Visual Studio Magazine Online
Classic VB Corner

Simple Asynchronous Downloads

ONLINE ONLY

No need to add external dependencies, or even call APIs, to easily grab online data.

March 27, 2008 · by Karl E. Peterson

There are probably as many strategies for downloading files from the Internet as there are .NET programmers -- that is, no where near the number of Classic VB developers, but still more than a mere handful. Ask the question in a newsgroup, and you'll quickly get a good half-dozen suggestions and a handful of links, all pointing to different ideas. But rarely, if ever, do I see anyone suggest what I consider to be probably the single most powerful "unknown" native method Classic VB offers: the AsyncRead method of UserControl and UserDocument objects.

Two things I like best about AsyncRead is that it's native -- no external dependencies are dragged into a project -- and it's asynchronous. When you call AsyncRead, execution returns immediately to your application, and one or more events are later fired alerting you to the download status. VB5 introduced the native UserControl and VB6 improved on it a bit -- in particular, in this very area. Both versions will fire an AsyncReadComplete event when the entire file has been received. VB6 also fires AsyncReadProgress events at semi-random intervals during the download process.

While you can certainly use these techniques in VB5, you'll find them somewhat more flexible in VB6. I'll point out the areas with significant improvements. You can use conditional compilation to easily take the most advantage of the newer capabilities VB6 offers. By defining a conditional constant in your UserControl that tells it whether it's being compiled into a VB5 or VB6 app, you selectively enable the enhanced capabilities of VB6 when available. For example, here's a simple method that kicks off a new download:

```
#If VB6 Then
Public Sub DownloadStart(ByVal URL As String, _
   Optional ByVal Mode As AsyncReadConstants = _
   vbAsyncReadResynchronize)
#Else
Public Sub DownloadStart(ByVal URL As String)
#End If
   If Len(URL) Then
      ' Already downloading something, need to cancel!
      If m_Key Then Me.DownloadCancel

      ' Use current time as PropertyName.
      m_Key = GetTickCount()
      Debug.Print CStr(m_Key); " - "; URL

      ' Request user-specified file from web.
      On Error Resume Next
      #If VB6 Then
         UserControl.AsyncRead URL, vbAsyncTypeByteArray, _
```

```
            CStr(m_Key), Mode
      #Else
         UserControl.AsyncRead URL, vbAsyncTypeByteArray, _
            CStr(m_Key)
      #End If
      If Err.Number Then
         Debug.Print "AsyncRead Error"; Err.Number, _
            Err.Description
      End If
   End If
End Sub
```

Microsoft actually provides a rudimentary Knowledge Base article on using AsyncRead, so I'll just focus on some of the more interesting aspects. Note that VB6 offers an array of option flags (AsyncReadConstants) that control how downloads are handled with respect to the cache. This allows you to force a new download (essentially, a Refresh), just use whatever's already cached (Offline-mode), only update files if the cached copy is older and so on. I've offered this as an optional parameter, for VB6 builds, in the routine above.

Probably the most important aspect to be aware of is the PropertyName parameter, to which I'm here passing a fairly unique string (GetTickCount). AsyncRead actually supports multiple simultaneous downloads, and this property is the only way you can control specific downloads. In this case, I chose to write a very simple control that just supports a single download, and cancels pending downloads using the CancelAsyncRead method when a new one is requested:

```
Public Sub DownloadCancel()
   ' Attempt to cancel pending download.
   On Error Resume Next
   UserControl.CancelAsyncRead CStr(m_Key)
   Debug.Print CStr(m_Key); " - cancel"
   If Err.Number Then
      Debug.Print "CancelAsyncRead Error"; Err.Number, _
         Err.Description
   End If
End Sub
```

Note that I also requested the data be delivered in a Byte array when I made the AsyncRead call. This seems more flexible than asking for a file to be created, as that's one of the simplest of all MSBASIC operations. You might want to play around with requesting a Picture object, though, if you're downloading images. But converting Byte arrays to Picture objects isn't very difficult; see the NetCam sample on my site for the code to do that.

When AsyncRead completes a download, an AsyncReadComplete event (d'oh!) fires. This is the spot to package up the received bytes, perhaps stashing them in a module level array for the user to grab as needed, and raise an event to notify the user:

```
Private Sub UserControl_AsyncReadComplete( _
   AsyncProp As AsyncProperty)
   ' Record duration of download.
   m_Duration = Abs(GetTickCount - m_Key)
```

```
   ' Reset key to indicate no current download.
   Debug.Print CStr(m_Key); " - done"
   m_Key = 0
   ' Extract downloaded data from AsyncProp
   With AsyncProp
      On Error GoTo BadDownload
      If .AsyncType = vbAsyncTypeByteArray Then
         ' Cache copy of downloaded bytes
         m_Bytes = .Value
         m_nBytes = UBound(m_Bytes) + 1
         RaiseEvent DownloadComplete(m_nBytes)
      End If
   End With
   Exit Sub
BadDownload:
   m_nBytes = 0
   RaiseEvent DownloadFailed(Err.Number, Err.Description)
End Sub
```

As I mentioned earlier, VB6 offers other neat flourishes, such as the intermediate notifications via the AsyncReadProgress event:

```
#If VB6 Then
Private Sub UserControl_AsyncReadProgress( _
   AsyncProp As AsyncProperty)
   ' Extract downloaded data from AsyncProp
   With AsyncProp
      On Error GoTo BadProgress
      If .AsyncType = vbAsyncTypeByteArray Then
         ' Cache copy of downloaded bytes
         m_Bytes = .Value
         m_nBytes = UBound(m_Bytes) + 1
         RaiseEvent DownloadProgress(m_nBytes)
      End If
   End With
   Exit Sub
BadProgress:
   ' No need to raise an event, as progress may resume?
End Sub
#End If
```

Providing the raw data to your user is as simple as exposing a read-only Bytes property:

```
#If VB6 Then
Public Property Get Bytes() As Byte()
#Else
Public Property Get Bytes() As Variant
#End If
   ' NOTE: Change conditional constant at top
   '       of module to match target language!
   Bytes = m_Bytes()
End Property
```

What could be simpler? You can download a completely functional copy of the NetGrab UserControl I wrote, along with a little demo showing how to use it, from my site.

Self-Updating Applications

When I showed a friend my little NetGrab control, his first reaction was that it'd be just perfect for providing an automated background download of program updates. Which, indeed, it would. You could easily use this control, together with a timer or just at startup, to occasionally check whether new versions are available. (Of course, don't do this without your user's permission!) If you don't want to burden the system with Timer ticks even once per minute, download the TimerObj sample from my site for a code-based timer that works with Interval settings in the Long range.

The best approach for a self-updating application actually involves bringing another little utility app into the equation. The perils of actually over-writing the executing application are just too great. So the simplest solution is to either write an app-launcher that first checks for available updates, downloads and installs them if available, and then launches the main application. A slight twist on this would be to download the update in the background, then ask the user if they'd like to restart, at which point you'd spawn your over-writing launch helper.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.