Name the Domain

by Karl E. Peterson

Click & Retrieve Source CODE!

API CALLS FACILITATE NAME QUERIES

Q Under Windows NT, I need to obtain the machine name and the domain name from within my application. Which API function(s) should I use to do that?
—Allen Khaligh, Herndon, Virginia

A Networking can cause much grief under VB. Fortunately, the API support in Windows NT is far superior to that in Windows 95, where the choices are grim at best. NT exposes a series of around 70 NetApi functions used to query and control most aspects of networking. However, NT offers nearly all the functions in Unicode-only versions, which can be frustrating due to VB's inability to directly pass Unicode strings to a DLL function.

Two API calls will help: NetWkstaGetInfo and NetWkstaGetUserInfo. I've written a single class module, CNetWksta, to encapsulate both. This class offers a Refresh method, which calls the two APIs and populates structures, which are exposed through read-only properties (see Listing 1). The complete CNetWksta class and a sample showing how to use it are available on the Registered Level of The Development Exchange. Premier Club members can download a similar class that fully exercises the NetUserGetInfo API, which obtains all available information for any given user on any server (see the Code Online box at the end of the column for details).

The first trick to making these calls is avoiding VB's nasty habit of converting all passed strings to ANSI on the way out, and reconverting them to Unicode on the way back. Notice that in the Declares, the "string" parameters such as lpServer are declared with As Any. The strategy is to maintain Unicode-only strings by holding the data in Byte arrays, which VB doesn't attempt to convert. In this example, you want information only for the local machine, so you can pass a Null pointer rather than specify a remote server.

The next hurdle is that the NetApi functions are pointer-oriented. NetWkstaGetInfo returns a pointer in the lpBuffer parameter to a structure it creates containing the requested information (based on the value passed in the Level parameter). To convert this pointer to a true VB structure, copy it from the location pointed to by lpBuffer. Complicating matters are a number of structure elements that are themselves pointers to Unicode string data.

My solution is to declare a pair of parallel structures for each API. One structure contains all Long elements, and the other contains a mixture of Long and String elements. After a successful call to NetWkstaGetInfo or NetWkstaGetUserInfo, use RtlMoveMemory (here aliased as CopyMem) to copy the data from the buffer created by the API into the structure containing all Longs. Finally, populate the mixed structure by directly assigning Long elements, and calling a special-purpose VB routine to copy the Unicode string data from memory.

The PointerToStringW function (see Listing 1) accepts a pointer to a Unicode string as input, and returns that string data as a VB String variable. If the passed pointer is valid (nonzero), call lstrlenW to determine the length of the string data pointed to. Because Unicode consists of two bytes per character, multiply this character length by two to create a Byte array long enough to receive the string data. Call CopyMem again to move the string data from memory into the Byte array. Because the string data is already Unicode, you can assign it directly to a VB String variable. Therefore, assign the Byte array as the return value of PointerToStringW.

Finally, after obtaining all the required data, the buffer created by the API call needs to be released. Although most API functions require a buffer created ahead of time, which they then fill, the NetApi functions often create the buffer themselves and return a pointer to it. In this case, memory might remain allocated long after its need has passed. Therefore, you must call NetApiBufferFree after copying the buffer's contents.

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the Visual Basic Programmer's Journal *Technical Review and Editorial Advisory boards. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls. Karl coauthored* Visual Basic 4 How-To *from Waite Group Press and contributes to various journals. Online, he's a Microsoft Developer MVP and a section leader for both* VBPJ *online forums. Contact Karl at karl@rtc.wa.gov.*

This is your forum for addressing the intricacies of Visual Basic. Send your questions, clever tips, and techniques. Visual Basic Programmer's Journal *will pay $25 for any submission, tip, or question we print. If your submission includes code, please send it electronically. Please include your mailing address with your submission. Mail submissions to Q&A Columnists, c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. CompuServe: 74774,305. Internet: vbpjedit@fawcette.com.*

