Call WinForms on Multiple Threads

Use the ISynchronizeInvoke interface to marshal calls to the correct thread, and put HTML on the clipboard that other apps can use.

by Juval Löwy and Karl E. Peterson

Technology Toolbox

- ❑ VB.NET
- ☑ C#
- ❑ SQL Server 2000
- ❑ ASP.NET
- ❑ XML
- ☑ VB6
- ☑ Other:
  - VB5
  - WinForms

Go Online!

Use these Locator+ codes at www.visualstudiomagazine.com to go directly to these related resources.

Download

VS0302QA Download the code for this article. It includes the Synchronizer project, which contains a WinForms client that uses a Calculator class employing a generic implementation of ISynchronizeInvoke; and the HtmlClip project, which contains routines for sending HTML to the clipboard.

Discuss

VS0302QA_D Discuss this article in the C# forum.

Read More

VS0302QA_T Read this article online. It includes Listing A.

VB0106AP_T Ask the VB Pro, "Soup Up Office VBA," by Karl E. Peterson

VB9812AP_T Ask the VB Pro, "Verifying Internet Access," by Karl E. Peterson

VB9712AP_T Ask the VB Pro, "Copy and Paste with RichTextBox," by Karl E. Peterson

Q: Call WinForms on Multiple Threads

My WinForms application has a worker thread that updates the main windows. The documentation warns against calling the form on multiple threads (why?), and indeed, it crashes occasionally if I do. How can I call methods on the form from multiple threads?

A:

Every WinForms class that derives from the Control class (including Control) relies on the underlying Windows messages and on a message pump loop to process them. The message loop must have thread affinity, because messages to a window are delivered only to the thread that creates it. As a result, you can't call message-handling methods from multiple threads, even if you provide synchronization. Most of the plumbing is hidden from you, because WinForms use delegates to bind messages to event-handling methods. WinForms convert the Windows message to a delegate-based event, but you still must be aware that only the thread that creates the form can call its event-handling methods, because of the primordial message loop. If you call such methods on your own thread, they'll execute on it instead of on the designated form thread. You can call any methods that you know aren't message handlers (such as your own custom methods) from any thread.

The Control class (and the derived classes) implement an interface defined in the System.ComponentModel namespace—ISynchronizeInvoke—to address the problem of calling message-handling methods from multiple threads:

```
public interface ISynchronizeInvoke
{
   object Invoke(Delegate
      method,object[] args);
   IAsyncResult BeginInvoke(Delegate
      method,object[] args);
   object EndInvoke(IAsyncResult
      result);
   bool InvokeRequired {get;}
}
```

ISynchronizeInvoke provides a generic, standard mechanism to invoke methods on objects residing on other threads. For example, the client on thread T1 can call ISynchronizeInvoke's Invoke() method on an object if the object implements ISynchronizeInvoke. The implementation of Invoke() blocks the calling thread, marshals the call to T2, executes the call on T2, marshals the returned values to T1, then returns control to the calling client on T1. Invoke() accepts a delegate targeting the method to invoke on T2, and a generic array of objects as parameters.

The caller can also check the InvokeRequired property, because you can call ISynchronizeInvoke on the same thread as the one the caller tries to redirect the call to. The caller can call the object methods directly if InvokeRequired returns false.

For example, suppose you want to invoke the Close method on some form from another thread. You can use the predefined MethodInvoker delegate, and call Invoke:

```
Form form;
/* obtain a reference to the form,
   then: */
ISynchronizeInvoke synchronizer;
synchronizer = form;

if(synchronizer.InvokeRequired)
{
   MethodInvoker invoker  = new
      MethodInvoker(form.Close);
   synchronizer.Invoke(invoker,null);
}
else
   form.Close();
```

