kernel"
       (ByVal ModuleName As String) As Integer
Declare Function Ctl3dAutoSubclass Lib "Ctl3D.DLL"
       (ByVal hInst As Integer) As Integer
Declare Function Ctl3dRegister Lib "Ctl3D.DLL"
       (ByVal hInst As Integer) As Integer
Declare Function Ctl3dUnregister Lib "Ctl3D.DLL"
       (ByVal hInst As Integer) As Integer
```

To the form's load procedure, add:

```
Sub Form_Load ()

Inst% = GetModuleHandle("Your app name here")
ret = Ctl3dRegister(Inst%)
ret = Ctl3dAutoSubclass(Inst%)

End Sub
```

To the form's unload procedure, add:

```
Sub Form_Unload (Cancel As Integer)

Inst% = GetModuleHandle("test.exe")
ret = Ctl3dUnregister(Inst%)

End Sub
```

Installing CTL3D with Your Application

This has been the area which has caused the most problems for application using CTL3D. For this reason we are adding new ways to used CTL3D.

CTL3D.DLL - The Original

The original way to use CTL3D was to link your application to CTL3D.LIB and use the CTL3D.DLL dynamic link library.

Important

An application's installation program must install CTL3D.DLL in the Windows SYSTEM directory, or if running on a networked Windows installation the Windows directory. CTL3D.DLL must not be installed in the application's own directory or any other directory.

Each application has to install CTL3D.DLL to the windows\system directory. For networked installations, where the application install program is not able to write to the windows\system directory, it is alright to put CTL3D.DLL in the windows directory. CTL3D.DLL must not be installed, or left, in any other directory. The reason for this is that when an application needs to update CTL3D.DLL to a newer version, it must know where the old one is located. Applications that required new features would not work correctly when an old version that had been left in some other directory was loaded instead of the correct version. Another requirement is that when an application installs CTL3D.DLL it must always do a version check to make sure it does not over write a newer version with an older one. If an application fails to do a version check it can cause other applications to work incorrectly or even crash.

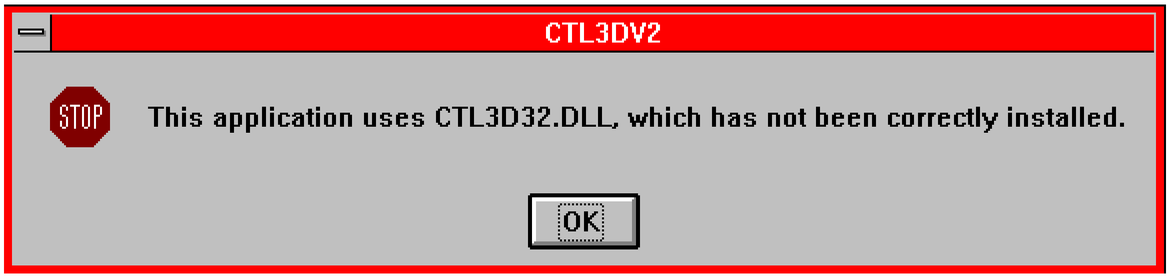
CTL3DV2.DLL - A new Version the CTL3D

Since there are a number of released applications that either don't do version checking or install CTL3D.DLL to the wrong directory, we have created a new version of CTL3D, CTL3DV2.DLLwhich applications link to with the CTL3DV2.LIB file. This new DLL name makes it impossible for older versions of CTL3D.DLL (version 1.x) to effect applications using CTL3DV2.DLL.

Important

An application's installation program must install CTL3DV2.DLL in the Windows SYSTEM directory, or if running on a networked Windows installation the Windows directory. CTL3DV2.DLL must not be installed in the application's own directory or any other directory.

CTL3DV2.DLL will not produce 3D effects unless it is running from either windows\system or the windows directory. This hopefully will encourage developers to correctly install CTL3DV2.DLL with their applications. When CTL3DV2.DLL is run from any other directory it will display the following message box:


CTL3DV2

This application uses CTL3D32.DLL, which has not been correctly installed.

CTL3DS.LIB (CTL3DD.LIB) - Staically linked CTL3D

Even with the new version of CTL3D, CTL3DV2, it is possible that at some time in the future applications that need a newer version of CTL3DV2 will be broken by another application installing an older version because the other application did not do any version checking. To avoid any possible conflicts over the different versions of CTL3D, an appplication can use the static linked version. To use this version applications need to link to either the CTL3DS.LIB file, if the application is an EXE file, or the CTL3DD.LIB file, if the applicaiton is a DLL.

To use the static link version of CTL3D you must:

1) Include ctl3d.h in the applications resource file:

```
#include "ctl3d.h"
```

2) Include a BITMAP statement in the application's (or dll's) resource file for CTL3D's bitmap, 3DCHECK.BMP (this file is included with the samples):

```
CTL3D_3DCHECK BITMAP "3dcheck.bmp"
```

3) For DLLs, merge the CTL3D.DEF file into your dll's .DEF file. You can change the ordinals for CTL3D's exported functions to any values that work for your application since only your application will be using these functions from your DLL.

4) Link the application with CTL3DS.LIB (for EXEs) or CTL3DD.OBJ (for DLLs). If you are building a 16-bit DLL and _hModule is an undefined external you must add a global variable, (HINSTANCE _hModule) to your DLL and initialize it to the hInst parameter passed to LibMain. Microsoft C/C++ 7.0 and Visual C++ 1.0 create this varaible automatically in their default LIBENTRY routines. If you DLL does not produce 3D effects try initializing this variable in you LibMain funciton.

NOTE: The static link version of CTL3D must run on Windows 3.1 or later.

CTL3D32 - 32 Bit CTL3D

CTL3D32.DLL is the first release of CTL3D for Win32. It includes the functionality of CTL3DV2, in other words it does the check to ensure it is running from the correct directory. Applications link to the CTL3D32.LIB library to use CTL3D32.DLL. There are two variations of CTL3D32.DLL, one for Windows NT and one for Win32s. The Win32s version is not UNICODE, which Win32s does not support, while the Windows NT version is UNICODE. There are also CTL3D32S.LIB and CTL3D32D.OBJ static linked versions.

Messages

WM_DLGBORDER

```
 wParam = 0;
 lParam = (int FAR *)lpDraw3d;
```

CTL3D sends this message to a dialog box just before it draws a 3-D border for the dialog box. If the dialog box sets *lpDraw3d* to CTL3D_NOBORDER, CTL3D will not draw the 3-D border. Use the WM_DLGBORDER message to disable the 3-D frame on selected dialog boxes when using automatic subclassing (**Ctl3dAutoSubClass**); for example:

```
case WM_DLGBORDER:
      //
      // Don't draw the 3-D frame.
      //
      *(int FAR*)(lParam) = CTL3D_NOBORDER;
break;
```

WM_DLGSUBCLASS

```
wParam = 0;
lParam = (int FAR *)lpSubClass;
```

CTL3D sends this message to a dialog box just before it automatically subclasses the dialog box. If the dialog box sets *lpSubClass* to CTL3D_NOSUBCLASS, CTL3D will not subclass the dialog box. Use the WM_DLGSUBCLASS message to disable automatic subclassing for selected dialog boxes; for example:

```
case WM_DLGSUBCLASS:
      //
      // Don't subclass this dialog.
      //
      *(int FAR*)(lParam) = CTL3D_NOSUBCLASS;
break;
```

Function Reference

Ctl3dRegister

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

The **Ctl3dRegister** function registers an application as a client of CTL3D. An application that uses CTL3D 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. **Ctl3dUnregister** returns TRUE if no controls are currently using CTL3D; 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. **Ctl3dAutoSubclass** returns FALSE if CTL3D:

Is running under Windows version 3.0 or earlier.

Does not have space available in its tables for the current application. CTL3D can service up to 32 applications at the same time.

Cannot install its CBT hook.

Otherwise, the function returns TRUE.

Ctl3dDlgFramePaint

```
LONG Ctl3dDlgFramePaint(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
```

Ctl3dDlgFramePaint handles the WM_NCACTIVATE and WM_NCPAINT messages for dialog boxes that want a 3-D border. For example:

```
case WM_NCPAINT:
case WM_NCACTIVATE:
     return Ctl3dDlgFramePaint(wm, wParam, lParam);
```

Ctl3dDlgFramePaint calls **DefWindowProc** and returns the value returned by **DefWindowProc**.

Ctl3dGetVer

```
WORD Ctl3dGetVer(void)
```

Ctl3dGetVer indicates the version of CTL3D 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_BUTTONSSubclass buttons.

CTL3D_LISTBOXESSubclass list boxes.

CTL3D_EDITSSubclass edit controls.

CTL3D_COMBOSSubclass combo boxes.

CTL3D_STATICTEXTSSubclass static text controls.

CTL3D_STATICFRAMESSubclass static frames.

CTL3D_ALLSubclass all controls.

Ctl3dSubclassDlgEx

```
PUBLIC BOOL FAR PASCAL Ctl3dSubclassDlgEx(HWND hwndDlg, DWORD grbit)
```

Ctl3dSubclassDlgEx subclasses all controls in a dialog box and the dialog window itself. An application must call this function in the WM_INITDIALOG message handler. **Ctl3dSubclassDlgEx** makes using CTL3D much easier when not using automatic subclassing. Since the function subclasses the dialog box as well as the dialog's controls, you don't need to add any code to the dialog procedure in the application. This function is especially useful in applications based on C++. The *grbit* parameter determines the specific control types to be subclassed. The CTL3D.H file defines the following values:

CTL3D_BUTTONSSubclass buttons.

CTL3D_LISTBOXESSubclass list boxes.

CTL3D_EDITSSubclass edit controls.

CTL3D_COMBOSSubclass combo boxes.

CTL3D_STATICTEXTSSubclass static text controls.

CTL3D_STATICFRAMESSubclass static frames.

CTL3D_ALLSubclass all controls.

CTL3D_NODLGWINDOWDon't subclass the dialog window.

Ctl3dCtlColor

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

Ctl3dCtlColorEx

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

Ctl3dCtlColorEx handles the WM_CTLCOLOR message for applications that use CTL3D. 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. 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.

How CTL3D Works

CTL3D uses subclassing to do its magicit subclasses each control to add 3-D functionality to the painting of the control. In some cases, CTL3D also adds functionality to other operations of the control. Let's look at what CTL3D does for each control type:

List BoxesCTL3D draws a 3-D frame around the list box. It does not add 3-D effects to the top of the list box if the list box is part of a simple combo box. CTL3D handles the WM_PAINT, WM_SHOWWINDOW, WM_WINDOWPOSCHANGE, and WM_NCCALCSIZE messages.

Edit ControlsCTL3D draws a 3-D frame around the edit control. It does not add 3-D effects to the bottom of the edit control if the control is part of a simple combo box. CTL3D handles the WM_PAINT, WM_SHOWWINDOW, and WM_WINDOWPOSCHANGE messages.

Combo BoxesCTL3D draws 3-D effects around each part (edit box and list box) of a combo box. To draw these effects, it uses the same routine that it uses for the individual controls.

ButtonsCTL3D completely replaces the standard button-drawing routines. To do this, it handles the WM_SETTEXT, WM_KILLFOCUS, WM_ENABLE, WM_SETFOCUS, WM_PAINT, BM_SETSTATE, and BM_SETCHECK messages. CTL3D does not subclass buttons that have the BS_OWNERDRAW or BS_LEFTTEXT style.

Static ControlsCTL3D replaces the standard painting routines for static controls. CTL3D handles the WM_PAINT and WM_ENABLE messages. CTL3D will not subclass static controls that have the SS_ICON, SS_LEFT, SS_CENTER, SS_LEFTNOWORDWRAP, SS_SIMPLE, or SS_NOPREFIX style.

How CTL3D Does Its Subclassing

Earlier versions of CTL3D did not accept subclassed controls. CTL3D did not store the original window procedure because, for a given control or dialog box, the next window procedure to call was the standard procedure for the window. This also ensured no performance penalty for looking up the original procedure from a table. This practice caused a conflict between the Foundation classes and CTL3D since the Foundation classes use a similar concept in subclassing. To resolve this conflict, CTL3D was changed to subclass windows that had been previously subclassed. CTL3D now:

Uses the window's class name to determine if CTL3D can subclass the control. Previously CTL3D used the window's procedure address for this.

Stores the window's original procedure as properties of the window. This is easy to do and does not affect performance since the lookup on the property is immediate.

Adds two global atoms, one for Windows NT, for adding, getting, and deleting the window properties. This is a major reason that the use of properties does not affect performance. Global atoms reduce the overhead of properties.

Checks the window properties before subclassing a window to ensure that CTL3D has not subclassed the control before.

Removes the properties when Windows sends a WM_NCDESTROY message to the window. Windows sends this message just before deleting the window, and it is the last chance to remove window properties.

Here are the subclass and cleanup routines for CTL3D:

```
PRIVATE VOID SubclassWindow(HWND hwnd, FARPROC lpfnSubclassProc)
   {
            FARPROC lpfnWndProc;

            // Is this already subclassed by CTL3D?
#ifdef WIN32
            if ( GetProp(hwnd,(LPCSTR)aCtl3d) == NULL )
#else
            if ( GetProp(hwnd,(LPCSTR)aCtl3dLow) == NULL &&
                 GetProp(hwnd,(LPCSTR)aCtl3dHigh) == NULL)
#endif
            {
                lpfnWndProc = (FARPROC)SetWindowLong((HWND) hwnd,
                               GWL_WNDPROC, (LONG) lpfnSubclassProc);
#ifdef WIN32
                SetProp(hwnd, (LPCSTR) aCtl3d, (HANDLE)(DWORD)lpfnWndProc);
#else
                SetProp(hwnd, (LPCSTR) aCtl3dLow,
                        LOWORD(lpfnWndProc));
                SetProp(hwnd, (LPCSTR) aCtl3dHigh,
                        HIWORD(lpfnWndProc));
#endif
            }
   }

PRIVATE LRESULT CleanupSubclass(HWND hwnd, UINT wm, WPARAM wParam,
                LPARAM lParam)
        {
        FARPROC lpfnWinProc;

        lpfnWinProc = LpfnGetDefWndProc(hwnd);
#ifdef WIN32
        RemoveProp(hwnd, (LPCSTR) aCtl3d);
#else
        RemoveProp(hwnd, (LPCSTR) aCtl3dLow);
        RemoveProp(hwnd, (LPCSTR) aCtl3dHigh);
#endif
        return CallWindowProc(lpfnWinProc, hwnd, wm, wParam, lParam);
   }
```

Getting the window procedure to call from the subclass functions is a simple matter of getting the properties:

```
   {
#ifdef WIN32
        return (FARPROC) GetProp(hwnd, (LPCSTR) aCtl3d);
#else
   return (FARPROC) MAKELONG((UINT) GetProp(hwnd, (LPCSTR) aCtl3dLow),
                                  GetProp(hwnd, (LPCSTR) aCtl3dHigh));
#endif
   }
```

Drawing the 3-D Effects

CTL3D does the following housekeeping to draw the 3-D effects.

1. It stores a table of current system colors for drawing the 3-D effects. Storing these values in a table gives CTL3D access to the current colors without having to ask Windows. CTL3D uses the colors:

   COLOR_BTNHIGHLIGHT

   COLOR_BTNFACE

   COLOR_BTNSHADOW

   COLOR_BTNTEXT

   COLOR_WINDOW

   COLOR_WINDOWTEXT

   COLOR_GRAYTEXT

   COLOR_WINDOWFRAME

2. It stores a table of brushes in the current system colors:

   COLOR_BTNHIGHLIGHT

   COLOR_BTNFACE

   COLOR_BTNSHADOW

3. It stores a bitmap for drawing radio buttons and check boxes. The developer draws the bitmap in the default system colors; CTL3D converts these standard colors to the current system colors when it loads the bitmap.
4. When Windows sends a WM_SYSCOLORCHANGE message and a CTL3D client calls the **Ctl3dChangeColor** function, CTL3D resets the color and brush tables and reloads the bitmap to reflect the new system colors.

CTL3D uses two primary techniques for drawing the 3-D effects:

To handle the WM_CTLCOLOR message, CTL3D calls **SetTextColor** to set the text color to COLOR_BTNFACE, calls **SetBkColor** to set the background color to COLOR_BTNFACE, and returns a handle to a COLOR_BTNFACE brush.

CTL3D adds the 3-D look to controls by drawing inset and outset rectangles in the appropriate places. The static frame controls in Figure 2 show both types of rectangles (the black frame is an inset rectangle, the white frame is an outset rectangle).

The DRAW3D.H and DRAW3D.C files in the CTL3D sample application contain the routines for adding 3-D effects to your own controls. You can use and modify these files as needed.

Automatic Subclassing

One of the most powerful features of CTL3D is automatic subclassing. An application can usually add 3-D effects to all of its dialog boxes by simply calling **Ctl3dAutoSubclass**.

Ctl3dAutoSubclass sets a task-specific CBT hook. CTL3D keeps track of the CBT hooks it sets for an application by storing each instance handle, task handle, and hook handle in a table. This table holds a maximum of 32 entries.

