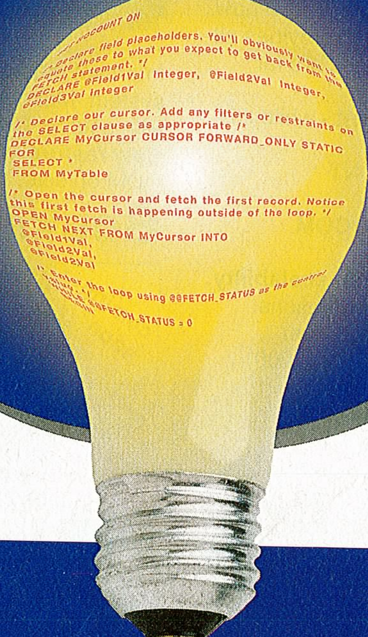
SUPPLEMENT TO

www.vbpj.com

A FAWCETTE TECHNICAL PUBLICATION, MARCH 2001 VOL. 11, NO. 3
INCLUDES WINDOWS PROGRAMMING & VISUAL PROGRAMMING

VISUAL BASIC®
PROGRAMMER'S JOURNAL

101 TECH TIPS



```
... NOCOUNT ON

/* Declare field placeholders. You'll obviously want to
equate these to what you expect to get back from the
FETCH statement. */
DECLARE @Field1Val Integer, @Field2Val Integer,
@Field3Val Integer

/* Declare our cursor. Add any filters or restraints on
the SELECT clause as appropriate /*
DECLARE MyCursor CURSOR FORWARD_ONLY STATIC
FOR
 SELECT *
 FROM MyTable

/* Open the cursor and fetch the first record. Notice
this first fetch is happening outside of the loop. */
OPEN MyCursor
FETCH NEXT FROM MyCursor INTO
  @Field1Val,
  @Field2Val,
  @Field2Val

  /* Enter the loop using @@FETCH_STATUS as the control
  value. */
  WHILE @@FETCH_STATUS = 0
  BEGIN
```

FOR VB DEVELOPERS

11th
EDITION

Welcome to the 11th Edition of the *VBPJ* Technical Tips Supplement!

These tips and tricks were submitted by professional developers using Visual Basic 3.0 through 6.0, Visual Basic for Applications (VBA), and Visual Basic Script (VBS). The editors at *Visual Basic Programmer's Journal* compiled the tips. Instead of typing the code published here, download the tips for free from the *VBPJ* Web site at www.vbpj.com.

If you'd like to submit a tip to *VBPJ*, please send it electronically to vbpjtips@fawcette.com. You can also send it to User Tips, Fawcette Technical Publications, 209 Hamilton Ave., Palo Alto, California, USA, 94301-2500; or fax it to 650-853-0230. Please include a clear explanation of what the technique does and why it's useful, and indicate if it's for VBA, VBS, VB3, VB4 16- or 32-bit, VB5, or VB6. Please limit code length to 20 lines. Don't forget to include your e-mail and mailing addresses, and let us know your payment preference: $25 per published tip or extending your *VBPJ* subscription by one year.

VB4 32, VB5, VB6
Level: Intermediate

Retrieve File Version Information

Win32 file images can contain a file version resource that stores product and version information about the file. The version number is actually four 16-bit values typically displayed using dot notation (such as 4.0.9.4566). You can use this information when determining whether one file is newer or older than another.

This code implements the GetVersionInfo procedure in a standard BAS module. Pass the name of a file to GetVersionInfo, and a dot-formatted string of the version number returns, if available, or "N/A" returns if the file does not contain a version resource:

```
Private Type VS_FIXEDFILEINFO
   dwSignature As Long
   dwStrucVersion As Long
   dwFileVersionMSl As Integer
   dwFileVersionMSh As Integer
   dwFileVersionLSl As Integer
   dwFileVersionLSh As Integer
   dwProductVersionMSl As Integer
   dwProductVersionMSh As Integer
   dwProductVersionLSl As Integer
   dwProductVersionLSh As Integer
   dwFileFlagsMask As Long
   dwFileFlags As Long
   dwFileOS As Long
   dwFileType As Long
   dwFileSubtype As Long
   dwFileDateMS As Long
   dwFileDateLS As Long
End Type
Private Declare Function GetFileVersionInfo _
   Lib "Version.dll" Alias _
   "GetFileVersionInfoA" (ByVal lptstrFilename _
   As String, ByVal dwHandle As Long, ByVal _
   dwLen As Long, lpData As Any) As Long
Private Declare Function _
   GetFileVersionInfoSize Lib "Version.dll" _
   Alias "GetFileVersionInfoSizeA" (ByVal _
   lptstrFilename As String, lpdwHandle As _
   Long) As Long
Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" (dest As Any, src As _
   Long, ByVal length As Long)
Private Declare Function VerQueryValue Lib _
   "Version.dll" Alias "VerQueryValueA" _
   (pBlock As Any, ByVal lpSubBlock As String, _
   lplpBuffer As Any, puLen As Long) As Long
Public Function GetVersionInfo(ByVal sFile As _
   String) As String
   Dim lDummy As Long
   Dim sBuffer() As Byte
   Dim lBufferLen As Long, lVerPointer As Long
   Dim lVerBufferLen As Long
   Dim udtVerBuffer As VS_FIXEDFILEINFO
   ' Default return value
   GetVersionInfo = "N/A"
   ' Attempt to retrieve version resource
   lBufferLen = GetFileVersionInfoSize(sFile, _
      lDummy)
   If lBufferLen > 0 Then
      ReDim sBuffer(lBufferLen)
      If GetFileVersionInfo(sFile, 0&, _
         lBufferLen, sBuffer(0)) <> 0 Then
         If VerQueryValue(sBuffer(0), _
            "\", lVerPointer, lVerBufferLen) _
            <> 0 Then
            CopyMemory udtVerBuffer, ByVal _
               lVerPointer, Len(udtVerBuffer)
            With udtVerBuffer
               GetVersionInfo = _
                  .dwFileVersionMSh & "." & _
                  .dwFileVersionMSl & "." & _
                  .dwFileVersionLSh & "." & _
                  .dwFileVersionLSl
            End With
         End If
      End If
   End If
End Function
```

—James D. Murray, Huntington Beach, California

VB5, VB6
Level: Beginning

Enforce Case With Enums

Enumerated constants are great, but they have a quirk that's a bit obnoxious: They don't retain their capitalization in the Integrated Development Environment (IDE), which a lot of folks use to provide visual feedback that they haven't misspelled the constant name. You can fool the IDE into retaining the capitalization by also declaring the Enums as public variables and surrounding the declarations with "#If False...#End If" compiler directives so they won't be compiled:

```
Public Enum MyEnum
   EnumOne=1
   EnumTwo
   EnumThree
End Enum
#If False Then
   Public EnumOne
   Public EnumTwo
   Public EnumThree
#End If
```

—Barry Garvin, Georgetown, Massachusetts

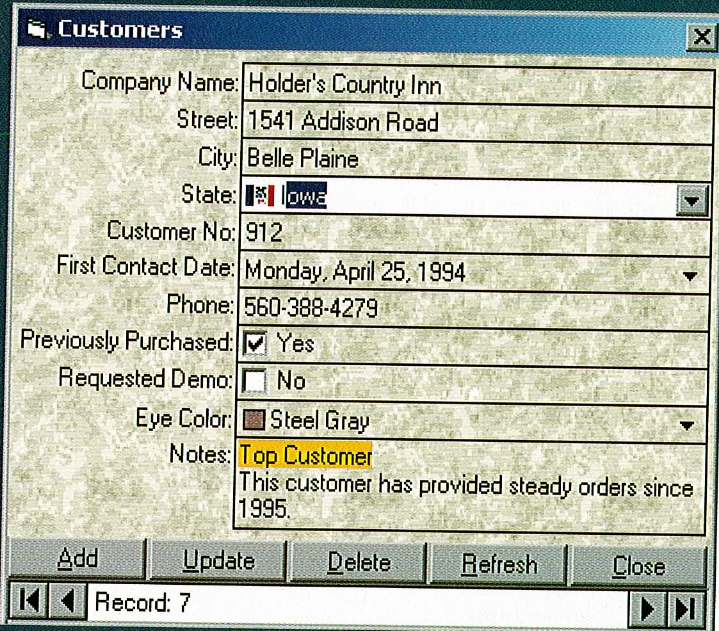
NEW! **EditPower** Woll2Woll Software

Woll2Woll Software is pleased to introduce the most powerful collection of ActiveX edit controls for Visual Basic. From our powerful mask language for user validation, to the most aesthetically pleasing looking controls, EditPower delivers quality and robust applications. For further info, visit http://www.woll2woll.com/editpower

The Premier Edit Control Collection For Developers

The Most Powerful & Expressive Edit Mask Language

EditPower includes the most powerful and expressive edit mask language for manipulating text & enforcing validation rules. Without any code, the user's input can be proper-cased (i.e. San Jose), auto-filled, or restricted to certain variable formats (i.e. 5 or 9 digit zip code format or an email address). Once you start using this mask language, you won't believe its capabilities, and how you survived so long without it!



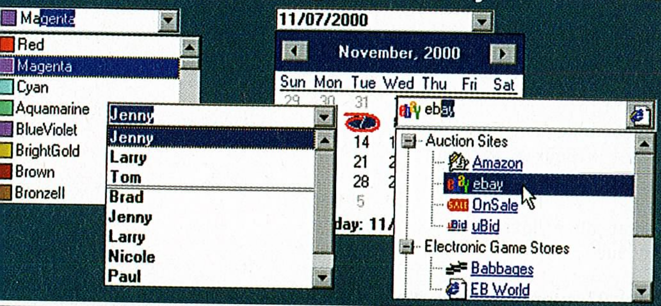
Company Name:	Holder's Country Inn
Street:	1541 Addison Road
City:	Belle Plaine
State:	Iowa
Customer No:	912
First Contact Date:	Monday, April 25, 1994
Phone:	560-388-4279
Previously Purchased:	✓ Yes
Requested Demo:	No
Eye Color:	Steel Gray
Notes:	Top Customer This customer has provided steady orders since 1995.

