Serve Data to Your Clients

by Stan Schultes and Chris Barlow

It's simple to use a Web-server method to share information across systems.

Click & Retrieve Source CODE!

As a VB developer, you frequently hear terms that describe how data is shared between systems in your enterprise. For example, you've probably heard the terms "client/server," "three-tier," and "n-tier." Don't be confused by terminology—these words simply describe how applications are partitioned and how they communicate with one another, such as passing data between parts of an application that might run on more than one computer. With all the hype surrounding these terms and all the potential ways of sharing data, how do you get started?

As with most programming tasks, you can share information between systems in many ways. Each method has its pros and cons, and each can work in different applications, depending on what you want to accomplish (see Table 1). At my company, SunOpTech, we build Windows-based, enterprise-class information and decision-support systems. The architecture of several of SunOpTech's systems uses temporal, append-only databases to store data. This architecture time-stamps and stores transactions in local databases that keep all transactions over time. The database therefore contains the complete history of any activity.

The key to implementing the temporal database is guaranteeing that each database update occurs in the proper order. While this is straightforward in a single database, the process quickly becomes complex when transactions simultaneously occur on multiple servers located around the world. When the data is replicated between servers, how can you be sure the transactions are properly ordered?

SunOpTech uses a common time-base approach across locations. Time-stamps generated for all transactions are in Universal Coordinated Time (UTC), also known as Greenwich Mean Time. With UTC, you can reference any transaction to any other using the same time system, no matter where and when in the world the two transactions occurred. Servers in different locations synchronize their time to a common time base on the Internet using one of the various Internet time protocols (for more information, see http://www.eecis.udel.edu/~mills/ntp/servers.html).

So, what's the best way to share information across systems? Microsoft provides a Windows API call, NetRemoteTOD (in NetAPI32.dll), that retrieves the time from another machine across the network. Unfortunately, Microsoft has included this Win32 API function only on Windows NT—you won't find it in either Windows 95 or Windows 98. So, only your clients using Windows NT can use this system call.

Method	Pros	Cons
Windows API Calls	Very fast	Might not be supported on all platforms
DCOM	Flexible	Might be difficult to configure
Web Server	Available everywhere	Might not be able to implement consistently across products
Sockets	Services common across platforms	Difficult to program without third-party tools
Microsoft Transaction Server (MTS)	Bundled with NT	More difficult to configure
Microsoft Message Queue (MSMQ)	Guaranteed delivery with multiple clients	Still early in release cycle
Third-Party Tools	Easy to set up and use	Might pay license fees or royalties

TABLE 1 ***Sharing Information Between Systems.*** *Microsoft Windows offers a number of methods to share information between systems. Here are some of the pros and cons.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications, including the ObjectBank and ObjectJob systems. Chris holds degrees from Harvard Business School and Dartmouth College, where he worked with Drs. Kemeny and Kurtz on the Basic language. Reach Chris by e-mail at Chris@VBExpert.com.

Stan Schultes is lead developer at SunOpTech, where he is responsible for development and worldwide support of the ObjectBank and ObjectOrder products. Stan has 20 years of experience in the computing field, most spent with a Fortune 200 company, where he developed manufacturing systems and was a corporate technology consultant. Stan holds a degree in computer engineering from Purdue University. Reach Stan at Stan@VBExpert.com.

