Class Persistence With VB

Learn how to persist the data in VB classes using simple file techniques.

by Stan Schultes and Chris Barlow

WHAT YOU NEED
Visual Basic 5.0 or 6.0
Access 97

Click & Retrieve Source CODE!

Ever since VB4 added support for classes, people have asked, "How do I persist the data in my classes?" You can use several approaches to implement class persistence. We'll take a look at a couple ways to persist the data in a class, then focus on a specific technique with some sample code.

First, what does the term "persist" mean? In the context of a class, persistence refers to the ability to save an instance of the class into some sort of storage so you can restore the contents later. The instance of the class itself is not saved, but the values contained in the class properties are saved. An instance of a class is often called an "object," but that term is so broadly used that to prevent confusion, we'll talk only of classes or instances of classes here.

If you have developed an ActiveX control, you might be aware that controls have always been able to persist their design properties. A control's design-time properties are stored in its CTX binary file through the PropertyBag object. VB6 has also added the ability for ActiveX components (DLLs or EXEs) to persist properties of their classes using the PropertyBag. (See the VB Help topic titled "Persisting a Component's Data" in the VB6 documentation for more information and sample code.)

Using the PropertyBag to persist design-time properties of ActiveX controls or ActiveX components is a powerful capability, and is useful when appropriate. To use the PropertyBag, however, you must develop ActiveX controls in VB5 or VB6, or build ActiveX components with VB6. Also, you need to write code to individually persist each property and to indicate changes to each property using a special function called PropertyChanged. This can be a lot of work for classes with a large number of properties.

You must first decide where to store the data when designing a class persistence mechanism. You can use any of the normal storage mechanisms—a database, files, the Registry, and so on. What you choose for a particular application depends on what specific problem you need to solve (and how much data is stored). Rather than trying to analyze a wide range of problems and tradeoffs, we'll look at a simple application you can build with VB5 or VB6.

The sample application illustrated here uses a simple technique for persisting class properties using binary files. The idea is to create a User Defined Type (UDT) containing elements corresponding to each property of the class. You can write the UDT to a binary file with a single Put command, and read it back in using a single Get command. This allows you to persist any data type as part of the UDT without having to do any manual data conversion. The source of data used in the sample application comes from a Microsoft accessory program: the Deluxe CD player database.

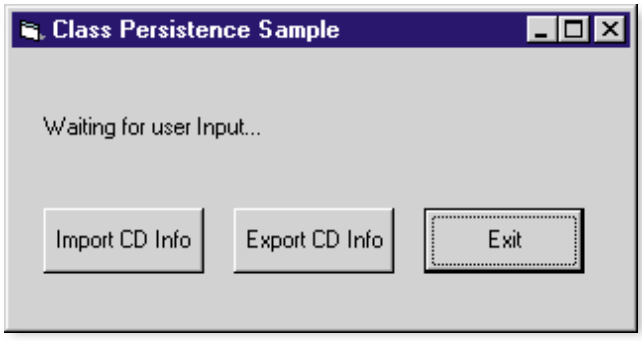

Figure 1 Simple Test Harness. Export CD records to object files, or import object files for a quick and easy replication utility.

Would You Like to Dance?

The Microsoft Plus! 98 CD contains an updated music CD player application called Deluxe CD that can download album title and track information from the Internet. Deluxe CD stores this information, along with some menu choice data, in an Access

97 database in the Windows directory. If you listen to music on several computers (especially if some have no Internet connection), the album information might not stay in sync between machines.

Although this is a somewhat contrived example, many business applications have this same data synchronization problem. The basic idea in this sample is to use classes to read data from a database and persist this data to files. You can transfer these binary "object files" to other computers whose databases are then updated. Sure, you could share the database on a server and do data replication in other ways, but you can use this simple technique even when there is no direct connection between computers. You can share files easily by compressing and e-mailing them.