Built-In Transparency & Frame Effects

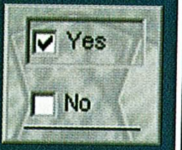
EditPower gives you the means to create professional forms that look just like the real hardcopy form they are based on. It's controls support transparency, focus and nonfocus frame styles.

Powerful & Sophisticated Collection of Edit Controls

EditPower includes sophisticated edit controls for any type of data including dates, times, and currency. Other powerful features include: quicken-style incremental searching, history and MRU (Most Recently Used) capabilities, maplist support (where a user sees descriptive text instead of a stored mapped code), validation masks, display formatting for currency & numeric fields, custom button glyphs, & more.



EditPower Professional version additionally includes: Transparent checkbox control w/framing, dropdown color controls, and an image combo or dropdown hierarchical (treeview) control.

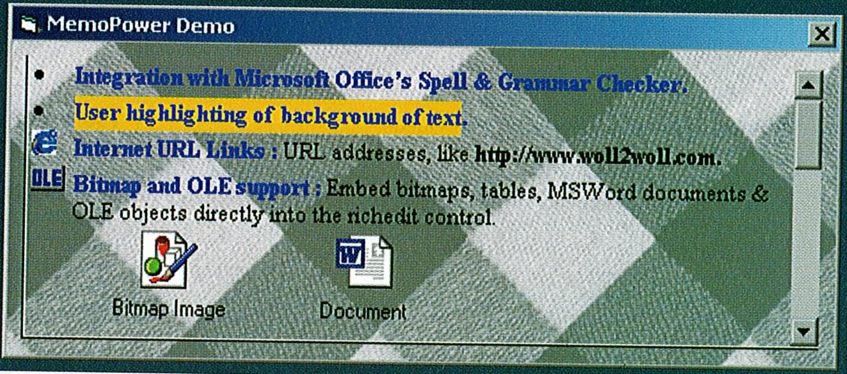


For a complete overview of all that EditPower has to offer you and for free demo and trial versions goto: http://www.woll2woll.com/editpower

MemoPower - The Ultimate Wordprocessor

FREE with EditPower Professional until May 2001 ($199 value).

Empower your users with formatted text capability. MemoPower is the most complete implementation of Microsoft's RichEdit control. It supports URL links, embedded OLE objects (such as images), MSWord Spell and Grammar Checker, integrated word-processor, and so much more.



- Integration with Microsoft Office's Spell & Grammar Checker.
- User highlighting of background of text.
- Internet URL Links : URL addresses, like http://www.woll2woll.com.
- Bitmap and OLE support : Embed bitmaps, tables, MSWord documents & OLE objects directly into the richedit control.

dress:	2217 Rhone Drive Livermore, CA 94550
one:	800-965-2965 (US) 925-371-1663 (INTL)
x:	925-371-1664
ail:	sales@woll2woll.com

For a limited time get MemoPower FREE ($199 Value) with your order of EditPower Professional.

EditPower Standard $199 EditPower Professional $299

Visit www.woll2woll.com/editpower for a tour, or to download a free trial version.

30 day money-back guarantee

Nothing But .NET

Wintellect
Raise Your Windows IQ™

1-877-WNTLLCT

www.wintellect.com

Wintellect announces its ***all star series*** *of .NET courses.*

Programming Microsoft .NET with C#
starring Jeffrey Richter

Programming Microsoft .NET with Visual Basic
starring Francesco Balena

Programming ASP.NET
starring Jeff Prosise

Programming ADO.NET
starring Dino Esposito

Debugging .NET Applications
starring John Robbins

Reserve your seat now at 1-877-WNTLLCT (877-968-5528) or visit www.wintellect.com for the latest info on the .NET tour schedule.

VB4 32, VB5, VB6
Level: Intermediate

Retrieve File Description

This routine takes a passed filename as an argument and generates a description for it. It returns the same string as Windows Explorer does when it has been set to Details view.

For example, if you pass the file c:\windows\win.com to the routine, it returns the string "MS-DOS Application." For files it can't describe, the routine returns a generic message of "{filetype} File." If the file passed doesn't exist, it returns "Unknown File," but you can change this easily. This code is especially useful for telling beginning users what type of file they're dealing with:

```
Private Declare Function SHGetFileInfo Lib _
   "shell32.dll" Alias "SHGetFileInfoA" _
   (ByVal pszPath As String, ByVal _
   dwFileAttributes As Long, psfi As _
   SHFILEINFO, ByVal cbFileInfo As Long, _
   ByVal uFlags As Long) As Long
Private Const SHGFI_TYPENAME = &H400
Private Const MAX_PATH = 260
Private Type SHFILEINFO
   hIcon As Long
   iIcon As Long
   dwAttributes As Long
   szDisplayName As String * MAX_PATH
   szTypeName As String * 80
End Type
Public Function GetFileType(lpStrFile As _
   String) As String
   Dim sfi As SHFILEINFO
   ' Make API Call to fill structure with
   ' information
   If SHGetFileInfo(lpStrFile, 0, sfi, _
      Len(sfi), SHGFI_TYPENAME) Then
      ' Return filetype string
      GetFileType = Left$(sfi.szTypeName, _
         InStr(sfi.szTypeName, vbNullChar) - 1)
   Else
      ' If failed then return "Unknown File"
      GetFileType = "Unknown File"
   End If
End Function
```

—Adam Lanzafame, Adelaide, South Australia, Australia

VB6
Level: Beginning

Employ Radio Buttons in a ListView

A simple piece of code can force the checkboxes in a ListView control to behave like radio buttons. Set the ListView's Checkboxes property to True and place this code in its ItemCheck event procedure:

```
Private Sub ListView1_ItemCheck(ByVal Item _
   As MSComctlLib.ListItem)
   Dim li As MSComctlLib.ListItem
   For Each li In ListView1.ListItems
      If li.Checked = True Then
         If li <> Item Then li.Checked = False
      End If
   Next li
End Sub
```

Each time the user checks one list item, any that were checked previously become unchecked.

—James D. Murray, Huntington Beach, California

VB4 32, VB5, VB6
Level: Intermediate

Authenticate Component Usage

Bundling functionality and program logic into an ActiveX DLL is an excellent form of encapsulation. But even when you expose functionality to your client application, you don't need to allow unrestricted access to all of your public functions. Use this simple mechanism to secure your proprietary functions from unauthorized access.

Create a private global variable, g_Authorized, of type Boolean to hold the authorization state for your DLL. When the DLL loads, g_Authorized is initialized to False. Each function (or sub) that you wish to protect should first check the value of g_Authorized before proceeding. If g_Authorized = False, then raise a runtime error advising the user that the function call is not authorized. If g_Authorized = True, then execute the function. For example, here is a snippet from one of the encryption routines. You use encryption to keep the data private, so you want to protect the encyption function itself from unauthorized access:

```
Public Function Encrypt(PlainText As String, _
   CipherType As axCipherType, Optional _
   ByVal Key As Long = 0) As String
   Dim iX As Long
   Dim iAscii As Integer
   Dim CipherText As String
   Dim StringLen As Long
   If Not g_Authenticated Then
      Err.Raise vbObjectError, "Encrypt", _
         "Application is not authorized " & _
         "to use this function"
   End If
```

The client application must call this public function to set the state of the DLL to Authorized (g_Authorized = True):

```
Public Sub Authenticate(Code As Variant)
   If Code = "asd93d,ssd" Then
      g_Authenticated = True
   End If
End Sub
```

Passing the code parameter as a Variant makes it more secure because a potential hacker would have no idea what sort of data the expected authentication code is.

This is the basic methodology for DLL protection. Actually, you could employ much more secure algorithms. You could derive the code from any number of potential values, such as the current system date/time, hard disk free space, or any other checkable value. Only your own applications would know the correct algorithm, so they would be the only applications on the client PC capable of authenticating the DLL for their use. Such a code would be more secure from a hack attack because it would actually change from minute to minute or machine to machine.

—Joseph Geretz, Monsey, New York

VB5, VB6
Level: Advanced

Unhook Subclassing When Windows is Ready

Don't unhook your Windows procedures from Form_Unload when subclassing forms. When you subclass forms, the hook is often set during Form_Load with code like this:

```
OriginalProc = SetWindowLong Me.hWnd, _
   GWL_WNDPROC, AddressOf MyWindowProc
```

A common mistake is forgetting to put the corresponding unhook call in your Form_Unload event:

```
SetWindowLong Me.hWnd, GWL_WNDPROC, _
   AddressOf OriginalProc
```

If you forget to reinstate the old procedure in your Form_Unload event, it's bye-bye VB. Instead, add this code within your subclassing procedure:

```
Select Case Msg
   Case WM_NCDESTROY
      If OriginalProc <> 0 Then
         Call SetWindowLong(hWnd, _
            GWL_WNDPROC, OriginalProc)
         OriginalProc=0
      End If
   Case ...
```

This code restores the original procedure automatically when the window is destroyed. To make it even safer, place all your subclassing code in a separate DLL and debug your subclassed forms without worrying about the Integrated Development Environment (IDE) crashing. You can always move the code back to your EXE when it's fully debugged.