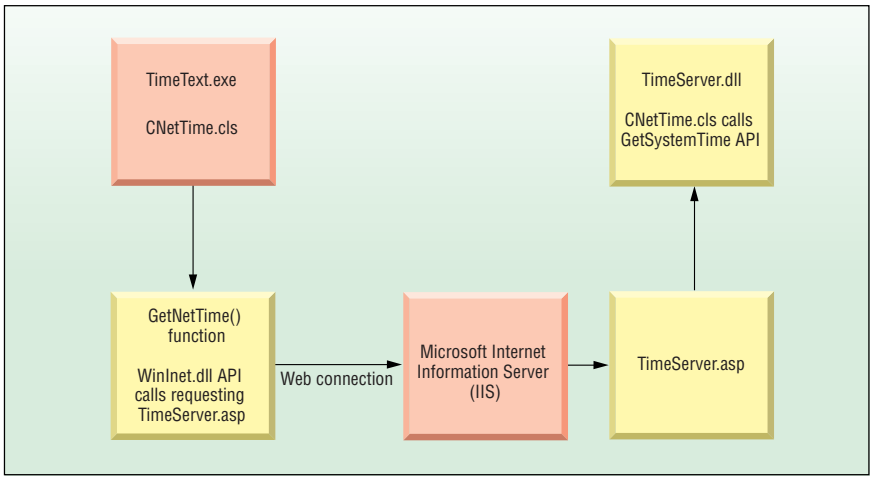

FIGURE 1 ***Web Server Overview.*** *A Web server is the basis for communicating between systems. The Web server returns the system time in Universal Coordinated Time (UTC) to the client.*

I'll show you how to solve this problem using the Web-server approach, but it's only one of several ways of getting the job done. This approach is simple to implement, reasonably accurate, and based on technology that most VB shops already have in place (see Figure 1). The client software uses the WinInet API to request the time from a Web server common to the local clients. The server, running Microsoft's Internet Information Server (IIS), executes an Active Server Pages (ASP) script that calls a time-service ActiveX DLL to return the time of the current server in UTC. The client simply parses the returned time string to get the time-stamp information.

This solution is simpler and more flexible than other methods. It allows a client application to connect to the server through an Internet (or intranet) connection, regardless of how the connection is made and

VBA5

```
Option Explicit
'Copyright © 1992-1998, SunOpTech®, Ltd., All Rights
'Reserved
'Class Name: cNetTime (NetTime.cls)
'Class Description: Network Time class
'Author: Stan Schultes
'Date Created: 4/30/98
'
'private class members
Private m_iMSec As Integer

'Internet connection declarations
Private Const INTERNET_OPEN_TYPE_PRECONFIG = 0
'uses Registry config info
Private Const INTERNET_FLAG_EXISITING_CONNECT = _
  &H20000000
Private Const INTERNET_FLAG_RELOAD = &H80000000

Private Declare Function InternetOpenUrl Lib _
  "wininet.dll" Alias "InternetOpenUrlA" _
  (ByVal hInternetSession As Long, _
  ByVal lpszUrl As String, ByVal lpszHeaders As _
  String, ByVal dwHeadersLength As Long, ByVal _
  dwFlags As Long, ByVal dwContext As Long) As Long
Private Declare Function InternetOpen Lib "wininet.dll" _
  Alias "InternetOpenA" (ByVal sAgent As String, _
  ByVal lAccessType As Long, ByVal sProxyName As _
  String, ByVal sProxyBypass As String, ByVal lFlags _
  As Long) As Long
Private Declare Function InternetReadFile Lib _
  "wininet.dll" (ByVal hFile As Long, ByVal sBuffer As _
  String, ByVal lNumBytesToRead As Long, _
  lNumberOfBytesRead As Long) As Integer
Private Declare Function InternetCloseHandle Lib _
  "wininet.dll" (ByVal hInet As Long) As Integer

Public Function GetNetTime(URL As String)
'returns the Server Time via HTTP, sets other properties
Dim sBuffer As String * 4096
Dim sReturn As String
Dim lNumBytes As Long
Dim lSession As Long
Dim lFile As Long
Dim bReadOK As Boolean

lSession = InternetOpen("NetTime", _
  INTERNET_OPEN_TYPE_PRECONFIG, _
  vbNullString, vbNullString, 0)
lFile = InternetOpenUrl(lSession, URL, vbNullString, 0, _
  INTERNET_FLAG_EXISITING_CONNECT Or _
  INTERNET_FLAG_RELOAD, 0)
If lFile Then
  Do
     bReadOK = InternetReadFile(lFile, sBuffer, _
        Len(sBuffer), lNumBytes)
     If lNumBytes Then
        sReturn = sReturn & Left$(sBuffer, lNumBytes)
     End If
  Loop While bReadOK And lNumBytes > 0
  InternetCloseHandle (lFile)
  If Len(sReturn) Then
     'get time as variant date & fill Milliseconds
     GetNetTime = ParseTime(sReturn)
  Else
     'log error & get time locally
  End If
Else
  'log error & get time locally
End If
End Function

Public Property Get Milliseconds() As Integer
  Milliseconds = m_iMSec
End Property

Private Function ParseTime(ASPReturn As String) As Date
'parses time values out of string returned from ASP
Dim iPos As Integer, iPos2
Dim sTimeStr As String
If Len(ASPReturn) Then
  'find the TimeUTC & EndTime values in the returned
  'string
  iPos = InStr(UCase$(ASPReturn), "TIMEUTC=")
  iPos2 = InStr(UCase$(ASPReturn), "=ENDTIME")
  If iPos Then
     'sTimeStr will be in form "5/1/98 16:43:18 PM,789"
     ' where 789 is the number of Milliseconds
     sTimeStr = Mid$(UCase$(ASPReturn), iPos + 8, _
        iPos2 - iPos - 8)
     iPos = InStr(sTimeStr, ",")
     ParseTime = CDate(Left$(sTimeStr, iPos - 1))
     m_iMSec = CInt(Mid$(sTimeStr, iPos + 1))
  End If
Else
  ParseTime = 0
  m_iMSec = 0
End If
End Function
```

