Why isn't my transparent static control transparent?

devblogs.microsoft.com/oldnewthing/20111028-00

October 28, 2011

Raymond Chen

A customer reported that their application uses transparent static controls positioned over a bitmap image control, but even though they set the Transparent property on the static control, the static control isn't transparent. The customer was kind enough to provide clear steps to illustrate the problem:

- Open Visual Studio 2005 or 2008.
- From the menu, select *File*, *New File*, Visual C++, *Resource Template File (RCT)*.
- Right-click on the RCT file, select *Add Resource*, and add a bitmap named `IDB_BITMAP1`.
- Open the dialog box (`IDD_DIALOG1`) and add a "Picture Control", specifying `IDC_BITMAP_1` as its ID.
- Change the `IDC_BITMAP_1` type to *Bitmap* and change the value of the Image property to `IDB_BITMAP1`.
- Add a "Static Text" control `IDC_TEST_STATIC` and set its caption to "This is a test".
- Reposition the static control so it overlaps the `IDC_BITMAP_1` control.
- On the `IDC_TEST_STATIC` control, set the *Transparent* property to *True*.
- Type Ctrl+T to test the dialog and observe that the static control is not transparent.

Dialog

This is a test

The *Transparent* property in Visual Studio corresponds to the `WS_EX_TRANSPARENT` window style, and the documentation explains that

> `WS_EX_TRANSPARENT` : The window should not be painted until siblings beneath the window (that were created by the same thread) have been painted. The window appears transparent because the bits of underlying sibling windows have already been painted.

The observed behavior, therefore, matches the documentation: The control underneath (the bitmap control) paints first, and then the static control paints on top of it. And a static text control paints by filling with the background brush and drawing the text on top of it. You can customize this behavior by responding to the `WM_CTLCOLORSTATIC` message:

```
HBRUSH CTestDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = __super::OnCtlColor(pDC, pWnd, nCtlColor);
 if (pWnd->GetExStyle() & WS_EX_TRANSPARENT) {
  pDC->SetBkMode(TRANSPARENT);
  hbr = GetStockBrush(HOLLOW_BRUSH);
  // even better would be to use a pattern brush, if the background is fixed
 }
 return hbr;
}
```

The customer appreciated the explanation but was puzzled as to why the *Transparent* is available if it doesn't work. "We understand that we can use the `WS_EX_TRANSPARENT` window style to create a transparent window and it will be painted after its underlying siblings, but the window style by itself doesn't make the static control transparent. But if we have to write the code above, why is the *Transparent* property available in the Properties box?" They included a screen shot from Visual Studio where the built-in help text for the *Transparent* property reads "Specifies that the control will have a transparent background."

The `WS_EX_TRANSPARENT` style doesn't mean "transparent"; it means "paint over siblings." The style is called "transparent" not because it makes the window transparent but because it makes transparency *possible*. It is one of the steps (but not the only one) for making child controls render transparently. Another important step is ensuring that the control does not erase its background in its `WM_ERASEBKGND`, and that's the step that the `OnCtlColor` override performs.

Why is the *Transparent* property offered for static controls when it doesn't actually work? Shouldn't it be disabled for static controls?

The reason why it is offered is that it is a general window style that can be set on any control. Visual Studio doesn't know which controls can render transparently and which ones don't, or what extra steps are necessary to get the ones who can render transparently to actually do so. It just exposes the `WS_EX_TRANSPARENT` style and hopes that you know what you're doing.

In retrospect, it was a poor chose of name for the style. And the incorrect online help doesn't make things any better.

Bonus chatter: Note that the `WS_EX_TRANSPARENT` extended style is overloaded. In addition to affecting painting, it also affects hit-testing.

Raymond Chen

Follow