—Simon Bryan, Newbury, Berkshire, England

VB4 32, VB5, VB6
Level: Beginning

Sort and Reverse-Sort a ListView

This routine performs the standard column sorting on a ListView control found in many commercial applications, such as Windows Explorer and Outlook. Using this routine, the ListView sorts itself automatically whenever the user clicks on a column. Clicking on the same column toggles the sort order between ascending and descending order. Call this routine from the ListView control's ColumnClick event procedure by passing both a reference to the ListView and the ColumnHeader reference passed to the original event:

```
Public Sub ListView_ColumnClick(ByRef _
   MyListView As ListView, ByVal ColumnHeader _
   As ColumnHeader)
   With MyListView
      .Sorted = False
      If .SortKey <> ColumnHeader.Index - _
         1 Then
         .SortKey = ColumnHeader.Index - 1
         .SortOrder = lvwAscending
      Else
         If .SortOrder = lvwAscending Then
            .SortOrder = lvwDescending
         Else
            .SortOrder = lvwAscending
         End If
      End If
      .Sorted = True
   End With
End Sub
```

—Jim Pragit, Glen Ellyn, Illinois

VB4, VB5, VB6
Level: Beginning

Test for Alpha Characters Only

Although VB has an IsNumeric function, it has no IsAlpha function. Use this routine whenever you want to determine whether a character or string of characters is alphabetic (A-Z or a-z). Add Case conditions for other characters you're willing to allow, such as hyphens, apostrophes, or whatever you consider legal:

```
Public Function IsAlpha(ByVal MyString As _
   String) As Boolean
   Dim i As Long
   ' Assume success
   IsAlpha = True
   ' Check to be sure
   For i = 1 To Len(MyString)
      Select Case Asc(Mid$(MyString, i, 1))
         Case vbKeyA To vbKeyZ
         Case (vbKeyA + 32) To (vbKeyZ + 32)
         Case vbKeySpace
         ' Add more tests to suit
         Case Else
            IsAlpha = False
            Exit For
      End Select
   Next i
End Function
```

—Jim Pragit, Glen Ellyn, Illinois

VB3 and up
Level: Beginning

Test for Alpha Characters Only, Part II

I have a much simpler form for the IsAlphaNum function:

```
Public Function IsAlphaNum(ByVal sString _
   As String) As Boolean
   If Not sString Like "*[!0-9A-Za-z]*" _
      Then IsAlphaNum = True
End Function
```

You can modify this function for other conditions. Simply put the acceptable characters—such as a space, hyphen, or dot—into the square brackets.

—Rick Rothstein, Trenton, New Jersey

VB3 and up
Level: Beginning

Test for Alpha Characters Only, Part III

Traditional testing for alphabetic characters—for example, to restrict characters that can be entered in a textbox—uses the ASCII value of the keypress:

```
If KeyAscii >=65 And KeyAscii < 113 Then
```

However, this test doesn't allow for international code pages, which might include characters with an ASCII code higher than 113. A more logical definition of an alphabetical character is one that has a distinct upper and lowercase. To test whether something is alphabetic, use this code:

```
' International IsAlpha character test.
' Returns true if the input letter is
' alphabetical in any code page or language.
Public Function IsAlphaIntl(sChar As String) _
   As Boolean
   IsAlphaIntl = Not (UCase$(sChar) = LCase$(sChar))
End Function
```

—Duncan Jones, Caistor, Lincolnshire, England

VB5, VB6
Level: Beginning

Open Your VB Projects With a Clean Slate

If you're like me, you hate all the clutter of open form, class, and module windows when you open your VB projects in the Integrated Development Environment (IDE). Here's a simple workaround to let you start your project with a clean workspace.

Edit the accompanying VB Workspace file for your project. It has the same name as your project file, but with a VBW file extension. Delete all the lines in this file and save it. Now make this file read-only by right-clicking on the file, choosing Properties, then selecting the read-only checkbox.

Whenever you save your project from then on, VB won't update this file because it is read-only, and it won't complain. Each time you open your project, your workspace will start fresh with no clutter. If for any reason you want to revert to the old way, simply change the read-only flag back.

—**Richard Edwards, Belleville, Ontario, Canada**

VB5, VB6
Level: Advanced

☆☆☆☆☆ **Five Star Tip**

Load a Bitmap Resource From a DLL

You can employ any DLL's bitmap resources in VB using this LoadPicture function. You need to set a reference to OLE Automation:

```
Private Type GUID
   Data1 As Long
   Data2 As Integer
   Data3 As Integer
   Data4(7) As Byte
End Type
Private Type PicBmp
   Size As Long
   Type As Long
   hBmp As Long
   hPal As Long
   Reserved As Long
End Type
Private Declare Function _
   OleCreatePictureIndirect Lib _
   "olepro32.dll" (PicDesc As PicBmp, RefIID _
   As GUID, ByVal fPictureOwnsHandle As Long, _
   IPic As IPicture) As Long
Private Declare Function LoadBitmap Lib _
   "user32" Alias "LoadBitmapA" (ByVal _
   hInstance As Long, ByVal lpBitmapID As _
   Long) As Long
Private Declare Function DeleteObject Lib _
   "gdi32" (ByVal hObject As Long) As Long
Private Declare Function LoadLibrary Lib _
   "kernel32" Alias "LoadLibraryA" (ByVal _
   lpLibFileName As String) As Long
Private Declare Function FreeLibrary Lib _
   "kernel32" (ByVal hLibModule As Long) _
   As Long
Public Function LoadPicture(sResourceFileName _
   As String, lResourceId As Long) As Picture
   Dim hInst As Long
   Dim hBmp As Long
   Dim Pic As PicBmp

   Dim IPic As IPicture
   Dim IID_IDispatch As GUID
   Dim lRC As Long
   hInst = LoadLibrary(sResourceFileName)
   If hInst <> 0 Then
      hBmp = LoadBitmap(hInst, lResourceId)
      If hBmp <> 0 Then
         IID_IDispatch.Data1 = &H20400
         IID_IDispatch.Data4(0) = &HC0
         IID_IDispatch.Data4(7) = &H46
         Pic.Size = Len(Pic)
         Pic.Type = vbPicTypeBitmap
         Pic.hBmp = hBmp
         Pic.hPal = 0
         lRC = OleCreatePictureIndirect(Pic, _
            IID_IDispatch, 1, IPic)
         If lRC = 0 Then
            Set LoadPicture = IPic
            Set IPic = Nothing
         Else
            Call DeleteObject(hBmp)
         End If
      End If
      Call FreeLibrary(hInst)
      hInst = 0
   End If
End Function
Private Sub Form_Load()
   ' Try ID 130 in Win98, or 131 in NT
   ' to see the Windows logo...
   Set Me.Picture = _
      LoadPicture("shell32.dll", 130)
End Sub
```

—**Michael Hill, Northridge, California**

VB4 32, VB5, VB6
Level: Intermediate

Add Incremental Search to a Combo Box

As the user types into a drop-down combo box, he or she passes keystrokes to the ComboIncrementalSearch routine, which then searches the combo box's data for the best match:

```
Private Declare Function SendMessage Lib _
   "user32" Alias "SendMessageA" (ByVal hWnd _
   As Long, ByVal wMsg As Long, ByVal wParam _
   As Long, lParam As Any) As Long
Private Const CB_FINDSTRING = &H14C
Private Sub Combo1_KeyPress(KeyAscii _
   As Integer)
   Call ComboIncrementalSearch(Combo1, _
      KeyAscii)
End Sub
Public Sub ComboIncrementalSearch(cbo As _
   ComboBox, KeyAscii As Integer)
   Static dTimerLast As Double
   Static sSearch As String
   Static hWndLast As Long
   Dim nRet As Long
   Const MAX_KEYPRESS_TIME = 0.5
   ' Weed out characters that are not scanned
   If (KeyAscii < 32 Or KeyAscii > 127) _
      Then Exit Sub
   If (Timer - dTimerLast) < _
      MAX_KEYPRESS_TIME And hWndLast = _
      cbo.hWnd Then
      sSearch = sSearch & Chr$(KeyAscii)
   Else
      sSearch = Chr$(KeyAscii)
      hWndLast = cbo.hWnd
   End If
   ' Search the combo box
   nRet = SendMessage(cbo.hWnd, _
      CB_FINDSTRING, -1, ByVal sSearch)
   If nRet >= 0 Then
      cbo.ListIndex = nRet
   End If
   KeyAscii = 0
   dTimerLast = Timer
End Sub
```

—**Michael Hill, Northridge, California**

For even more tricks and tips go to
www.vbpj.com

VB4 32, VB5, VB6
Level: Beginning

Operate on Array of Selected ListItems

The fast way to get multiple selected items from a ListBox control is to send it a LB_GETSELITEMS window message. Here's a simple example that moves items from one ListBox to another ListBox. To test this example, place two ListBox controls (lstFrom and lstTo) and a Command button control (cmdMove) on a form, then copy this code into the form's code editing page:

```
Option Explicit
Private Declare Function SendMessage Lib _
   "user32" Alias "SendMessageA" (ByVal hWnd _
   As Long, ByVal wMsg As Long, ByVal wParam _
   As Long, lParam As Any) As Long
Private Const LB_GETSELCOUNT = &H190
Private Const LB_GETSELITEMS = &H191
Private Sub Form_Load()
   ' Add some items into source list
   lstFrom.AddItem "Matthew So"
   lstFrom.AddItem "Join"
   lstFrom.AddItem "Hello"
   lstFrom.AddItem "Morning"
   lstFrom.AddItem "Apple"
End Sub
Private Sub cmdMove_Click()
   Dim nRet As Long
   Dim nSel() As Long
   Dim i As Long
   nRet = SendMessage(lstFrom.hWnd, _
      LB_GETSELCOUNT, 0, ByVal 0&)
   Me.Caption = CStr(nRet)
   If nRet > 0 Then
      ' Allocate enough memory for the array
      ReDim nSel(0 To nRet - 1)
      ' Get an array of ListIndexes for the
      ' selected items
      nRet = SendMessage(lstFrom.hWnd, _
         LB_GETSELITEMS, lRet, nSel(0))
      ' Start from the end of list to avoid
      ' index change of the source list
      For i = UBound(nSel) To LBound(nSel) _
         Step -1
         ' Copy item from source list to
         ' destination list
         lstTo.AddItem lstFrom.List(nSel(i))
         ' Remove selected item from source
         ' list
         lstFrom.RemoveItem nSel(i)
      Next i
   End If
End Sub
```

The tricks here are to redimension the nSel array using the count of selected items returned by LB_GETSELCOUNT, and to move the selected items starting from the end of the origin ListBox. If you try to move items from the beginning of the ListBox, the ListItems are shifted downward and the saved array of item ListIndexes is no longer valid.