```
// CLIent Hook
typedef struct
   {
   HANDLE hinstApp;
   HANDLE htask;
   HHOOK hhook;
   } CLIHK;

#define iclihkMax 32
int iclihkMac = 0;
CLIHK rgclihk[iclihkMax];
```

Here's what **Ctl3dAutoSubclass** actually does.

```
/*-------------------------------------------------------------
|  Ctl3dAutoSubclass
|
|     Automatically subclasses all dialogs of the client app.
|
|  Arguments:
|     HANDLE hinstApp:
|
|  Returns:
|
-------------------------------------------------------------*/
PUBLIC BOOL FAR PASCAL Ctl3dAutoSubclass(HANDLE hinstApp)
   {
   HHOOK hhook;
   HANDLE htask;

   if (verWindows < ver31)
      return fFalse;

   if (iclihkMac == iclihkMax)
      return fFalse;

#ifdef WIN32
   htask = (HANDLE)GetCurrentThreadId();
   hhook = SetWindowsHookEx( WH_CBT, (HOOKPROC)Ctl3dHook, hmodLib,
                            (DWORD)htask );
#else
   htask = GetCurrentTask();
   hhook = (*lpfnSetWindowsHookEx)(WH_CBT, (HOOKPROC) Ctl3dHook,
            hmodLib, hinstApp == NULL ? NULL : htask);
#endif
   if (hhook != NULL)
      {
      rgclihk[iclihkMac].hinstApp = hinstApp;
      rgclihk[iclihkMac].htask = htask;
```

```
        rgclihk[iclihkMac].hhook = hhook;
        htaskCache = htask;
        iclihkCache = iclihkMac;
        iclihkMac++;
        return fTrue;
        }
    return fFalse;
    }
```

Having set the hook, CTL3D simply sits back and waits for its filter function (**Ctl3dHook**) to be called. When Windows calls the filter function and the filter function finds a HCBT_CREATEWND hook code, the filter function checks to see if a dialog box is being created. If it is, the filter function asks the application for permission to subclass this particular dialog box with the WM_DLGSUBCLASS message. If the application agrees and the dialog has not been subclassed, the filter function subclasses the dialog box. Here is how the **Ctl3dHook** filter function works.

```
/*-----------------------------------------------------------------
|  Ctl3dHook
|
|     CBT Hook to watch for window creation.
|     Automatically subclasses all
|     dialogs with Ctl3dDlgProc.
|
|  Arguments:
|     int code:
|     WORD wParam:
|     LONG lParam:
|
|  Returns:
|
----------------------------------------------------------------*/
LRESULT _loadds WINAPI Ctl3dHook(int code, WPARAM wParam, LPARAM lParam)
    {
    static HWND hwndHookDlg = NULL;
    int iclihk;
    HANDLE htask;

    if (code == HCBT_CREATEWND)
        {
        LPCREATESTRUCT lpcs;