You can find the code for this article on The Development Exchange (see the Download Free Code box at the end of this column for details). The sample is fairly simple, serving only to illustrate the class persistence technique discussed earlier. If you don't have the Deluxe CD player from the Plus! 98 CD, the download includes a small sample database along with some sample object files. Start a Standard EXE project in VB, and set the project name to Persist. Add a class module by selecting Add Class Module from the Project menu. Name the class CTitle in the Properties window. Add a module by selecting Add Module from the Project menu and name it modPersist. Then save the project. With Access 97, open the sample database (DeluxeCD.mdb) that comes with the downloaded code. The "real" database should be in your Windows directory (usually C:\Windows\ on Win95/98 and C:\Winnt\ on WinNT).

Look at the Titles table in design mode. In the UDT that represents this table, the text fields become Strings, the numeric fields become Long, and the binary field becomes a byte array. In the CTitle class, add a private Type declaration to store the class properties:

```
Private Type TitlesStruct
   'file header info
   m_sHDRID As String * 10
   m_sVersion As String * 5
   m_lFileDTS As Long
   'actual Title properties
   m_lTitleID As Long
   m_sArtist As String * 128
   m_sTitle As String * 128
   m_sCopyright As String * 128
   m_sLabel As String * 128
   m_sReleaseDate As String * 128
   m_lNumTracks As Long
   m_lNumMenus As Long
   m_sPlayList As String * 255
   m_byteTitleQuery(254) As Byte
End Type
Private m_yTitles As TitlesStruct
```

VB5, VB6 | CTitle Class FileSave Method

```
Public Function FileSave(ObjectsPath As String) As Boolean
'saves contents of the Title object to the specified path
  Dim sFileName As String
  Dim iFileNum As Integer
  Dim idx As Integer
  Dim oTemp As Object

  sFileName = ObjectsPath & "\" & TitleID & ".cdo"
  'delete existing file before creating the new one
  If Len(Dir(sFileName)) Then
    Kill sFileName
  End If

  iFileNum = FreeFile
  Open sFileName For Binary Lock Read Write As #iFileNum
  'write the structures out
  Put #iFileNum, , m_yProperties
  'now Tracks collection
  For idx = 1 To NumTracks
    Set oTemp = m_colTracks.Item(CStr(idx))
    oTemp.FileSave iFileNum
  Next
  Close #iFileNum
  Set oTemp = Nothing
  FileSave = True
End Function
```

Listing 1 Use the VB Put command through a UDT to persist Title properties to a binary file. Then call the FileSave method for each CTrack class instance in the collection, persisting each Track's data to the file.

The first three fields are not contained in the database, but serve as a simple object file header. Declare the database string fields as fixed-length strings in the UDT. This follows the database design, which allots a specific size to each text field. The binary database field is declared in the UDT as a byte array of 255 elements. The "m_" prefix on each element indicates these are private class member variables. For each of the class properties, create Let and Get property procedures such as these Title property procedures:

```
Public Property Let Title(ByVal _
   sData As String)
   m_yTitles.m_sTitle = Trim$(sData)
End Property

Public Property Get Title() As String
   Title = Trim$(m_yTitles.m_sTitle)
End Property
```

In the sample code, note that the TitleQuery property procedures contain extra code required to copy byte arrays. The downloaded code also contains error handling in each property procedure, which the preceding code omits for space reasons.

The database includes three additional tables: Batch, Menus, and Tracks. The Tracks table contains a list of