—Matthew So, Hong Kong

VB6
Level: Beginning

Return Empty Arrays Too

With VB6 came the ability to return arrays from functions. Returning an uninitialized array is a problem because there is no easy way—other than error-trapping—to find whether an array has been dimensioned. Also, ReDim myArr(-1) does not work. You can use the Split function to return an empty array—one with no elements and no data—and an LBound of 0 and an UBound of -1. This practice simplifies code for looping through the returned array:

```
Private Function Foo(args...) As String()
   Dim myArr() As String
   ' Initialize array dimensions as 0 to -1
   myArr = Split("")
   If Condition Then
      ReDim myArr(n)
      ' further processing...
   End If
   Foo = myArr
End Function
```

Here, no additional checking is required to use Foo. But omitting the call to Split can lead to a "Subscript out of range" error in a routine that attempts to use Foo's return; when you use Split to establish an empty array, this loop is simply skipped over:

```
Dim i As Integer
Dim retArr() As String
retArr = Foo
For i = LBound(retArr) To _
   UBound(retArr)
   ' Does not execute as LBound > UBound.
Next i
```

—Anand Likhite, Orlando, Florida

VB3 and up
Level: Beginning

Handle Errors Within Forms

When you load or show a form, errors don't bubble up. That is, even if the calling procedure has an error handler, and an error occurs in Form_Load, Form_Initialize, or any other form event, processing doesn't transfer to the calling error handler. In this code, Sub Main has an error handler. But when an error occurs in the Form_Load, the error handler isn't called:

```
' Code in a bas module
Sub Main()
   On Error Resume Next
   Load Form1
   ' further processing code ...
End Sub
' Code in Form1
Private Sub Form_Load()
   Dim a As Integer
   a = 1 / 0   ' error is fatal!
End Sub
```

If you check the call stack when the error occurs, you see an entry '<Non-Basic Code>' before the Form_Load. Even though Sub Main is loading Form1, Sub Main is not the direct caller of Form_Load, and that results in this behavior.

—Ravindra Okade, Phoenix, Arizona

VB3 and up
Level: Beginning

Assign Null Fields to Controls Without Error

Here's a great way to assign Null values from a database to your VB controls. Concatenating an empty string to Null produces an empty string. Using the & operator, you can convert all nonstring values to strings using little code:

```
txtName.Text = rs("Name") & ""
```

—Deborah Hammel, Sparks, Maryland

VB4, VB5, VB6
Level: Beginning

Use Objects Directly Within Collections

If you use collections in your apps, you're probably familiar with the For...Each loop to iterate through the collection. But when you want to access only a single element of the collection to perform some temporary calculations or modifications, you might be tempted to do something like this:

```
Set Obj = coll.Item(KeyName)
Obj.Property = something
Call Obj.Method(parameter)
   ...<etc.>
```

Instead, you can simplify your code by using the Collection object in place of Obj. Then you never have to dimension these temporary holders for collection items:

```
coll(KeyName).Property = something
Call coll(KeyName).Method(parameter)
...<etc.>
```

Or you can use this:

```
With coll(KeyName)
   .Property = something
   Call .Method(parameter)
   ...<etc.>
End With
```

—Ian Fenton, Burlington, Ontario, Canada

VB3 and up
Level: Beginning

Use Safer International Conversions

I have problems developing applications for non-English–speaking users because of the noninternational behavior of the Val function. The functions CSng and CDbl provide internationally aware conversions, but they don't work correctly if the argument is empty or contains alpha characters, as can often be the case when converting from nonvalidated TextBox controls. I avoid the errors with this simple function:

```
Private Function CTxtToSng(sInput As String) As Single
   ' CTxtToSng at beginning is worth 0 (zero)
   ' If CSng produces a conversion error, the
   ' value stays 0 (zero)
   On Error Resume Next
   CTxtToSng = CSng(sInput)
End Function
```

You can edit this sample to produce a Double return, if that's more desirable.

—Giovanni Buommino, Dreieich, Germany

VB6
Level: Intermediate

Store Multiple Values in Tag

It would often be convenient to store multiple values in the Tag property. Here are two simple functions that help you do that. The first function stores the value in the control's Tag, tagging the new value with a key value of your choice:

```
Public Function SetTag(ctl As Control, ByVal _
   Key As String, NewValue As String) As String
   Dim myArry() As String
   Dim i As Integer, k As Integer
   Dim yTag As String, yValue As String
   k = -1
   Key = UCase$(Key)
   If ctl.Tag = "" Then
      ctl.Tag = ctl.Tag & "|" & Key & "=" & _
         NewValue
   Else
      myArry = Split(ctl.Tag, "|")
      For i = LBound(myArry) To UBound(myArry)
         If UCase$(Left$(myArry(i), _
            Len(Key))) = Key Then k = i
      Next
      If k > -1 Then
         myArry(k) = Key & "=" & NewValue
         ctl.Tag = Join(myArry, "|")
      Else
         ctl.Tag = ctl.Tag & "|" & Key & "=" _
            & NewValue
      End If
   End If
   SetTag = ctl.Tag
End Function
```

You can store the Tag value easily using a statement like this:

```
Call SetTag(myCtrl, "ID", LoginID)
```

A complementary function allows you to retrieve the stored value:

```
Public Function GetTag(ctl As Control, Key As _
   String) As String
   Dim myArry() As String,
   Dim i As Integer, k As Integer
   Dim yPoze As String, yValue As String
   k = -1
   If ctl.Tag = "" Then
      yValue = ""
   Else
      myArry = Split(ctl.Tag, "|")
      For i = LBound(myArry) To UBound(myArry)
         If UCase$(Left$(myArry(i), _
            Len(Key))) = UCase$(Key) Then k = i
      Next i
      If k > -1 Then
         yPoze = InStr(myArry(k), "=")
         yValue = Mid$(myArry(k), yPoze + 1)
      Else
         yValue = ""
      End If
   End If
   GetTag = yValue
End Function
```

You can retrieve the code using a statement like this:

```
LoginID = GetTag(myCtrl, "ID")
```

As written, these functions are case-insensitive with the key names. If you want case-sensitive key values, remove all UCase calls.

—Enrico Di Cesare, Arese, Italy

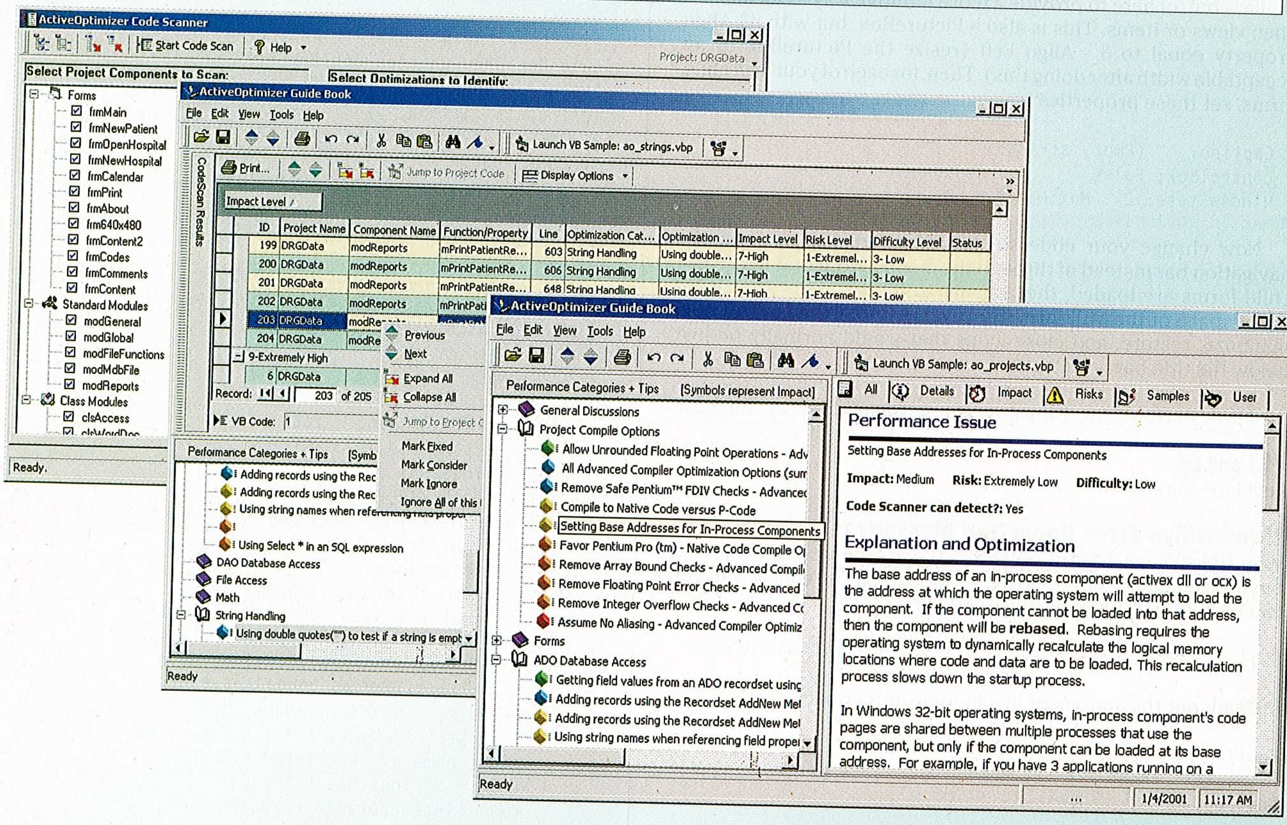
ActiveOptimizer™

Is your application too fast?

