Visual Studio Magazine Online
Classic VB Corner

Inside Alternative Data Streams

NTFS offers an almost unknown way to obscure streams of data behind the most innocent looking files. Find out how to do this with VB6.

November 3, 2009 · by Karl E. Peterson

NTFS, the file system of choice on most machines these days, offers something called Alternate Data Streams (ADS) to tuck data away, out of sight from most users. If you search for ADS, you'll see that these neat little payloads are the current rage of the security crowd. As they were five years ago. And before that, too. They certainly are eye-opening, the first time you run into them.

The NTFS file system (do you hate redundancies like that as much as I do?) supports multiple streams of data within each file and folder, and has been around since the introduction of Windows NT. It was originally meant to provide some level of compatibility with HFS, the old Macintosh Hierarchical File System. It's somewhat ironic that the FAT32 thumb drives many folks carry around don't support ADS, but do offer cross-platform capabilities with today's Apple computers.

Under FAT32, the file system stores the filename and extension, its associated attributes, and the file's data. In NTFS, the file system of course also stores the filename and extension, security attributes, the main stream of file data, and optionally many other alternate streams of data as well. The main data stream is unnamed, but each of the alternate streams goes by its own name.

Working with ADS

To specify an ADS, you simply append a colon to the file name and follow that with the stream name. Each stream, including the main one, also carries a totally superfluous extension, ":$DATA", which may be freely omitted when referencing a stream. The gritty details are all available through Google, but the important thing to be aware of is that most VB file I/O functionality works perfectly fine with streams.

There are a few areas where native VB needs a little augmentation from Windows, however. Foremost among them would be the enumeration of ADS within any given file or folder. I've created a drop-in ready CStreams class, suitable for VB5/VB6/VBA, which you may download from the Streams sample on my website. CStreams provides an enumeration of all the streams, and their sizes, within any file or folder. It must be said, though, that ADS within folders require an extra level of permissions to get into. More on that in a bit.

So let's get right into it. Here's the CStreams.Refresh method. This is called as needed, after the class has been handed a file name to work with, and has used GetVolumeInformation to determine the file does indeed reside on an NTFS volume.

```
Public Sub Refresh()
   Dim hFile As Long
   Dim ioStatus As IO_STATUS_BLOCK
   Dim InfoBlock() As Byte
   Dim BlockSize As Long
   Dim Flags As Long
   Dim nRet As Long

   ' Reset cached values.
   m_Count = 0

   ' Streams not available for folders unless
   ' process has backup privilege.
   If IsFolder(m_FileName) Then
      If BackupPrivs(True) = False Then Exit Sub
      Flags = FILE_FLAG_BACKUP_SEMANTICS
   End If

   ' Attempt to read stream names by building progressively
   ' larger buffer until sufficient to contain all data.
   hFile = CreateFile(m_FileName, 0&, FILE_SHARE_READ, _
      ByVal 0&, OPEN_EXISTING, Flags, 0&)
   If hFile <> INVALID_HANDLE_VALUE Then
      BlockSize = 1024
      Do
         ReDim InfoBlock(0 To BlockSize - 1) As Byte
         nRet = NtQueryInformationFile(hFile, ioStatus, _
            InfoBlock(0), BlockSize, FileStreamInformation)
         Select Case nRet
            Case STATUS_SUCCESS
               If ioStatus.Information Then
                  Call GetStreams(InfoBlock())
               End If
            Case STATUS_BUFFER_OVERFLOW
               BlockSize = BlockSize * 2
            Case Else
               Debug.Print "NtQueryInformationFile failed: &h" _
                           & Hex$(nRet), BlockSize
         End Select
      Loop While nRet = STATUS_BUFFER_OVERFLOW

      ' Release open file handle.
      Call CloseHandle(hFile)

      ' Restore prior process privileges.
      If CBool(Flags And FILE_FLAG_BACKUP_SEMANTICS) Then
         Call BackupPrivs(False)
      End If
   End If
End Sub
```

Let's skip over the backup semantics necessary for folders, for a moment. Enumerating ADS requires a call to the only-recently documented (and rather sparsely at that) NtQueryInformationFile API function, requesting a memory block that contains all the stream information. In order to make this call, we first must obtain an API-based file handle using CreateFile to open the existing file.