VB5, VB6 | CTitle Class DBLoad Method

```
Public Function DBLoad(ADOConn As Connection, & TitleIDInput As String) _
  As Boolean
'Loads from database to Title class
  Dim cm As New ADODB.Command
  Dim rs As New ADODB.Recordset
  Dim vArray As Variant
  Dim iCount As Integer
  Dim oTrack As CTrack
  On Error GoTo DBLoad_Error

  'find the specified Title
  cm.ActiveConnection = ADOConn
  cm.CommandText = "Select * From Titles Where " _
    & "TitleID = " & TitleIDInput
  cm.CommandType = adCmdText
  Set rs = cm.Execute

  'load the structure with the current Title record
  If Not rs.EOF Then
    TitleID = rs("TitleID")
    Artist = "" & rs("Artist")
    Title = "" & rs("Title")
    Copyright = "" & rs("Copyright")
    Label = "" & rs("Label")
    ReleaseDate = "" & rs("ReleaseDate")
    NumTracks = rs("NumTracks")
    NumMenus = rs("NumMenus")
    PlayList = "" & rs("PlayList")
    vArray = "" & rs("TitleQuery")
    TitleQuery = vArray
  End If

  'create a collection of Track structures that go with this Title
  cm.CommandText = "Select * From Tracks Where " & "TitleID = " & _
    TitleIDInput
  Set rs = cm.Execute
  If m_colTracks Is Nothing Then
    Set m_colTracks = New Collection
  Else
    ClearCollection m_colTracks
  End If
  Set oTrack = New CTrack
  Do While Not rs.EOF
    iCount = iCount + 1
    If oTrack.DBLoad(rs) Then
      m_colTracks.Add oTrack, CStr(iCount)
    Else
      MsgBox "Error adding Track " & CStr(iCount) & " for TitleID: " & _
        TitleIDInput
    End If
    rs.MoveNext
    Set oTrack = New CTrack
  Loop
  'adjust count in Titles
  If iCount <> NumTracks Then
    'log the difference?
    MsgBox "TitleID: " & TitleIDInput & ", NumTracks/Actual= " & _
      CStr(NumTracks) & " / " & CStr(iCount)
    NumTracks = iCount
  End If
```

Continued on page 74.

Listing 2 Use ADO to find and read a specific Title record from the database, then create a collection of Track class instances. This form of class persistence uses a database as the storage.

the Track names that correspond to each Title. The linking field is TitleID. The free downloaded code does not implement the Batch or Menus tables. Create another class named CTrack to manage the information for a track. Here's the UDT for this table (this structure also contains two file header fields that do not appear in the database itself):

```
Private Type TracksStruct
   'track header info
   m_sHDRID As String * 10
   m_sVersion As String * 5
   'actual track properties
   m_lTitleID As Long
   m_lTrackID As Long
   m_sTrackName As String * 128
End Type
Private m_yProperties As TracksStruct
```

Think about how Titles and Tracks are related. Each music CD has a single Title. A variable number of Tracks correspond to specific Titles. An easy way to represent this relationship in code is with a VB Collection. In the CTitle class, add a member collection to contain CTrack class instances:

```
Private m_colTracks As Collection
```

Add a Clear method to both the CTitle and CTrack classes so you can clear the UDTs and initialize the header fields. This allows you to reuse the class instances easily. In the Class_Initialize event for each class, call the Clear method to initialize the UDTs when you create instances of the class. The CTitle class also contains a ClearCollection method to empty the set of CTrack class instances associated with a Title.

Now for the two key methods that actually persist the class to a file: FileSave and FileLoad. The Title class FileSave method (see Listing 1) opens a file as binary (named as the TitleID with extension CDO, for CD Object). This code in FileSave writes the file:

```
Open sFileName For Binary Lock Read _
   Write As #iFileNum
'write the structures out
Put #iFileNum, , m_yProperties
'now Tracks collection
For idx = 1 To NumTracks
   Set oTemp = _
      m_colTracks.Item(CStr(idx))
   oTemp.FileSave iFileNum
Next
Close #iFileNum
```

Continued from page 73.

```
  DBLoad = True
DBLoad_Exit:
  Set rs = Nothing
  Set cm = Nothing
  Exit Function
DBLoad_Error:
  Call RaiseError(Err, mk_sClassName & ": DBLoad method" & "/" & _
    Err.Source)
  Resume DBLoad_Exit
End Function
```

Using the Put command, the entire UDT is written to the file in one fast operation, with no data conversion required. Call the FileSave method of each CTrack instance in the collection to write its own properties to the file:

```
Public Sub FileSave(FileNum As Integer)
'writes the Track object to a file
   Put #FileNum, , m_yProperties
End Sub
```

NumTracks in the Title class indicates how many tracks are in the collection. Reading a file in FileLoad is similar, except you read the contents using the Get command, and the Tracks collection is created. In a manner similar to saving, the UDTs are each loaded in one Get operation with no data conversion.