Your users don't think so - guaranteed! Now, using ActiveOptimizer™ for Visual Basic, you can satisfy your users with substantial performance gains. ActiveOptimizer™ provides you with the knowledge and tools to make your applications scream! ActiveOptimizer™ is a unique tool that provides you with over 15 categories and over 100 performance improvement techniques. And we mean real performance improvement techniques - tips we have researched, tested, and profiled to report the performance improvement levels you can expect for each technique. Plus, ActiveOptimizer™ can even scan your proejct code and identify techniques for you to apply!

ActiveProfiler | Trace Console | CodeScanner | GuideBook

Integrated directly into the Visual Basic IDE!



ID	Project Name	Component Name	Function/Property	Line	Optimization Cat...	Optimization ...	Impact Level	Risk Level	Difficulty Level	Status
199	DRGData	modReports	mPrintPatientRe...	603	String Handling	Using double...	7-High	1-Extremel...	3- Low	
200	DRGData	modReports	mPrintPatientRe...	606	String Handling	Using double...	7-High	1-Extremel...	3- Low	
201	DRGData	modReports	mPrintPatientRe...	648	String Handling	Using double...	7-High	1-Extremel...	3- Low	
202	DRGData	modReports	mPrintPati							
203	DRGData	modRe								
204	DRGData	modRe								
9-Extremely High										
6	DRGData									

Performance Categories + Tips [Symb
- Adding records using the Rec
- Adding records using the Rec
- Using string names when referencing field proper
- Using Select * in an SQL expression
- DAO Database Access
- File Access
- Math
- String Handling
  - Using double quotes("") to test if a string is empt

ActiveOptimizer Guide Book

File Edit View Tools Help

Launch VB Sample: ao_projects.vbp

Performance Categories + Tips [Symbols represent Impact]
- General Discussions
- Project Compile Options
  - Allow Unrounded Floating Point Operations - Adv
  - All Advanced Compiler Optimization Options (sum
  - Remove Safe Pentium™ FDIV Checks - Advanced
  - Compile to Native Code versus P-Code
  - Setting Base Addresses for In-Process Components
  - Favor Pentium Pro (tm) - Native Code Compile O
  - Remove Array Bound Checks - Advanced Compil
  - Remove Floating Point Error Checks - Advanced
  - Remove Integer Overflow Checks - Advanced Co
  - Assume No Aliasing - Advanced Compiler Optimiz
- Forms
- ADO Database Access
  - Getting field values from an ADO recordset using
  - Adding records using the Recordset AddNew Me
  - Adding records using the Recordset AddNew Me
  - Using string names when referencing field proper

All | Details | Impact | Risks | Samples | User

Performance Issue

Setting Base Addresses for In-Process Components

Impact: Medium **Risk:** Extremely Low **Difficulty:** Low

Code Scanner can detect?: Yes

Explanation and Optimization

The base address of an in-process component (activex dll or ocx) is the address at which the operating system will attempt to load the component. If the component cannot be loaded into that address, then the component will be **rebased**. Rebasing requires the operating system to dynamically recalculate the logical memory locations where code and data are to be loaded. This recalculation process slows down the startup process.

In Windows 32-bit operating systems, in-process component's code pages are shared between multiple processes that use the component, but only if the component can be loaded at its base address. For example, if you have 3 applications running on a

Ready ... 1/4/2001 11:17 AM

Make Your Applications Scream!

✓ **Optimizing GuideBook with CodeScanner**

Absolutely the most comprehensive database of performance improvement techniques ever compiled - complete with detailed explanations, impact timings, risk assessments, and samples. Don't know where to start? No problem! Let the Active CodeScanner show you where improvements where improvements can be made. It is just that easy! Stop wasting your valuable time trying to guess at performance improvement techniques. Allow ActiveOptimizer™ to guide you to better performing applications right now. **Available now - order**

✓ **Complete Code Profiler with Remote Tracing ***

Quick and easy to use, yet incredibly powerful. Complete reporting including program execution trace, bottleneck analysis, multiple run comparisons, ... and more! You can even collect trace information on your user's machine, <u>royalty free</u>! (Available March 2001)

PlatformDev

Platform Development Technologies, Inc.

Tel. 678-413-1037 email:info@platformdev.com

30 day trial available on our website

Profiler Only	$149
GuideBook Only	$199

* Profiler available March 2001
GuideBook/CodeScanner is available now!

www.platformdev.com

VB5, VB6
Level: Intermediate

Detect Change of Windows Locale

While designing multilingual applications, I had to make them respond when a user changes the Windows locale setting—for example, by loading a different language's captions or text. Although it's possible to intercept a Windows message generated and broadcast when the Windows locale changes, this requires subclassing, which is not always desirable.

Another solution is to create an ActiveX control responsible for detecting the change and raising an event. The UserControl object raises a private AmbientChanged event when any of the AmbientProperties change, and it passes the changed property's name to the event. The Ambient object provides a LocaleID property, which you can pass to the client with a public custom LocaleChanged event. Your ActiveX control might have no visual interface, pretty much like a Timer control, and you can place it on any form required to react to a change of Windows locale. You can load the new language's elements subsequently from a database or a resource file:

```
Public Event LocaleChanged(ByVal LocaleID As Long)
Private Sub _
   UserControl_AmbientChanged(PropertyName As String)
   If PropertyName = "LocaleID" Then
      RaiseEvent LocaleChanged(Ambient.LocaleID)
End Sub
```

—**Brian Hunter, Brooklyn, New York**

VB6
Level: Advanced

Clean Up the MonthView

If you've tried using the Microsoft MonthView control, part of the Windows Common Controls 2 collection, you probably discarded it after discovering the quirky spinner that pops up when you click on the year. This is supposed to make it easy to change years. Unfortunately, when you click on one of the spinner buttons, an ugly border artifact appears to the right of the spinner. If you don't mind eliminating the spinners, you can still use the control. You must subclass the MonthView control temporarily and destroy the spinner button when it is created:

```
' Form code
Option Explicit
Private Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal _
   hWnd As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long
Private Const GWL_WNDPROC = (-4)
Private Sub MonthView1_MouseDown(Button As _
   Integer, Shift As Integer, X As Single, Y _
   As Single)
   Dim d As Date
   If MonthView1.HitTest(X, Y, d) = _
      mvwTitleYear Then
      lmvWndProc = _
         SetWindowLong(MonthView1.hWnd, _
         GWL_WNDPROC, AddressOf MVWndProc)
   End If
End Sub
' Module code
Option Explicit
Private Declare Function CallWindowProc Lib _
   "user32" Alias "CallWindowProcA" (ByVal _
   lpPrevWndFunc As Long, ByVal hWnd As Long, _
   ByVal msg As Long, ByVal wParam As Long, _
   ByVal lParam As Long) As Long
Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" (hpvDest As Any, _
   hpvSource As Any, ByVal cbCopy As Long)
Private Declare Function DestroyWindow Lib _
   "user32" (ByVal hWnd As Long) As Long
Private Declare Function SetWindowLong Lib _
   "user32" Alias "SetWindowLongA" (ByVal _
   hWnd As Long, ByVal nIndex As Long, ByVal _
   dwNewLong As Long) As Long
Private Const GWL_WNDPROC = (-4)
Private Const WM_CREATE = &H1
Private Const WM_PARENTNOTIFY = &H210
Public lmvWndProc As Long
Public Function MVWndProc(ByVal hWnd As Long, _
   ByVal msg As Long, ByVal wParam As Long, _
   ByVal lParam As Long) As Long
   Select Case msg
      Case WM_PARENTNOTIFY
         Select Case LoWord(wParam)
            Case WM_CREATE
               DestroyWindow lParam
               SetWindowLong hWnd, _
                  GWL_WNDPROC, lmvWndProc
         End Select
   End Select
   MVWndProc = CallWindowProc(lmvWndProc, _
      hWnd, msg, wParam, lParam)
End Function
Public Function LoWord(lnum As Long) As Integer
   CopyMemory LoWord, lnum, 2
End Function
Public Function HiWord(lnum As Long) As Integer
   CopyMemory HiWord, ByVal VarPtr(lnum) + 2, 2
End Function
```

—**Matt Hart, Tulsa, Oklahoma**

VB5, VB6
Level: Advanced

☆☆☆☆☆ Five Star Tip

Capture Reference to UserControl

Many programmers are familiar with declaring an object variable in class modules and other places to capture events from a form and handle them in a generic way:

```
Private WithEvents m_Form As Form
```

It might be useful to do this for user controls as well, but you need a reference to the UserControl object. Getting this reference proves harder than it should be. This code sets up the m_UserControl variable:

```
' Declarations
Private WithEvents m_UserControl As UserControl
Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" (pDest As Any, _
   pSource As Any, ByVal ByteLen As Long)
Private Sub UserControl_Initialize()
   ' Code to set up the m_UserControl variable
   Dim UC As UserControl
   CopyMemory UC, UserControl, 4
   Set m_UserControl = UC
   CopyMemory UC, 0&, 4
End Sub
```

Once this code has been executed, the m_UserControl events fire as expected. Using this technique and sharing the created reference, you can sink the UserControl events in a class module, allowing development of generic event handlers for your controls.