VB4 32-bit VB5

```
'
' Win32 NetAPIs.
'
Private Declare Function NetWkstaGetInfo Lib _
  "Netapi32.dll" (lpServer As Any, _
  ByVal Level As Long, lpBuffer As Any) As Long
Private Declare Function NetWkstaUserGetInfo Lib _
  "Netapi32.dll" (ByVal reserved As Any, _
  ByVal Level As Long, lpBuffer As Any) As Long
Private Declare Function NetApiBufferFree Lib _
  "Netapi32.dll" (ByVal lpBuffer As Long) As Long
'
' Data handling APIs
'
Private Declare Sub CopyMem Lib "kernel32" Alias _
  "RtlMoveMemory" (pTo As Any, uFrom As Any, _
  ByVal lSize As Long)
Private Declare Function lstrlenW Lib "kernel32" _
  (ByVal lpString As Long) As Long
Private Declare Function lstrcpyW Lib "kernel32" _
  (lpString1 As Byte, ByVal lpString2 As Long) As Long

Private Type WKSTA_INFO_102
  wki102_platform_id As Long
  wki102_computername As Long
  wki102_langroup As Long
  wki102_ver_major As Long
  wki102_ver_minor As Long
  wki102_lanroot As Long
  wki102_logged_on_users As Long
End Type

Private Type WkstaInfo102
  PlatformId As Long
  ComputerName As String
  LanGroup As String
  VerMajor As Long
  VerMinor As Long
  LanRoot As String
  LoggedOnUsers As Long
End Type

Private Type WKSTA_USER_INFO_1
  wkui1_username As Long
  wkui1_logon_domain As Long
  wkui1_oth_domains As Long
  wkui1_logon_server As Long
End Type

Private Type WkstaUserInfo1
  UserName As String
  LogonDomain As String
  OtherDomains As String
  LogonServer As String
End Type

Private Const NERR_Success As Long = 0&
'
' Member variables
'
Private m_Wks As WkstaInfo102
Private m_User As WkstaUserInfo1
Private m_IsWinNT As Boolean

' ****************************************************
'  Public Methods
' ****************************************************
Public Sub Refresh()
  Dim lpBuffer As Long
  Dim nRet As Long
  Dim wki As WKSTA_INFO_102
  Dim wkui As WKSTA_USER_INFO_1
  '
  ' These functions only exist in Windows NT!!!
  '
  If Not m_IsWinNT Then Exit Sub
  '
  ' Obtain workstation information
  '
  nRet = NetWkstaGetInfo(ByVal 0&, 102&, lpBuffer)
  If nRet = NERR_Success Then
     '
     ' Transfer data to VB-friendly structure
     '
     CopyMem wki, ByVal lpBuffer, Len(wki)
     m_Wks.PlatformId = wki.wki102_platform_id
     m_Wks.ComputerName = _
        PointerToStringW(wki.wki102_computername)
     m_Wks.LanGroup = _
        PointerToStringW(wki.wki102_langroup)
     m_Wks.VerMajor = wki.wki102_ver_major
     m_Wks.VerMinor = wki.wki102_ver_minor
     m_Wks.LanRoot = _
        PointerToStringW(wki.wki102_lanroot)
     m_Wks.LoggedOnUsers = wki.wki102_logged_on_users
     '
     ' Clean up
     '
     If lpBuffer Then
        Call NetApiBufferFree(lpBuffer)
     End If
  End If
  '
  ' Obtain user information for this workstation
  '
  nRet = NetWkstaUserGetInfo(0&, 1&, lpBuffer)
  If nRet = NERR_Success Then
     '
     ' Transfer data to VB-friendly structure
     '
     CopyMem wkui, ByVal lpBuffer, Len(wkui)
     m_User.UserName = _
        PointerToStringW(wkui.wkui1_username)
     m_User.LogonDomain = _
        PointerToStringW(wkui.wkui1_logon_domain)
     m_User.OtherDomains = _
        PointerToStringW(wkui.wkui1_oth_domains)
     m_User.LogonServer = _
        PointerToStringW(wkui.wkui1_logon_server)
     '
     ' Clean up
     '
     If lpBuffer Then
        Call NetApiBufferFree(lpBuffer)
     End If
  End If
End Sub

' ****************************************************
'  Private Methods
' ****************************************************
Private Function PointerToStringW(lpStringW As Long) _
  As String
  Dim buffer() As Byte
  Dim nLen As Long

  If lpStringW Then
     nLen = lstrlenW(lpStringW) * 2
     If nLen Then
        ReDim buffer(0 To (nLen - 1)) As Byte
        CopyMem buffer(0), ByVal lpStringW, nLen
        PointerToStringW = buffer
     End If
  End If
End Function
```

LISTING 1 ***Obtain Workstation Information Through Networking APIs.*** *A class module wraps NetWkstaGetInfo and NetWkstaGetUserInfo to obtain information such as domain and user names. Public read-only property procedures (not shown) offer elements of the WkstaInfo102 and WkstaUserInfo1 types to your application.*

By the way, to answer your question, the domain name is somewhat hidden in the WkstaInfo102 structure. It goes by the name LanGroup with this API.