LISTING 1 ***NetTime Class.*** *This class is the core of the client end of this example. It encapsulates the WinInet calls and parses the HTML returned from the server-side ASP script. Time in UTC is returned by GetNetTime(), and the Milliseconds property is set.*

where the two ends are located. It also works with any Windows 32-bit operating system—Windows 95, Windows 98, NT3, NT4, or NT5—on both the client and server end, without any fancy footwork.

You can download the code from the free, Registered Level of The Development Exchange (see the Code Online box at the end of the column for details). I'm assuming you're running IIS and that you have the privileges to register an ActiveX DLL and put an ASP page on the Web server. For demonstration purposes, you can do this on a Win95 machine using the Microsoft Personal Web Server. If you don't have these privileges, you can try out the client portion of the code and set it to access the time server set up at http://www.vbexpert.com/timeserver.asp.

AND NOW FOR THE CODE

This sample code features three parts. To build this component, you need either the Professional or Enterprise Edition of VB5. Starting with the server end, you use an ActiveX DLL with one method and one property. The GetServerTime Method returns a Variant of type Date representing the server's current time as a UTC value. The Milliseconds property provides a way of getting more than one second of resolution from the DLL.

Start by creating an ActiveX DLL project, and call it TimeServerDLL. Name the default class CServerTime and add this code to the Declarations section:

```
Option Explicit
Private Type SYSTEMTIME
  iYear As Integer
  iMonth As Integer
  iDayOfWeek As Integer
  iDay As Integer
  iHour As Integer
  iMinute As Integer
  iSecond As Integer
  iMilliseconds As Integer
End Type
Private Declare Sub GetSystemTime Lib _
  "kernel32" (pSystemTime As _
  SYSTEMTIME)
Private m_iMSec As Integer
```

This declares the GetSystemTime Win32 API call to VB and defines the SYSTEMTIME data structure filled by GetSystemTime. The m_iMSec variable is a private class member that stores additional time resolution as milliseconds. Next, add the public function GetServerTime, which handles the main task of the class:

```
Public Function GetServerTime()
'Returns the Server time as a variant
'Date value, and sets the Millisecond
'property
  Dim pTime As SYSTEMTIME
  Dim vDateTime As Variant
  GetSystemTime pTime
  'API call for time from the server
  'DateSerial and TimeSerial return
  'Variant Dates
  vDateTime = DateSerial( _
     pTime.iYear, pTime.iMonth, _
     pTime.iDay) + TimeSerial( _
     pTime.iHour, pTime.iMinute, _
     pTime.iSecond)
  m_iMSec = pTime.iMilliseconds
  GetServerTime = vDateTime
End Function
```