—**Jeremy Adams, Tiverton, Devon, England**

VB4 32, VB5, VB6
Level: Intermediate

Limit User Typing in Combo Box

The standard textbox has a MaxChars property that lets you limit the number of characters a user can type into it. The drop-down combo does not, but you can emulate this property setting with a simple API call:

```
Private Declare Function SendMessage Lib _
   "user32" Alias "SendMessageA" (ByVal hWnd _
   As Long, ByVal msg As Long, ByVal wParam _
   As Long, ByVal lParam As Long) As Long
Private Const CB_LIMITTEXT = &H141
Private Sub Form_Load()
   Const Max_Char = 24
   Call SendMessage(Combo1.hWnd, _
      CB_LIMITTEXT, Max_Char, 0&)
End Sub
```

Editor's Note: This tip works in 16-bit versions of VB as well, but you'll need to substitute the correct 16-bit declarations for SendMessage and CB_LIMITTEXT.

—Jim Deutch, Syracuse, New York

VB3 and up
Level: Intermediate

Ask for Directions

In graphical applications, you often need to know the angle between two lines. You can move their intersection point to the origin easily, so all you need to do is choose a point on each line and find the angle between a line from the origin to that point and the x-axis. The angle between the lines is the difference. The arctangent of y/x is the mathematical function you need to find these angles, but VB's Atn() function only returns angles between -PI/2 and PI/2 (-90 to +90 degrees). You lose half the circle! And it fails completely if x = 0 (divide by zero error!).

Many languages provide an Atn2() function that extends Atn() to the entire circle, taking the x and y arguments separately to avoid the divide error. Here's a straightforward VB function that treats all possible cases separately:

```
Function Atn2(ByVal x As Double, ByVal y _
   As Double) As Double
   Dim theta As Double
   Const pi As Double = 3.14159265359
   If x <> 0 Then
      theta = Atn(y / x)
      If x < 0 Then
         theta = theta + pi
      End If
   Else
      If y < 0 Then
         theta = 3 * pi / 2   ' 90 deg
      Else
      theta = pi / 2   ' 270 deg
      End If
   End If
   Atn2 = theta
End Function
```

—Jim Deutch, Syracuse, New York

VB3 and up
Level: Beginning

Center Your Logo on MDI Forms

You can display a logo in the middle of your MDI form. The logo stays in the middle even when the MDI form is resized. After creating your own MDI form, add a standard form to your project and put an Image control named imgLogo on it. Instead of the Image control, you can use a Label control or whatever you want. The standard form (frmLogo) should have these properties set:

```
PROPERTIES of frmLogo:
   MDIChild = True
   BorderStyle = 0 - None
```

Then put this code in your MDI form Resize event:

```
Private Sub MDIForm_Resize()
   ' Now center the frmLogo form in your MDI form
   frmLogo.Left = (Me.ScaleWidth - frmLogo.Width) / 2
   frmLogo.Top = (Me.ScaleHeight - frmLogo.Height) / 2
End Sub
```

Put this code in the logo form's Activate and Resize events:

```
Private Sub Form_Activate()
   ' Force logo to background
   Me.ZOrder vbSendToBack
End Sub
Private Sub Form_Resize()
   ' Move logo to upper-left
   imgLogo.Move 0, 0
   ' The next fragment makes frmLogo's
   ' Width and Height equal to imgLogo's
   ' Width and Height
   Me.Width = imgLogo.Width
   Me.Height = imgLogo.Height
End Sub
```

Load and show the logo form during the MDI form's Load event:

```
Private Sub MDIForm_Load()
   frmLogo.Show
End Sub
```

—Pavel Tsekov, Varna, Bulgaria

VB5, VB6
Level: Beginning

View Right Side of Truncated String

You see trailing ellipses (...) when VB truncates either the expression or data portion of a data tip (the mouse-hover watch value you get while debugging). This is great if you want to see the left side of a long string value, but not quite as compelling if you care about the right side. Hold the control key down and rehover over the expression to force VB to truncate on the left instead of the right. VB truncates all instant watch strings at 251 characters, so you won't see the end of very long strings.

—Matt Curland, Redmond, Washington

VB3 and up
Level: Beginning

Pass ByVal to ByRef Parameters
By default, VB passes all arguments to a procedure by reference, which means the procedure can change the values of the variables you pass. However, there's a simple way to override a ByRef argument without changing the procedure's ByRef behavior. Here's a typical procedure with ByRef arguments:

```
Private Sub ModifyByRef(sVar1 As String, _
   sVar2 As String)
   sVar1 = sVar1 & " has been modified."
   sVar2 = sVar2 & " has been modified."
End Sub
```

Use this syntax before calling the procedure:

```
   sVar1 = "Var1"
   sVar2 = "Var2"
   Call ModifyByRef(Var1,Var2)
```

Then you'll get this result:

```
   sVar1 contains "Var1 has been modified."
   sVar2 contains "Var2 has been modified."
```

To override the ByRef for Var1, use VB's expression evaluator and place parentheses around the variable before calling the Modify-ByRef() procedure:

```
   ' Note the parentheses around Var1.
   Call ModifyByRef((Var1),Var2)
```

Then you'll get this result:

```
   sVar1 contains the original value "Var1"
   sVar2 contains the changed value _
      "Var2 has been modified."
```

Using the expression evaluator's parentheses gives you control over exactly how parameters are passed into a called procedure, without having to resort to assigning temporary variables to override the behavior of ByRef. By using parentheses, you have a choice.

—**David Tate Helene, Rockville, Maryland**

VB5, VB6
Level: Beginning

Use Control/Space for VB IntelliSense
You can press Ctrl-Spacebar to make IntelliSense prompt you for variables, methods, properties, or events at any point in a code window.

For example, if you have a variable named myvariable, typing myv, then pressing Ctrl-Spacebar autocompletes the variable name. If more than one item matches what you type, IntelliSense offers a list of matches.

—**Doug Waterman, Appleton, Wisconsin**

VB4 32, VB5, VB6
Level: Intermediate

Copy an Array Faster
Here's an optimized method of copying one array to another. Usually when copying an array to another array, the developer iterates through each item of the source array, assigning the item to the associated item of the destination array:

```
Private Declare Function timeGetTime Lib _
   "winmm.dll" () As Long
Private Sub Copy()
   Dim i
   Dim startTime As Long
   Dim endTime As Long
   Dim intSrc(1 To 6000000) As Integer
   Dim intDest(1 To 6000000) As Integer
   startTime = timeGetTime
      For i = LBound(intSrc) To UBound(intSrc)
         intDest(i) = intSrc(i)
      Next i
   endTime = timeGetTime
   Debug.Print "Copy took: " & endTime - _
      startTime & " ms."
End Sub
```

Instead, use the Win32 API function CopyMemory to copy the array from source to destination:

```
Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" (Destination As Any, _
   Source As Any, ByVal Length As Long)
Private Sub FastCopy()
   Dim startTime As Long
   Dim endTime As Long
   Dim bytes As Long
   Dim intSrc(1 To 6000000) As Integer
   Dim intDest(1 To 6000000) As Integer
   bytes = (UBound(intSrc) - LBound(intSrc) _
      + 1) * Len(intSrc(LBound(intSrc)))
   startTime = timeGetTime
      CopyMemory intDest(LBound(intDest)), _
         intSrc(LBound(intSrc)), bytes
   endTime = timeGetTime
   Debug.Print "FastCopy took: " _
      & endTime - startTime & " ms."
End Sub
```

When compiled to native code with all optimizations, this second method averages up to 15 times faster, and a stunning 30 to 35 times faster when running as compiled p-code or in the Integrated Development Environment (IDE). But be warned: You can also GPF at blinding speeds if you miscalculate the number of bytes to copy, use bad source or destination addresses, or if your destination array isn't sized sufficiently.

—**Andrew Holliday, Phoenix, Arizona**

VB6
Level: Advanced

Don't Use Default Properties When Working With Hierarchical Recordsets

It's common VB knowledge that you can omit the default property of a control or an object. For example, statements such as Text1.Text = "blah" and Text1 = "blah" are equivalent, with Text being a default property of text control. Objects behave similarly.

When you have two tables, Orders and Items, in a parent-child relationship by OrderNumber, you can build a hierarchical recordset that stores Items' data for a particular order within its parent's record in a column called chapter1 as specified in the rsOrders.Open... statement.

To retrieve the Items for particular orders, you need to loop through the parent recordset rsOrders, and assign the data stored in column chapter1 to a child recordset rsItems. Use a statement such as this:

```
Set rsItems = rsOrders("chapter1").Value
```

If you rely on Value being the default property of Column object and you omit this property in code, you get a Type Mismatch error if your rsItems variable is declared as ADODB.Recordset. Some examples I found in the Microsoft help files would declare this variable as Variant. In this case, you can pass the assignment Set rsItems = rsOrders("chapter1") with no error generated, but later on, if you try to loop through the child set of data, you get an Object Required error referring to an .EOF property that does not exist on a Variant.