Q AVOID INVALID NULL ERROR

What do I do when this code generates the error "Invalid use of Null"?

```
txtData(0).Text = MyRS("Topic")
```

If the field in the Access database contains a Null, I get an error when trying to assign it to the Text property of a text box. Should I modify the database, or handle the error in my code?
—Patrick Kelly, received by e-mail

A You'll love to hear that not only is this incredibly easy to fix, but also that this problem has bitten nearly everyone who's ever worked with Access databases. Although the problem has been around as long as VB3 has, there's a constant parade of folks who've never run into it. You could attack it with lots of nasty code, checking for Nulls in every field. Or, you could be sneaky and take advantage of what VB does when you perform operations on Nulls. Simply put, concatenate a Null string to the Null, and you end up with a Null string. Or, simpler still:

```
txtData(0).Text = MyRS("Topic") & ""
```

That's all there is to it.

Q FIVE, 10, 15, 20 . . .

My program is for the steel-detailing industry and is used to count bolts. I need to add a feature to round up all totals to the nearest five. For example, say you arrive at a total of 471 bolts of a certain size. I need to round this figure up to 475.
—David Nelms, Angier, North Carolina

A The answer to this lies in two VB functions: one under appreciated and the other virtually unknown. I'll look at the solution first, then pick apart why it works:

```
Function RoundUpToFives(ByVal n As Long) As Long
   RoundUpToFives = ((n \ 5) + Sgn(n Mod 5)) * 5
End Function
```

Start by doing an integer divide on the original number by five. This gives you the total number of times that five goes into the original without any remaining fraction. Then, use the Mod operator to determine what the remainder would be in such an integer division. The potential remainder values range from zero to four when dividing by five. This is where the Sgn function comes in. Sgn returns –1 for negative numbers, 0 for zero, and 1 for positive numbers. If the original number is evenly divisible by five, the Mod result is 0, so adding the Sgn of this latter result and then multiplying by five brings you right back where you started. However, if there is any remainder, that is your clue that you need to round up to the next multiple of five. If the Mod result is 1 through 4, adding the Sgn of this value before multiplying by five bumps up the result to the next multiple of five. Simple in retrospect, but a bit of a head-scratcher at first glance.

Q TILING THE BACKGROUND

I'm trying to design forms in VB5 from my Access 97 tables. I have a nifty little background I'd like to use on

VB4 32-bit VB5

```
Public Sub TileBlt(ByVal hWndDest As Long, _
  ByVal hBmpSrc As Long)
  '
  ' 32-Bit Tiling BitBlt Function
  ' Written by Karl E. Peterson, 9/22/96.
  ' Tiles a bitmap across the client area of
  ' destination window.
  '
  ' Parameters
*****************************************************
  '  hWndDest:      hWnd of destination
  '  hBmpSrc:       hBitmap of source
  '
****************************************************
  '
  Dim bmp As BITMAP         ' Header info for passed
                            ' bitmap handle
  Dim hDCSrc As Long        ' Device context for source
  Dim hDCDest As Long       ' Device context for
                            ' destination
  Dim hBmpTmp As Long       ' Holding space for temporary
                            ' bitmap
  Dim dRect As RECT         ' Holds coordinates of
                            ' destination rectangle
  Dim Rows As Long          ' Number of rows in
                            ' destination
  Dim Cols As Long          ' Number of columns in
                            ' destination
  Dim dX As Long            ' CurrentX in destination
  Dim dY As Long            ' CurrentY in destination
  Dim i As Long, j As Long
  '
  ' Get destination rectangle and device context.
  '
  Call GetClientRect(hWndDest, dRect)
  hDCDest = GetDC(hWndDest)
  '
  ' Create source DC and select passed bitmap into it.
  '
  hDCSrc = CreateCompatibleDC(hDCDest)
  hBmpTmp = SelectObject(hDCSrc, hBmpSrc)
  '
  ' Get size information about passed bitmap, and
  ' Calc number of rows and columns to paint.
  '
  Call GetObj(hBmpSrc, Len(bmp), bmp)
  Rows = dRect.Right \ bmp.bmWidth
  Cols = dRect.Bottom \ bmp.bmHeight
  '
  ' Spray out across destination.
  '
  For i = 0 To Rows
     dX = i * bmp.bmWidth
     For j = 0 To Cols
        dY = j * bmp.bmHeight
        Call BitBlt(hDCDest, dX, dY, bmp.bmWidth, _
           bmp.bmHeight, hDCSrc, 0, 0, SRCCOPY)
     Next j
  Next i
  '
  ' and clean up
  '
  Call SelectObject(hDCSrc, hBmpTmp)
  Call DeleteDC(hDCSrc)
  Call ReleaseDC(hWndDest, hDCDest)
End Sub
```