Finally, add a read-only property to return the milliseconds value. A read-only property has only the Property Get procedure, omitting the Property Let:

```
Public Property Get Milliseconds() _
  As Integer
  Milliseconds = m_iMSec
End Property
```

Yes, that's all it takes to create an ActiveX DLL that returns the server's time in UTC. Now, build the DLL, specifying the file name as TimeServer.dll in the Make Project dialog box. When you're done, copy the DLL to the server into path c:\winnt\system32. Replace this path with the server's Windows System directory name. Register the DLL on the server using RegSvr32. The simplest way of doing this is to use the Windows Run command under the Start menu, and type this code in the combo box:

```
RegSvr32 timeserver.dll
```

VBSCRIPT IN THE MIDDLE

Active Server Pages are extensions to your Web server that allow a scripting language to execute on the server side of your Web connection. This scripting language (the default is VBScript) allows you to create custom, dynamic content easily on your Web pages without exposing the code you use to generate the HTML output. Create the TimeServer.asp file with these lines:

```
<%
'** TimeServer.asp-VBPJ Getting
'**Started, August, 1998**
Set ts = Server.CreateObject( _
  "TimeServerDLL.cServerTime")
%>
TimeUTC=<% =ts.GetServerTime %>,<% _
  =ts.Milliseconds %>=EndTime
```

This simple VBScript does the job efficiently. The <% and %> symbols delineate VBScript that runs when the page is accessed—telling the server to execute this code to create the HTML page returned to the client. Variable *ts* is set as a reference to the TimeServer DLL. In the last line, =ts.GetServerTime returns a Variant Date that appears as a formatted date string, and =ts.Milliseconds returns an integer. TimeUTC=, the comma, and =EndTime appear as text and are used for parsing on the client end.

Put the TimeServer.asp file on your Web site. For demonstration purposes, I put mine into d:\inetpub\wwwroot—my root Web site folder. To test your new ASP script, start your favorite browser and enter the URL http://myserver/timeserver.asp, where "myserver" is replaced by the name of your Web server (see Figure 2). The first time you

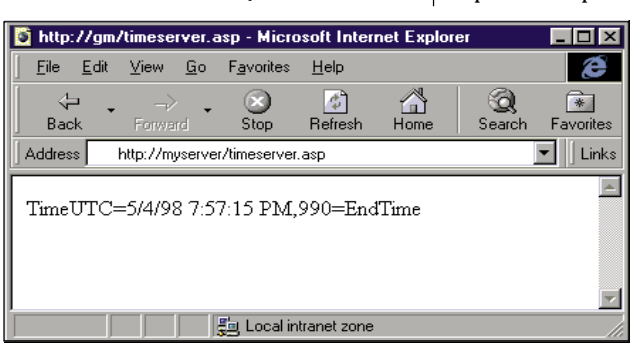

FIGURE 2 ***Browser Returns Value.*** *Internet Explorer 4 displays the time string returned from the TimeServer.asp script on the Web server.*

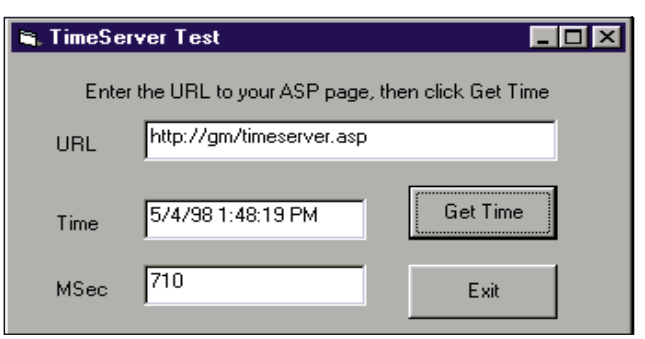

FIGURE 3 ***Test Harness at Work.*** *The TimeTest program provides a simple test host for the NetTime class. It displays the results returned from a request for system time from the Web server.*

request this page, you'll experience a slight delay while the DLL loads. If you refresh the page, it quickly returns the new time string. After the last access, IIS keeps the DLL loaded for 30 minutes or so by default to improve the response of successive requests.