None of these problems happen if you declare rsItems properly as ADODB.Recordset and use the Value property of the Column object explicitly. Strangely enough, when you omit .Columns (which is a default property of Recordset object), nothing bad happens. Here's a code example that works:

```
Private Sub cmdGetRecords_Click()
Dim connstring As String
Dim cnn As ADODB.Connection
Dim rsOrders As ADODB.Recordset
Dim rsItems As ADODB.Recordset
Set rsOrders = New ADODB.Recordset
Set cnn = New ADODB.Connection
connstring = "Provider=MSDataShape.1;Data " & _
   "Source=TestDatabase;Initial " & _
   "Catalog=SalesOrderProcess; Connection " & _
   "Timeout=15;DataProvider=SQLOLEDB; User " & _
   "ID=sa; Password=pass"
cnn.Open connstring
rsOrders.Open "SHAPE {Select * From " & _
   "Orders} APPEND ({Select * From Items} " & _
   "as chapter1 RELATE OrderNumber TO " & _
   "OrderNumber)", cnn
Do Until rsOrders.EOF
   Set rsItems = rsOrders("chapter1").Value
   Do Until rsItems.EOF
      Debug.Print rsItems(0), rsItems(1), _
         rsItems(2), rsItems(3)
      rsItems.MoveNext
   Loop
   rsOrders.MoveNext
Loop
End Sub
```

—Brian Hunter, Brooklyn, New York

VB4, VB5
Level: Beginning

Duplicate the Join Function for VB4 and VB5

The native VB6 Split and Join functions have highlighted a number of useful techniques, and now VB5 and VB4 programmers can use this extended facility as well. This code emulates the Join function of VB6 for use in earlier versions. This function takes in an array of information and gives a String as output with delimiters per the user request:

```
Public Function Join(arr As Variant, Optional _
   ByVal delimiter) As String
   Dim sRet As String
   Dim i As Integer
   If IsArray(arr) Then
      If IsMissing(delimiter) Then
         delimiter = " "
      ElseIf Len(CStr(delimiter)) = 0 Then
         delimiter = ""
      Else
         delimiter = CStr(delimiter)
      End If
      For i = LBound(arr) To UBound(arr)
         sRet = sRet & arr(i) & delimiter
      Next i
   End If
   Join = Left(sRet, Len(sRet) - Len(delimiter))
End Function
```

—G. Ajay Kumar, Chennai, India

VB3 and up
Level: Beginning

Store Primary Key in ItemData

Loading a combo/listbox is pretty easy, and determining what the combo/listbox Text property selects is even easier. But if you load a table that might contain duplicate values, you might run into a problem—for example, many people might share the same last name.

Here's the solution. First, load your combo box with a table from your database. A sub such as this works fine, by loading the list with names and storing a lookup key in each item's ItemData property:

```
Public Sub FillComboBox(ctrControl As Control)
   Set rs = db.OpenDatabse("Contact", _
      dbReadOnly)
   If Not rs.EOF Then
      With ctrControl
         Do Until rs.EOF
            .AddItem rs("LastName")
            .ItemData(.NewIndex) = rsTemp("ContactID")
            rs.MoveNext
         Loop
      End With
   End If
   rs.Close
   Set rs = Nothing
End Sub
```

You can now easily determine exactly which name is selected:

```
strSQL = "SELECT * FROM Contact Where " & _
   "ContactID = " & cboMyComboBox.ItemData( _
   cboMyComboBox.ListIndex)
```

—Ken Kilar, Los Angeles, California

VB3 and up
Level: Beginning

Force Tri-State Checkbox Cycling

The CheckBox control in VB supports three positions: Checked, Unchecked, and Grayed. Unfortunately, the default behavior for the control is to cycle between Checked and Unchecked. To set it to Grayed, you must do it programatically.

This code shows you how to cycle between the three positions (the order is Checked->Unchecked->Grayed->Checked ...):

```
Private Sub Check1_Click()
   Static iState As CheckBoxConstants
   Static bUserClick As Boolean
   ' Trap if the user clicked on the control
   ' or if the event was fired because you
   ' changed the value below
   bUserClick = (iState <> Check1.Value)
   ' Prevents you from entering an infinite
   ' loop and getting an Out of Stack Space error
   If bUserClick Then
      Select Case iState
         Case vbChecked
            iState = vbUnchecked
         Case vbUnchecked
            iState = vbGrayed
         Case vbGrayed
            iState = vbChecked
      End Select
      ' This will raise another click event but
      ' your boolean check prevents you from looping
      Check1.Value = iState
   End If
End Sub
```

—Eric Litwin, Thousand Oaks, California

VB3 and up
Level: Intermediate

Use the Immediate Window to Write Repetitive Code

You can stop a program's execution and use the debug window to generate code you can paste into your program. For example, you have a recordset called rs and you wish to manually move the contents into controls on your form or into declared variables. Place a breakpoint after you open the recordset, press Ctrl-G to open the Immediate window, and type this:

```
for each x in rs.Fields : ?"= rs.Fields(""" & _
   x.name & """)" : next
```

When you press Enter, you get one line per field. The output should resemble this:

```
= rs.Fields("Edition")
= rs.Fields("Num")
= rs.Fields("Title")
= rs.Fields("ReaderName")
= rs.Fields("ReaderFrom")
= rs.Fields("Bits16")
= rs.Fields("Bits32")
= rs.Fields("Level")
= rs.Fields("Tip")
```

Copy and paste this output into your code. Now you only need to enter the destination control or variable's name on the left side of the equal signs. If you have a recordset with a large number of fields, this tip is worth its weight in gold. It prevents typing errors and saves time because the field names are pulled right from the recordset.

—Larry Johnson, Trenton, Georgia

VB5, VB6
Level: Beginning

Format Your Version Info

Many professional applications are required to display a version number on all screens to indicate to users which version of the app is currently running. This also helps with configuration management. Here's a function that appends the VB project's version number to a text description passed to the function as input. The version information is embedded in a project by assigning major, minor, and revision values on the Make tab of the Project Properties dialog. Then when you right-click on the resulting EXE file in Windows, go to Properties, and click on the Version tab, the version number matches those on your screens, providing a nice consistency. Putting the function in a standard module—particularly one made of generic reusable functions and subprocedures—allows other developers to plug the module into their projects and use the routine:

```
Public Function GetVersion(strApp As String) _
   As String
   ' Pass in the application name you want
   ' displayed as part of the form's caption. A
   ' blank character and the version number are
   ' appended to the application name
   ' completing the caption.
   GetVersion = strApp & " " & _
      Format(App.Major, "#0") & "." & _
      Format(App.Minor, "#00") & "." & _
      Format(App.Revision, "0000")
End Function
```

Here's a sample call to this function:

```
Dim strVersion As String
strVersion = "Application XYZ Version"
frmMain.Caption = GetVersion(strVersion)
' Set form's caption
```

—Michael T. Hutman, Germantown, Maryland

VB4 32, VB5, VB6
Level: Intermediate

Bind Option Buttons to Data Controls

The Option button is a convenient way to display multiple options from which only one can be selected. One problem is that the Option button cannot be bound to a data control. Here's an easy workaround: Create an array of Option buttons and also create a hidden text field and bind it to your data control. Place this code in your form:

```
Private Sub Option1_Click(Index As Integer)
   Text1.Text = Index
End Sub
Private Sub Text1_Change
   Option1(Val(Text1.Text)).Value = True
End Sub
```

Whenever the value in Text1 is changed by the data control, it sets the Option button of the corresponding index value to True. Whenever the Option button is changed, it stores the corresponding Index in the textbox. Because the textbox is bound to the data control, the value is saved in the database.

—Chris Schneider, Newark, Delaware

VB4, VB5, VB6, DAO 3.x
Level: Intermediate

Execute Parametrized QueryDefs Simultaneously in DAO

In Microsoft Access, you can execute a parameterized query that uses other parameterized queries, as long as their parameter names are the same. Save these queries in an Access database:

```
"QueryOne"
PARAMETERS MyDate DateTime;
SELECT Date1 FROM TableOne WHERE Date1>MyDate;
"QueryTwo"
PARAMETERS MyDate DateTime;
SELECT Date1 FROM TableTwo WHERE Date1>MyDate;
"QueryUnion"
PARAMETERS MyDate DateTime;
SELECT * FROM QueryOne
UNION
SELECT * FROM QueryTwo;
```

You can execute QueryUnion from VB code by passing the MyDate parameter. This example is for DAO 3.5:

```
Sub ExecuteQuery()
Dim db As Database
Dim rs As Recordset
Dim qd As QueryDef
Set db = OpenDatabase("<database name>")
Set qd = db.QueryDefs("QueryUnion")
qd.Parameters(0).Value = CDate("3/1/00")
Set rs = qd.OpenRecordset(dbOpenSnapshot)
' <.....>
rs.Close
db.Close
End Sub
```

—Pavel Maksimuk, Brooklyn, New York

VB4, VB5, VB6
Level: Advanced

Save Expensive Heap Allocations

Fixed-size arrays in local variables use a stack-allocated descriptor as expected, but all the data for an array is allocated on the heap. However, fixed-size arrays embedded in structures are fully stack-allocated. This means that you can save yourself expensive heap allocations by defining a (Private) type with a single fixed-size array element and using a UDT-typed variable in place of the local fixed-size array. You can optimize the number of allocations you need to load a standard module or create a class instance using the same technique at module-level.

—Matt Curland, Redmond, Washington

VB5, VB6
Level: Intermediate

Allow Context-Sensitive Help for Disabled Controls

If you want a form to support context-sensitive help, set the WhatsThisButton and WhatsThisHelp properties on the form to True, and set the WhatsThisHelpID property to a corresponding help-file topic ID for any control on that form for which you want help to be displayed.

Unfortunately, the help isn't shown if the control's Enabled property is set to False. To solve this problem, create a label under the control with the same dimensions, and clear its caption to make it invisible. Set the WhatsThisHelpID property to the same value as the disabled control's property.