LISTING 2 ***Tile a Bitmap Across the Background.*** *This code enables users to point to a destination window for operations such as pasting information from the clipboard. Apply the same technique in 16-bit VB apps by modifying the API declarations for that platform. Due to space limitations, the API declarations are available on the Registered Level of The Development Exchange.*

all the forms for unity's sake. However, I can't seem to get the image to tile across the entire form. Please don't tell me I must make backgrounds the size that I want them and then place them as pictures for either the form or image. Is there a tiling or clipping feature that I'm just not seeing in the Properties menu?
—Nick DeVore, received on The Development Exchange BBS

A No, you're not missing anything. There isn't any built-in tiling functionality in VB. That isn't to say it isn't doable, though. Tiling an image across a window requires only that you loop through as many iterations as required to fully cover the window with your image. VB provides the PaintPicture method as one way to transfer the image, but I prefer to go straight to the API and call the BitBlt function. This opens more possibilities, including building the image in memory rather than using a stored image.

I wrote a function some time ago for this purpose (see Listing 2). Notice the only parameters required are the window handle of the destination and the bitmap handle of the source. Asking for the bitmap handle rather than the window handle of the source lets you store the source in a Picture object instead of a Picture control. Picture objects consume fewer resources and don't need to be attached to a form, so they're preferable when they serve the intended purpose.

Don't be put off by the number of API calls used in this routine, because it's self-contained. To use it with either VB4 or VB5, pass the Handle property of any Picture object, including the Picture property of Picture controls:

```
Call TileBlt(Me.hWnd, Picture1.Picture.Handle)
```

TileBlt works by creating a compatible device context in memory, then selecting the passed bitmap into that device context. Once the bitmap is selected, call GetObj to retrieve its critical height and width measurements. Retrieve the destination dimensions by calling GetClientRect. Having this information at hand lets you calculate how many rows and columns you need to paint, and where each new copy should be placed.

Remember the golden rule when working with GDI objects: always return things to the state you found them in. Notice the cleanup code that returns everything to where it was. This avoids resource leaks (and worse!). ☒

Code Online

You can find all the code published in this issue of **VBPJ** ***on The Development Exchange (DevX) at http://www.windx.com. All the listings and associated files essential to the articles are available for free to Registered members of DevX, in one ZIP file. This ZIP file is also posted in the Magazine Library of the*** **VBPJ** ***Forum on CompuServe. DevX Premier Club members ($20 for six months) can get each article's listings in a separate file, as well as additional code and utilities for selected articles, plus archives of all code ever published in*** **VBPJ** ***and*** **Microsoft Interactive Developer** ***magazines.***

Name the Domain

Locator+ Codes

Listings ZIP file plus a class, CNetWksta, which encapsulates both NetWkstaGetInfo and NetWkstaGetUserInfo, and a CustomBlt module containing complete tiling code and declares (free Registered Level): VBPJEN97

✪ ***Listings for this article, plus a class, CNetWksta, which encapsulates both NetWkstaGetInfo and NetWkstaGetUserInfo; a class, CNetUser, which fully encapsulates the NetUserGetInfo API by providing all user information on any server; a CustomBlt module containing complete tiling code and declares; and a small form that demonstrates calling the CNetUser class (subscriber Premier Level): QAEN97P***