        lpcs = ((LPCBT_CREATEWND)lParam)->lpcs;
        if (lpcs->lpszClass == WC_DIALOG )
            {
            hwndHookDlg = (HWND) wParam;
            }
        else if (hwndHookDlg != NULL)
            {
            BOOL fSubclass;

            fSubclass = fTrue;
            SendMessage((HWND) hwndHookDlg, WM_DLGSUBCLASS, 0,
                        (LPARAM)(WORD FAR *)&fSubclass);
            if (fSubclass)
                {
                    SubclassWindow((HWND) hwndHookDlg,
(FARPROC)Ctl3dDlgProc);
                hwndHookDlg = NULL;
                }
            }
        }
#ifdef WIN32
    htask = (HANDLE)GetCurrentThreadId();
#else
    htask = GetCurrentTask();
#endif
    if (htask != htaskCache)
        {
        for (iclihk = 0; iclihk < iclihkMac; iclihk++)
            {
            if (rgclihk[iclihk].htask == htask)
                {
                iclihkCache = iclihk;
                htaskCache = htask;
                break;
                }
            }
        // Didn't find task in hook table. This could be bad, but
        // returning 0L is about all we can do.
        return 0L;
        }
#ifdef WIN32
    return CallNextHookEx( rgclihk[iclihkCache].hhook, code, wParam, lParam
);
#else
    return (*lpfnCallNextHookEx)(rgclihk[iclihkCache].hhook, code,
                                 wParam, lParam);
#endif
    }
```

The next step in the automatic subclassing chain is the dialog box subclassing procedure. The **Ctl3dDlgProc** function handles the subclassed dialog box for CTL3D. This function simply follows the same steps that an application would take to add 3-D effects to a dialog box. Here is the source for **Ctl3dDlgProc**.

```
/*-----------------------------------------------------------------
|  Ctl3dDlgProc
|
|     Subclass DlgProc for use with Ctl3dAutoSubclass.
|
|
|  Arguments:
|     HWND hwnd:
|     int wm:
|     WORD wParam:
|     LONG lParam:
|
|  Returns:
|
-----------------------------------------------------------------*/
LRESULT _loadds WINAPI Ctl3dDlgProc(HWND hwnd, UINT wm,
                                    WPARAM wParam, LPARAM lParam)
```

```
    {
    HBRUSH hbrush;
    FARPROC lpfnDlgProc;

    switch (wm)
        {
    case WM_NCDESTROY:
        return CleanupSubclass(hwnd, wm, wParam, lParam);

    case WM_INITDIALOG:
       {
        long l;
        HWND hwndCtl;
        l = CallWindowProc(LpfnGetDefWndProc(hwnd), hwnd, wm, wParam,
lParam);
        Ctl3dSubclassDlg(hwnd, CTL3D_ALL);
        InvalidateRect(hwnd, NULL, TRUE);
        return l;
       }
        break;
    case WM_NCPAINT:
    case WM_NCACTIVATE:
    case WM_SETTEXT:
        return Ctl3dDlgFramePaint(hwnd, wm, wParam, lParam);
        break;
#ifdef WIN32
    case WM_CTLCOLORSCROLLBAR:
    case WM_CTLCOLORBTN:
    case WM_CTLCOLORDLG:
    case WM_CTLCOLOREDIT:
    case WM_CTLCOLORLISTBOX:
    case WM_CTLCOLORMSGBOX:
    case WM_CTLCOLORSTATIC:
#else
    case WM_CTLCOLOR:
#endif
                (FARPROC) lpfnDlgProc = (FARPROC)
                              GetWindowLong(hwnd, DWL_DLGPROC);
#ifdef WIN32
                if (lpfnDlgProc == NULL ||
                    IsBadReadPtr(lpfnDlgProc, 1) ) {
                   hbrush = Ctl3dCtlColorEx(wm, wParam, lParam);
                }
                else {
                   hbrush = (HBRUSH) (*lpfnDlgProc)(hwnd, wm, wParam,
lParam);
                if (hbrush == (HBRUSH) fFalse ||
                    hbrush == (HBRUSH)1)
                      hbrush = Ctl3dCtlColorEx(wm, wParam, lParam);
                }
#else
                if (lpfnDlgProc == NULL ) {
                   hbrush = Ctl3dCtlColorEx(wm, wParam, lParam);
                }
                else {
                   hbrush = (HBRUSH) (*lpfnDlgProc)(hwnd, wm, wParam,
lParam);
```