Once we have the file handle, we create what we hope to be an adequate-sized buffer for NtQueryInformationFile to stuff and give that a shot. If there are a lot of ADS within the file in question, the buffer may need to be expanded until it is sufficiently sized to hold all the requested information. We then begin the happy task of picking apart the buffer to obtain the information we sought. Each ADS within the file is identified within the buffer with a structure that looks roughly like this:

```
Private Type FILE_STREAM_INFORMATION
   NextEntryOffset As Long
   StreamNameLength As Long
   StreamSize As LARGE_INTEGER
   StreamAllocationSize As LARGE_INTEGER
   StreamName As String
End Type
```

I say "roughly" because Windows doesn't do strings quite the way VB does. What we really have are just the raw Unicode bytes laid out in order, and we can extract the StreamName string directly using a bit of math to calculate its offset and the length provided in the StreamNameLength element. Walking through this InfoBlock is just this simple:

```
Private Sub GetStreams(InfoBlock() As Byte)
   Dim nIndex As Long
   Dim lpBlock As Long

   ' Reset count to zero, and walk through information block.
   m_Count = 0
   nIndex = LBound(InfoBlock)
   Do
      ' Expand persisted storage for stream information
      ReDim Preserve m_Streams(0 To m_Count) _
         As FILE_STREAM_INFORMATION

      With m_Streams(m_Count)
         ' Calculate pointer to beginning of this block.
         lpBlock = VarPtr(InfoBlock(nIndex))

         ' Find offset to next record.
         .NextEntryOffset = PointerToDWord(lpBlock)

         ' Read each of the remaining attributes for this stream.
         .StreamNameLength = PointerToDWord(lpBlock + 4)
         .StreamSize = PointerToLargeInt(lpBlock + 8)
         .StreamAllocationSize = PointerToLargeInt(lpBlock + 16)
         If .StreamNameLength Then
            .StreamName = PointerToStringW(lpBlock + 24, _
                                           .StreamNameLength)
         End If

         ' Increment count of streams.
         m_Count = m_Count + 1

         ' Bump up buffer pointer to next record.
         If .NextEntryOffset Then
            nIndex = nIndex + .NextEntryOffset
         Else
            Exit Do
```

```
         End If
      End With
   Loop
End Sub
```

The only way to know when you've iterated through the entire returned structure will be that the NextEntryOffset element of the current item is 0. Unlike files, which always contain a default unnamed ADS, it's possible that there are no ADS at all in folders, so you need to be prepared for this scenario. I chose to bail in the Refresh method, if the returned ioStatus structure indicated no data had been returned, as this allowed a single GetStreams routine to work with both files and folders.

Now, as I said, your application will require the SE_BACKUP_NAME privilege in order to enumerate ADS in folders. This may be an issue in least-privileged user situations. Luckily, file ADS provides nearly all the opportunity you'll likely need, so it will be the rather rare case indeed where this matters. In order to elevate your application to this privilege level, you need to invoke a few more APIs:

```
Private Function BackupPrivs(ByVal Enable As Boolean) As Boolean
   Dim hProcess As Long
   Dim DesiredAccess As Long
   Dim hToken As Long
   Dim tkp As TOKEN_PRIVILEGES
   Dim nRet As Long

   ' Cache a copy of priviliges as we found them.
   Static bup As TOKEN_PRIVILEGES

   ' Get psuedohandle to current process.
   hProcess = GetCurrentProcess()
   ' Ask for handle to query and adjust process tokens.
   DesiredAccess = TOKEN_QUERY Or TOKEN_ADJUST_PRIVILEGES
   If OpenProcessToken(hProcess, DesiredAccess, hToken) Then
      ' Get LUID for backup privilege name.
      If LookupPrivilegeValue(vbNullString, SE_BACKUP_NAME, _
            tkp.LUID) Then
         If Enable Then
            ' Enable the backup priviledge.
            tkp.PrivilegeCount = 1
            tkp.Attributes = SE_PRIVILEGE_ENABLED
            If AdjustTokenPrivileges(hToken, False, tkp, _
                  Len(bup), bup, nRet) Then
               BackupPrivs = True
            End If
         Else
            ' Restore prior backup privilege setting.
            If AdjustTokenPrivileges(hToken, False, bup, 0&, _
                  ByVal 0&, nRet) Then
               BackupPrivs = True
            End If
         End If
      End If
      ' Clean up token handle.
      Call CloseHandle(hToken)
   End If
End Function
```