Have Data, Will Test

Microsoft provides a wide range of tools that allow you to access and manipulate data in databases, such as Data Access Objects (DAO), Remote Data Objects (RDO), and ActiveX Data Objects (ADO). Although any of these work, we used ADO for the sample Persist application. On the Project menu under References…, include a reference to the Microsoft ActiveX Data Objects 2.0 Library (by default in C:\Program Files\Common Files\System\ADO\msado15.dll). ADO 1.5 also works in VB5 with the code in this sample.

Use simple ADO functions to read and write data between the Access 97 database and classes. Both CTitle and CTrack have methods named DBLoad (see Listing 2) and DBSave. The modPersist module contains ExportAlbums and ImportAlbums functions that handle class instantiation, database management tasks, and searching logic.

Finally, to test the classes, add a form to the project (see Figure 1), three command buttons (cmdExit, cmdExport, and cmdImport), and one label control (lblStatus). Add code to the event procedure for each button (download Listing 3 from DevX; see the Download Free Code box for details). Notice write-only property procedure Status in Form1. This allows you to set the label control's caption from outside the form without referring directly to the label control by name. The database and object file paths passed to the ExportAlbums and ImportAlbums functions default to App.Path, a built-in VB property. App.Path returns the source code directory path while in VB design mode. Run the application and step through the code to see how simple class persistence works.

Back to the two tables we ignored in the Deluxe CD database: Menus and Batch. The Menus table contains menu items appearing under the Internet Options button on the Deluxe CD player application. We left it as an exercise for you to implement class persistence on the Menus table. Hint: Add another class called CMenu, and add another collection to the CTitle class. The CMenu class is similar to the CTrack class (but be sure to handle the binary MenuQuery field properly). The Batch table is used for downloading track information; you can ignore it. A viewer application for the exported binary object files would be another interesting project.

You can often modify existing code easily to use this class persistence technique by adding Type and End Type statements around the declarations for private member variables in a class (and referring to the new Type in the property Let and Get procedures). As you might guess, this method won't work for classes that expose their properties through Public variables instead of property Let and Get procedures (but you don't do that in your code, right?).

Now you've seen a way to replicate information between databases using a simple and effective class persistence technique. The FileSave and FileLoad class methods are the key, storing the properties of a class in UDTs, allowing a quick and easy way of persisting the properties to files for later retrieval. You can exchange these files easily between computers to provide a simple form of data replication. Another use would be to store the files as backups of database records or even of database transactions. You could use the transaction files to build an offline or backup copy of the database if desired. Your imagination is the limit. VBPJ

About the Author

Stan Schultes is an experienced project manager and VB developer, and has spoken on VB at Microsoft's DevDays conference. Stan is an MCP in Visual Basic and holds a degree in computer engineering from Purdue University. Reach Stan at Stan@VBExpert.com.

Chris Barlow, a recognized expert in Internet, Web, messaging, and applications development, speaks at VBITS, Tech•Ed, and DevDays, and has been featured in two Microsoft videos. Chris holds degrees from Harvard Business School and Dartmouth College. Reach him at Chris@VBExpert.com or on the Web at www.VBExpert.com.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from www.vbpj.com.

To get the free code for this entire issue, click on Locator+, the rightmost option on the menu bar at the top of the VBPJ home page, and type **VBPJ0399** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, including a listing containing the modPersist module's ExportAlbums function, plus a sample application illustrating a simple technique of class persistence.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0399GS** into the Locator+ field. The bonus code includes all the free code described above, plus a fully developed utility for doing synchronization of Deluxe CD Player databases between two computers.