```
                    if (hbrush == (HBRUSH) fFalse ||
                        hbrush == (HBRUSH)1)
                       hbrush = Ctl3dCtlColorEx(wm, wParam, lParam);
                }
#endif

                if (hbrush != (HBRUSH) fFalse)
                        return  (LRESULT)hbrush;
                break;

                }
    return CallWindowProc(LpfnGetDefWndProc(hwnd), hwnd, wm, wParam,
lParam);
    }
```

The final step is subclassing the controls. CTL3D calls **Ctl3dSubclassDlg**, which loops through each child window of the dialog box.

```
/*-----------------------------------------------------------------
|  Ctl3dSubclassDlg
|
|     Call this during WM_INITDIALOG processing.
|
|  Arguments:
|     hwndDlg:
|
-----------------------------------------------------------------*/
PUBLIC BOOL FAR PASCAL Ctl3dSubclassDlg(HWND hwndDlg, WORD grbit)
   {
   HWND hwnd;

   if (!f3dDialogs)
      return fFalse;

   for(hwnd = GetWindow(hwndDlg, GW_CHILD); hwnd != NULL &&
      IsChild(hwndDlg, hwnd); hwnd = GetWindow(hwnd, GW_HWNDNEXT))
      {
      DoSubclassCtl(hwnd, grbit);
      }
   return fTrue;
   }
```

For each child window, CTL3D then loops through its list of standard control window procedures. If it finds a match for the child window, CTL3D checks to ensure that it can work with the particular control style (for example, CTL3D will not subclass a static control that has the SS_ICON style, as explained earlier in this article). If the control has the correct style(s), CTL3D subclasses the control.

```
/*-------------------------------------------------------------
|  DoSubclassCtl
|
|     Actually subclass the control.
|
|
|  Arguments:
|     HWND hwnd:
```

```
|      WORD grbit:
|
|      Returns:
|
-----------------------------------------------------------------*/
PRIVATE BOOL DoSubclassCtl(HWND hwnd, WORD grbit)
    {
    LONG style;
    int ct;
    BOOL fCan;
    extern HANDLE hinstLib;
    char szClass[64];

    // Is this already subclassed by CTL3D?
#ifdef WIN32
    if ( GetProp(hwnd,(LPCSTR)aCtl3d) != NULL )
#else
    if ( GetProp(hwnd,(LPCSTR)aCtl3dLow) != NULL &&
         GetProp(hwnd,(LPCSTR)aCtl3dHigh) != NULL)
#endif
       return fFalse;

    GetClassName(hwnd, szClass, sizeof(szClass));

    for (ct = 0; ct < ctMax; ct++)
            {
            if ((mpctcdef[ct].msk & grbit) &&
                     (lstrcmp(mpctcdef[ct].sz,szClass) == 0))
                    {
                    style = GetWindowLong(hwnd, GWL_STYLE);
                    fCan = mpctcdef[ct].lpfnFCanSubclass(hwnd,
                                                 style, grbit);
                    if (fCan == fTrue)
                            {
                            SubclassWindow(hwnd, mpctctl[ct].lpfn);
                            }
                    return fCan != fFalse;
                    }
            }
    return fFalse;
    }
```

A Note About the CTL3D Source Code

The source for CTL3D was originally distributed by the Microsoft Developer Network to show how CTL3D works and to help debug problems. However, we received a number of requests from developers asking how they could modify CTL3D to suit their own needs. Because a number of applications rely on CTL3D working as originally distributed, the need for a single release point for the library became apparent. For this reason, Microsoft will no longer distribute the source for CTL3D.

To help compensate for the missing source code, we have added the "How CTL3D Works" section to this article and included the DRAW3D.C and DRAW3D.H files. If you have questions about a particular aspect of CTL3D, feel free to contact Kyle Marsh.

Acknowledgments

We would like to thank Andreas Furrer for providing information on using CTL3D with Pascal.