VB5, VB6 • Prepare HTML for the Clipboard

```
Public Function HtmlDescribed(ByVal Fragment As _
  String) As String
  Dim Data As String
  Dim nPos As Long
  Const Description As String = _
    "Version:1.0" & vbCrLf & _
    "StartHTML:aaaaaaaaaa" & vbCrLf & _
    "EndHTML:bbbbbbbbbb" & vbCrLf & _
    "StartFragment:cccccccccc" & vbCrLf & _
    "EndFragment:dddddddddd" & vbCrLf
  Const FragmentStart As String = _
    "<!--StartFragment-->"
  Const FragmentEnd As String = _
    "<!--EndFragment-->"
  Const Fmt As String = "0000000000"

  ' Add the starting and ending tags for the
  ' HTML fragment by looking for <body> tag.
  nPos = InStr(1, Fragment, "<body", _
    vbTextCompare)
  Select Case nPos
    Case 0
      Fragment = "<html><body>" & vbCrLf & _
        FragmentStart & Fragment
    Case Else
      nPos = InStr(nPos, Fragment, ">")
      If nPos > 0 And nPos < Len(Fragment) _
        Then
        Fragment = Left$(Fragment, nPos) & _
          FragmentStart & Mid$(Fragment, _
          nPos + 1)
      End If
  End Select
  nPos = InStr(1, Fragment, "</body", _
    vbTextCompare)
  Select Case nPos
    Case 0
      Fragment = Fragment & FragmentEnd & _
        vbCrLf & "</body></html>"
    Case Else
      Fragment = Left$(Fragment, nPos - 1) & _
        FragmentEnd & Mid$(Fragment, nPos)
  End Select

  ' Build the HTML given the description, the
  ' fragment, and the context. And, replace the
  ' offset placeholders in the description with
  ' values for the offsets of StartHMTL,
  ' EndHTML, StartFragment, and EndFragment.
  ' Offsets need to be zero-based when placed on
  ' clipboard, so subtract 1
  ' from each before injecting.
  Data = Description & Fragment
  Data = Replace(Data, "aaaaaaaaaa", _
    Format$(Len(Description), Fmt))
  Data = Replace(Data, "bbbbbbbbbb", _
    Format$(Len(Data), Fmt))
  nPos = InStr(Data, FragmentStart) - 1
  Data = Replace(Data, "cccccccccc", _
    Format$(nPos + Len(FragmentStart), Fmt))
  nPos = InStr(Data, FragmentEnd) - 1
  Data = Replace(Data, "dddddddddd", _
    Format$(nPos, Fmt))
  ' Return attributed string.
  HtmlDescribed = Data
End Function
```

Listing 1 This routine is useful when you build the descriptive header string for an HTML fragment. If you pass an entire HTML document, the StartFragment tag is injected immediately following the <body> tag, and the EndFragment tag immediately before </body>. If you pass a fragment rather than a complete document, the routine makes it minimally whole by the addition of <html> and <body> tag sets. Use the ideas in this routine to construct variations for other scenarios easily.

ISynchronizeInvoke isn't limited to WinForms. For example, a Calculator class provides the Add() method for adding two numbers, and it implements ISynchronizeInvoke. A client makes sure the method executes on the correct thread by calling ISynchronizeInvoke.Invoke() (download Listing A from the *VSM* Web site; see the Go Online box for details).

You might want to be able to invoke the call asynchronously, because it's marshaled to a different thread from that of the caller. The BeginInvoke() and EndInvoke() methods let you do this. You use these methods in accordance with the general .NET asynchronous programming model: Use BeginInvoke() to dispatch the call, and EndInvoke() to wait or be notified about completion and collect returned results.

It's worth mentioning that ISynchronizeInvoke methods aren't type-safe. A mismatch in type causes an exception to be thrown at run time, rather than a compilation error. Pay extra attention when you use ISynchronizeInvoke, because the compiler won't be there for you.

Implementing ISynchronizeInvoke requires you to use a delegate to invoke the method dynamically using late binding. Every delegate type provides the DynamicInvoke() method:

```
public object DynamicInvoke(object[]
  args);
```

In the abstract, you must post the method delegate to the actual thread the object needs to run on, and have it call DynamicInvoke() on the delegate in Invoke() and BeginInvoke(). Implementing ISynchronizeInvoke is a nontrivial programming feat. The source files accompanying this article contain a helper class called Synchronizer and a test application demonstrating how the Calculator class in Listing A can implement ISynchronizeInvoke using the Synchronizer class (download the source code). Synchronizer is a generic implementation of ISynchronizeInvoke. You can use Synchronizer as-is by either deriving from it or containing it as a member object, then delegating your implementation of ISynchronizeInvoke to it.