—Frank Addati, Melbourne, Australia

VB4, VB5, VB6
Level: Intermediate

Use the ListIndex Property to Store Primary Keys From a Recordset

Here's an easy way to fill a listbox or combobox with names, then retrieve the UserID of that name. This example loads names into a listbox from a SQL Server stored procedure. When you click on a name in the listbox, the Key value is stored in the lngUserID variable. Then you can use the lngUserID variable in other parts of the program to retrieve related information for the selected name. The names are set up as character fields with the UserID being an AutoNumber field and also the primary key. This tip is valid only if you can translate the field value to a number:

```
Private Sub Form_Load()
   Call LoadData(List1)
End Sub
Private Sub List1_Click()
   If List1.ListIndex>=0 Then
      lngUserID = _
         List1.ItemData(List1.ListIndex)
   End If
End Sub
Private Sub LoadData(ByRef obj As Object)
' Assumes the Object is either a ListBox or ComboBox
   Dim com as ADODB.Command
   Dim rs as ADODB.Recordset
   Set com=CreateObject("ADODB.Command")
   Set rs=CreateObject("ADODB.Recordset")
   com.CommandText = "procGetData"
   com.CommandType = adCmdStoredProc
   com.ActiveConnection = strConnect
   Set rs=com.Execute
   obj.Clear
   Do While Not rs.EOF
      obj.AddItem rs!Name
      obj.ItemData(obj.NewIndex) = rs!UserID
      rs.MoveNext
   Loop
   rs.Close
   set rs=Nothing
   set com=Nothing
End Sub
```

—Steve Ramsey, Tyrone, Pennsylvania

VB5, VB6
Level: Intermediate

Use Bitwise Comparison in SQL Server Queries

The newsgroups offer a lot of discussion about bitwise comparison in SQL statements. VB supports true bitwise arithmetic with And, but SQL supports only a logical AND and returns only TRUE or FALSE. Here's a quick way to test against a single bit in SQL:

```
SELECT MyField
FROM MyTable
WHERE (MyTable.MyField \ 2 ^ (MySingleBit - 1) _
   MOD 2 = 1)
```

The \ operator specifies integer division, although you could have used INT (MyTable.MyField / ...) just as easily. MySingleBit is the bit you want to test: 1,2,3,4,5, and so on. More complicated ways of doing this—such as with table joins—might be faster, but this is about as simple as it gets.

—Merv Pate, Houston, Texas

XML Solutions and Strategies:

Get them here first!

How to Choose an XML Serv
XML
MAGAZINE
Display Until 11/27/00
A Fawcette Technical Publication Fall 2000 Vol. 1, No. 4
www.xmlmag.com
.NET=XML
How XML Will Move Microsoft Beyond Windows
Apps to Web Services with SOAP
Serving XML with JavaServer Pages
Expert Voices: .NET's Wascha on XLANG, Hejlsberg on C#
Cocoon Separates Logic from Layout
U.S. $7.95 (CANADA $9.95)
Microsoft .net
FREE Subscription!

XML Magazine is the only independent publication devoted 100-percent to the latest information about using XML for enterprise-level challenges. Created by the same experts who gave you *VBPJ, Exchange & Outlook* and *Java Pro,* this is the one indespensible XML resource that you can't afford to be without. Published every other month, you'll get practical, hands-on articles, tips and enterprise-level solutions, including:

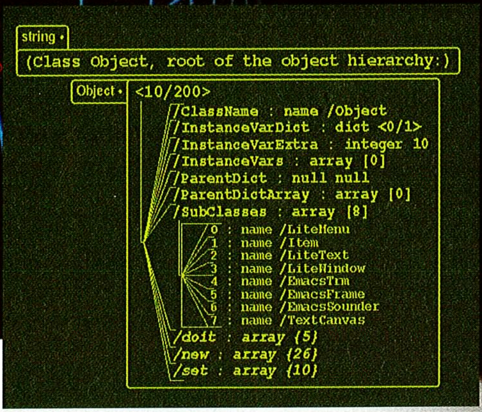
- Pragmatic techniques for developing with XML
- Cutting-edge analysis of industry trends
- Independent product reviews and new product announcements
- Columns by the industry experts who design the technology
- Complete source code examples

Sign up today for your FREE subscription @ www.xmlmag.com/free

FTP FAWCETTE TECHNICAL PUBLICATIONS

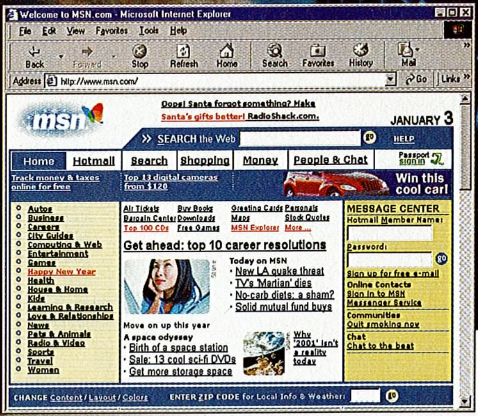
209 Hamilton Avenue, Palo Alto, CA 94301-2500 • Phone: 650.833.7100 • Fax: 650.853.0230

SoftWIRE is to programming what the browser was to the WEB

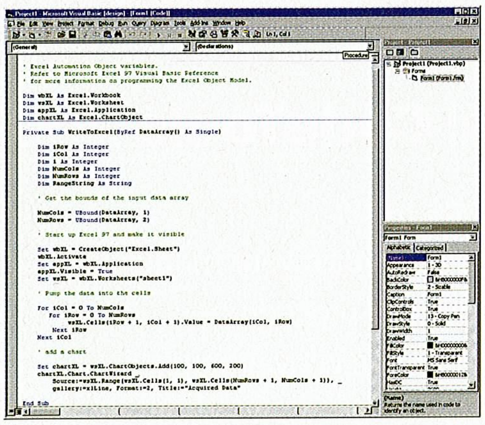


```
string •
(Class Object, root of the object hierarchy:)
Object • <10/200>
         /ClassName : name /Object
         /InstanceVarDict : dict <0/1>
         /InstanceVarExtra : integer 10
         /InstanceVars : array [0]
         /ParentDict : null null
         /ParentDictArray : array [0]
         /SubClasses : array [8]
              0 : name /LiteMenu
              1 : name /Item
              2 : name /LiteText
              3 : name /LiteWindow
              4 : name /EmacsTrm
              5 : name /EmacsFrame
              6 : name /EmacsSounder
              7 : name /TextCanvas
         /doit : array {5}
         /new : array {26}
         /set : array {10}
```

Before the WEB browser, only scientists used the internet, and did it on text-based monitors.

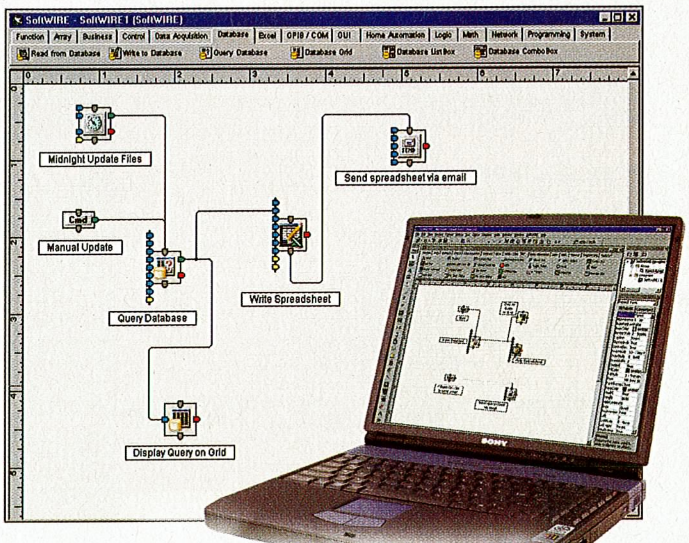

Welcome to MSN.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address http://www.msn.com/
msn
Oops! Santa forgot something? Make Santa's gifts better! RadioShack.com.
JANUARY 3
SEARCH the Web
Home Hotmail Search Shopping Money People & Chat
Win this cool car!
Get ahead: top 10 career resolutions
MESSAGE CENTER
CHANGE Content / Layout / Colors ENTER ZIP CODE for Local Info & Weather:

Now everyone uses the WEB!



```
Project1 - Microsoft Visual Basic [design] - [Form1 (Code)]
(General)                              (Declarations)

' Excel Automation Object variables.
' Refer to Microsoft Excel 97 Visual Basic Reference
' for more information on programming the Excel Object Model.

Dim wbXL As Excel.Workbook
Dim wsXL As Excel.Worksheet
Dim appXL As Excel.Application
Dim chartXL As Excel.ChartObject

Private Sub WriteToExcel(ByRef DataArray() As Single)

    Dim iRow As Integer
    Dim iCol As Integer
    Dim i As Integer
    Dim NumCols As Integer
    Dim NumRows As Integer
    Dim RangeString As String

    ' Get the bounds of the input data array

    NumCols = UBound(DataArray, 1)
    NumRows = UBound(DataArray, 2)

    ' Start up Excel 97 and make it visible

    Set wbXL = CreateObject("Excel.Sheet")
    wbXL.Activate
    Set appXL = wbXL.Application
    appXL.Visible = True
    Set wsXL = wbXL.Worksheets("sheet1")

    ' Pump the data into the cells

    For iCol = 0 To NumCols
       For iRow = 0 To NumRows
            wsXL.Cells(iRow + 1, iCol + 1).Value = DataArray(iCol, iRow)
        Next iRow
    Next iCol

    ' add a chart

    Set chartXL = wsXL.ChartObjects.Add(100, 100, 600, 200)
    chartXL.Chart.ChartWizard _
        Source:=wsXL.Range(wsXL.Cells(1, 1), wsXL.Cells(NumRows + 1, NumCols + 1)), _
        gallery:=xlLine, Format:=2, Title:="Acquired Data"

End Sub
```

Before Visual Basic with SoftWIRE, only programmers programmed.


SoftWIRE - SoftWIRE1 (SoftWIRE)
Function Array Business Control Data Acquisition Database Excel OPIB / COM OUI Home Automation Logic Math Network Programming System
Read from Database Write to Database Query Database Database Grid Database ListBox Database ComboBox
Midnight Update Files
Manual Update
Query Database
Write Spreadsheet
Send spreadsheet via email
Display Query on Grid
SONY

Now anyone can create programs!

SoftWIRE™ 1.0

www.softwiretechnology.com
16 Commerce Blvd., Middleboro, MA 02346
phone: (508) 946-8900 • fax: (508) 946-9500

Download SoftWIRE today — Use it FREE for 60 days — Purchase for only $495