Now it's time for the client code. It uses the WinInet API (in WinInet.dll) to communicate between a VB program and a Web server, so it's the most complex part of the example.

The meat of the client-application code is NetTime.cls—a class that wraps the WinInet functionality. Start by opening a new standard EXE project, adding a class, and naming it NetTime. Save the project as TimeTest.vbp (see Listing 1). The declarations in the class header make four WinInet functions available to the VB program. Public function GetNetTime does the bulk of the communication work to the server with this code fragment:

```
lSession = InternetOpen("NetTime", _
  INTERNET_OPEN_TYPE_PRECONFIG, _
  vbNullString, vbNullString, 0)
lFile = InternetOpenUrl(lSession, URL, _
  vbNullString, 0, _
  INTERNET_FLAG_EXISITING_CONNECT Or _
  INTERNET_FLAG_RELOAD, 0)
If lFile Then
  Do
     bReadOK = InternetReadFile( _
        lFile, sBuffer, _
        Len(sBuffer), lNumBytes)
     If lNumBytes Then
        sReturn = sReturn & _
           Left$(sBuffer, lNumBytes)
     End If
  Loop While bReadOK And lNumBytes > 0
  InternetCloseHandle (lFile)
'{code omitted - see Listing 1}
End If
```

First, an INet connection is initialized with InternetOpen, then the InternetOpenURL function specifies the ASP page address. Note the use of the INTERNET_FLAG_RELOAD value, which guarantees that a new time value comes back on each request. The InternetReadFile function is called repeatedly until the HTML stream returned by the ASP page is exhausted. To finish the session, the INet handle is closed with InternetCloseHandle.

Next, the ParseTime function processes the HTML stream returned from the ASP page. ParseTime finds the time string, first looking for the "TimeUTC=" and "=EndTime" markers, then looking for the comma separating the time value from the milliseconds value. ParseTime returns the time value as a Variant Date and sets the Milliseconds property.

BUILD A TEST HARNESS

Testing the NetTime class is easy. On the Form1 default form in the TimeTest project, add three text boxes and a command button (see Figure 3). The URL for TimeServer.asp goes into Text1, Text2 displays the time value, and Text3 displays the milliseconds value. Put this code in the Command1_Click function:

```
Private Sub Command1_Click()
  Dim nt As New CNetTime
  Text2 = nt.GetNetTime(Text1)
  Text3 = nt.Milliseconds
End Sub
```

The *nt* variable is declared to be a new object of type CNetTime. Executing the GetNetTime function with the URL of the ASP script should return a time value. Try clicking several times in succession to see the time value change. The CNetTime class is a source component that you can reuse in any project that needs to obtain the time from a common server machine.

Now that you've seen how easy it is to write client code to request information from your Web server using ASP, think of what else you can do with this technique. For example, you can easily implement a disk-space monitor across a number of machines. Write the ActiveX DLL to return total and free disk space from the drives on the machine by using another Windows API call. You can collect disk space from a network of computers monitored by a central application. Each monitored machine needs a Web server, the registered DLL, and an ASP script that calls the DLL.

Code Online

You can find all the code published in this issue of VBPJ on The Development Exchange (DevX) at http://www.vbpj.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Serve Data to Your Clients

Locator+ Codes

Listings for the entire issue, plus the code associated with this column (free Registered Level): VBPJ0898

Listings for this article only, the code mentioned above, plus two extra samples (Windows Sockets and DCOM) and a text document that helps with setup (subscriber Premier Level): GS0898