The key element of implementing Synchronizer is using a nested class called WorkerThread. WorkerThread has a queue of work items. WorkItem is a class containing the method delegate and the parameters. Both Invoke() and BeginInvoke() add a work-item instance to the queue. WorkerThread creates a .NET worker thread, which monitors the work-item queue. When the queue has items, the worker thread retrieves them, then calls DynamicInvoke() on the method. —*J.L.*

Q: Work With HTML and the Clipboard

My application needs to place HTML on the clipboard, but I can't figure out how to do this so that other applications understand that's what it is. I've seen references to

the HTML Clipboard Format (CF_HTML), but I can't find the definition for that constant. How should I proceed?

A:

Using the CF_HTML clipboard format with the Windows clipboard is a bit confusing, in part because it's not a native clipboard format; it's a *registered format*, so it isn't a constant at all, because its value differs from system to system. You can obtain registered clipboard-format values with a simple API call—RegisterClipboardFormat. The first time this function is called with a given string, it returns a unique number in the range C000–FFFF. Each subsequent call that any process running on the system makes returns the same value. The magic string to use for this format is "HTML Format":

```
Private Declare Function _
   RegisterClipboardFormat _
   Lib "user32" _
   Alias "RegisterClipboardFormatA" _
   (ByVal lpString As String) As Long
Dim CF_HTML As Long
Const RegHtml As String = "HTML Format"
CF_HTML = _
   RegisterClipboardFormat(RegHtml)
```

Only the thread that creates the form can call its event-handling methods, because of the primordial message loop.

You must construct a descriptive header and prepend it to the data before you can place your HTML data onto the clipboard. This header provides other applications with the description's version information, with offsets within the data where the HTML starts and stops, and with information about where the actual selection begins and ends. Conceptualize the selection by considering a user who might select a portion of an HTML document or even an element (such as a few rows in a table). Other portions of the page (such as inline style definitions) might be required to render the selection fully. You likely must supply more than the raw selection to put HTML on the clipboard in its full context. A sample header might look like this:

```
Version:1.0
StartHTML:000000258
EndHTML:000001491
StartFragment:000001172
EndFragment:000001411
```

Applications use the StartFragment and EndFragment attributes to determine which data to paste, and they might or might not use the remaining HTML to help format the selected portion. You must inject HTML comments into the data to identify the selected area further. Obviously, you must do this before you build the final header, because the offsets won't be stable otherwise. The opening/closing comment tags for the selected data are "<!—StartFragment—>" and "<!—EndFragment—>", respectively (see Listing 1).

I don't have enough room here to detail all of this header's aspects, so I'll hit a few highlights and refer you to the sample code and further reading (see Additional Resources). You must keep several critical points in mind. The offsets listed in the header are zero-based, so you must adjust your string-manipulation routines accordingly. Also, if you're reading as well as writing these headers, you must assume that the number of digits is variable (for example, Internet Explorer [IE] uses 9, and Word uses 10).

Finally, if you place only CF_HTML on the clipboard, applications such as Word and FrontPage don't know what to do with it. You must also supply a plain-text rendition of the stylized HTML to the clipboard for these apps to behave as expected. Scads of tools perform HTML-to-text conversions, or the extremely macho might prefer to roll their own parsers. But, no Windows programmer should ever have to hand-parse HTML again. You can call upon the OS instead for this everyday task:

```
Public Function Html2Text(ByVal Data _
   As String) As String
      Dim obj As Object
      On Error Resume Next
      Set obj = _
         CreateObject("htmlfile")
      obj.Open
      obj.Write Data
      Html2Text = obj.Body.InnerText
End Function
```

Leveraging IE isn't necessarily the quickest method for parsing HTML, but the expediency it offers is a good tradeoff in this case. —*K.E.P.*

Juval Löwy is a software architect and the principal of IDesign, a consulting and training company focused on .NET design and .NET migration. Juval is a Microsoft regional director for the Silicon Valley, working with Microsoft on helping the industry adopt .NET. His latest book is *Programming .NET Components* (O'Reilly & Associates). Contact him at www.idesign.net.

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the *VSM* Editorial Advisory Board. Online, he's a Microsoft MVP and a section leader on several DevX forums. Find more of Karl's VB samples at www.mvps.org/vb. Reach him at karl@mvps.org.

Additional Resources

- "HOWTO: Add HTML Code to the Clipboard by Using Visual Basic": http://support.microsoft.com/default.aspx?scid=kb;en-us;q274326
- "HTML Clipboard Format": http://msdn.microsoft.com/workshop/networking/clipboard/htmlclipboard.asp