

A Simple Workaround for Strange Repainting Problems

By Jeff Trahan (CIS:72737,2154)

The problem is simply stated: When you unload a form, the forms behind the newly unloaded form do not repaint properly until Windows is allowed again to process events in the queue. While at first putting a "DoEvents" statement seems to fix the problem, this has other problems associated with it that I will discuss later.

I will be blunt in stating that this should be a language feature of VB. Hopefully, this will be in VB 4.0 or 4.5. A property for the Form object such as "RepaintOnHide" would be simple in implementation and would avert the need for a workaround such as this.

Why "DoEvents" is a non-solution

I am convinced that only in certain places (and few at that) do you want a "DoEvents" statement. The problem lies in that every time "DoEvents" is called, all pending events for your application and all others are processed. While it may be OK to let other applications breathe, you don't necessarily want your application to service events at odd times. For example, if you are unloading one modal dialog box and loading another and you have a "DoEvents" after the unload, any events pending will be serviced. Let's say the user clicked on the app's button bar. You would start a task generated by that button bar click before the second modal dialog box is in control (remember, a modal dialog box takes control of all events until it is "unloaded"). This could be disastrous! It is easy to see that DoEvents is a non-solution if you are writing an "idiot-proof" application.

In all fairness, there is a way around this "gotcha" and will make DoEvents workable. It is to disable all screen elements that shouldn't be clicked. In some instances, this means every screen element. Others will only require a portion to be disabled. Either way, this is a lot of statements and requires knowing what forms (modal, non-modal, and MDI) are present. Sounds like a lot of coding!

I have found that a DoEvents is essential to proper handling of a modal dialog box with some looping in it. In an application I am working on, I have a "progress" modal dialog box that is displayed while a database is searched. To get the Cancel button to work correctly, a DoEvents is executed regularly in the search loop. This is a rare instance and in this case having all the pending events serviced is exactly what is desired.

The Solution

I must confess that I stumbled upon this solution. I first tried a DLL routine in "C" using PeekMessage(), GetMessage(), etc. I thought that I could filter out all events except "paint"-type events. Using this technique, I was not able to get it to function correctly. This could be my implementation or could be that WM_PAINT and the other paint events are special in nature and can't be filtered. I couldn't find any explanation in the Win/16 SDK docs or in Petzold's book.

I then thought about traversing the "Forms" collection in VB and calling the *Refresh* method. After dealing with the runtime error resulting from the MDIForm not having a *Refresh* method, I found that it works! I put the following routine in one of the global .BAS modules and call it in the appropriate places:

```
' This uses the "Forms" collection to force a repaint of all forms.
The
' error release must be in place for the one MDIForm as it has no
Refresh
' method.
```

```

Sub RefreshForms ()

    On Error Resume Next ' For those forms without a Refresh method

    Dim i%

    For i% = 0 To Forms.Count - 1
        Forms(i%).Refresh
    Next i%

End Sub

```

I found that putting a "RefreshForms" after each "Form.Show 1" gives very good results.

This seems so simple I thought it must in a MS knowledgebase or in some other source. I searched the Developer Network CD8 and found nothing using the forms collection and refresh method. I also looked at the current version of the VBTIPS.ZIP and there is no reference to this technique. I doubt if I am the first to think of this, but am willing to share it with all. I am forwarding this info to both sources mentioned above, so please don't deluge them with this info.

I also thought that forcing a Refresh of every form would cause "phantom" repainting and slow things down. It doesn't. Either through Windows or VB there is a mechanism keeping track to avoid unnecessary repainting. There is seemingly no performance penalty for using this technique compared with the DoEvents solution.

Compatibility

I have tested this with MDI parents and their children as well as modal and non-modal non-MDI forms using VB 3.0 Pro for Windows. Also, the Videosoft VSVBX, and many of MicroHelp's VBTools 4 package were a part of my test. So far, I have seen no strange behavior. Based on this, I would suspect any strange painting behavior is a bug in the custom control in question.

Contacting Me

I would appreciate hearing from those using this technique and anyone having problems with this technique. We are going to ship this routine in a product very soon, so the more testing the better. Please contact me via CompuServe either in the "Programming Issues" section of the MSBASIC forum or by private e-mail. My CIS account is 72737,2154.

Jeff Trahan