I built this routine so it could work as a toggle, requesting and restoring privileges as needed. It does this by storing the previous privilege state in a static TOKEN_PRIVILEGES structure between calls. Please realize that this design is somewhat fragile, as there are no checks to insure you are going T-F-T-F-T-F rather than T-T-F-T-T-F, or following some other non-regular pattern. Worst case, though, that would simply mean your application would never back down its elevated privilege level, which probably won't be too dire a situation.

I said earlier that most normal VB file I/O methods will work just fine with ADS. One that doesn't is Kill, which returns an error 53 (File not found) if you pass it an ADS name. To overcome that limitation, I've added a KillStream method to the CStream class which uses the DeleteFile API directly:

```
Public Function KillStream(ByVal Index As Long) As Boolean
   If Index >= 0 And Index < m_Count Then
      ' Whack-a-Mole!  VB's Kill triggers an error 53.
      KillStream = CBool(DeleteFile(m_FileName & _
         m_Streams(Index).StreamName))
   End If
End Function
```

Just to show you how easy it is to use and abuse ADS, consider this simple little scenario:

```
Public Sub Main()
   Call WriteFile("C:\test.txt", "This is a normal data stream.")
   If WriteFile("C:\test.txt:MyADS", "This is an ADS.") Then
      Debug.Print ReadFile("C:\test.txt:MyADS")
   End If
End Sub

Public Function ReadFile(ByVal FileName As String) As String
   Dim hFile As Long
   On Error GoTo Hell
   hFile = FreeFile
   Open FileName For Binary As #hFile
      ReadFile = Space$(LOF(hFile))
      Get #hFile, , ReadFile
   Close #hFile
Hell:
End Function

Public Function WriteFile(ByVal FileName As String, _
      ByVal Text As String) As Boolean
   Dim hFile As Long
   On Error GoTo Hell
   hFile = FreeFile
   Open FileName For Output As #hFile
   Print #hFile, Text;
   Close #hFile
Hell:
   WriteFile = Not CBool(Err.Number)
End Function
```

What do you suppose shows up in the Immediate window? Yep, it's that easy! And note that this is without any sort of supporting CStreams class, or anything else I've written about, whatsoever. And it will be the rare user indeed that ever notices that extra ":MyADS:$DATA" stream tacked onto their formerly innocent little text file.

Be Careful Out There

If you're still having trouble imagining what you might do with ADS, take a look at your own hard drive. The Streams sample on my website provides a little tool that will recursively drill through a directory hierarchy, and provide you with a listing of all the files that contain ADS content.

Ever wonder how Windows knows to pester you about running that EXE you downloaded? Each one of them is tagged with an ADS named "Zone.Identifier", which becomes a tag-along INI file the system may query at will. How about those pesky thumbnails Windows wants to persist for all your image files? You may be shocked if you enumerate your "My Pictures" folder with my Streams tool. If you right-click on a text file, select Properties, and edit a few of the fields on the Summary tab, any guesses where that info is stored?

It's no wonder the security freaks are freaked out by these critters! These aren't your ordinary boogeyman-type threats. ADS are literally everywhere. They travel almost invisibly, only going away when their host file is transferred to a non-NTFS file system. And, very darn few users have any concept of ADS whatsoever. Handle with care.

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.

1105 Redmond Media Group

Copyright 1996-2009 1105 Media, Inc. View our Privacy Policy.