VBZ

The Electronic Journal for Visual Basic Programmers

Copyright 1993 User Friendly, Inc.
Issue 04: July/August 1993

Welcome to Issue 4 of our journal. We are gratified by all the positive feedback we have been receiving from you, and we are convinced that we are on the way to providing the best possible resource for Visual Basic programmers. We continue to battle valiantly with the Windows help system to provide you with as valuable a hypertext document as we can. This issue marks the beginning of searching functionality and a more consistent organization of articles and reference materials, and the review of Q+E MultiLink in our review section contains the first bitmaps we have published in *VBZ*. Let us know if you like the pop-up method we have used to display them. We walk a fine line between a journal and an add-on, and we hope that, with your help, *VBZ* is able to meet both sets of expectations.

Some of your most popular requests have been answered in this issue, including custom cursor functionality and *VBZ's* first custom controls: a list box that displays bitmaps (VBZList), and a mixed attribute label (VBZLabel). We hope you find them unique and useful. Introducing these controls has raised the question of on-line help for the tools, via the hypertext document. As an experiment, VBZLabel provides on-line help, but requires that a copy of VBZ04.HLP reside in either your VB directory or the Windows directory. How would you like to see this handled in the future? A separate HLP file for controls? An INI file in which to specify the location of your *VBZ* issues? How do you organize your issues now? Are they all in the same directory (currently this is required for jumps across issues, as in our "What's Gone Before" section)? Any answers you can provide to these questions will help us serve you better. If you would like an automated survey, we can do that too. Just let us know. Pay special attention to the "What's Coming Up" section this month, and send us all the feedback you can because, as I have said before, our goal is to make *VBZ* the premier resource for Visual Basic programmers. Enjoy!

Jonathan Zuck, President
User Friendly, Inc.
CIS: 76702,1605

Table of Contents

Features

Departments

Creating Custom Cursors

A number of applications, especially graphical ones, employ custom cursors as part of their user interface. The paintbrush program that comes with Windows, for example, changes the cursor to reflect the drawing tool that is being used. A number of design tools change the cursor to reflect the object that is about to be dropped on the design surface.

There is a way to accomplish this using just VB and the Windows API, but the technique has its problems. The best solution -- as it often is -- is a utility we present as another *VBZ* exclusive. This utility, VBZCursr.DLL, provides three functions: **Icon2Cursor**, which creates a handle to a cursor from an icon; **SetControlCursor** which assigns a cursor (using its handle) to a control in your application (using the control's hand); and **DestroyCursor**, which returns the cursor to whatever the default is for the window at hand.

First an API Technique
Using VBZCursr
VBZCursr.DLL Commands

Creating Custom Cursors: First an API Technique

To use API calls to set a cursor, the first thing you need to do is get a handle to a cursor. One way to do this is to retrieve one from a DLL.   This is accomplished by loading the DLL, and then a cursor from the DLL.

```
hLib = LoadLibrary ("MYDLL.DLL")
hCursor = LoadCursor (hLib, "MYCURSOR")
```

Now that you have a cursor handle, it's time to set the cursor to it.   The good news is that it's just one more API call, `SetCursor(hCursor)`.   The bad news is that it will only work for a fraction of a second.   The reason this is the case is that there is a message sent to all windows when the cursor is hovering over them, querying about the cursor to display.   That message is WM_SETCURSOR.   If a control doesn't respond to this message, it gets passed on to its parent, and so on, until there is no more parent, at which point it is passed back to Windows for default processing.   At this point, the class cursor is used.   The class cursor is the "default" cursor for that type of window which is established when the window gets created.   Often it's that arrow pointing North - West.

Accordingly, when you use **SetCursor** to change the cursor, it does in fact change, but since Windows is continually sending that WM_SETCURSOR message to the window under the cursor, it is quickly changed back.   If you wanted to see your new cursor, you would have to do it inside of a loop in which you never yielded to the system; this would not be a very good loop to have running for very long.

Creating Custom Cursors: Using VBZCursr

VBZCursr not only allows you an easy way to assign a cursor to a control, but it also provides a great way to get the handle to a cursor. Its **Icon2Cursor** command makes it possible for you to add all the cursors you'll need as icons. All you need to do is create hidden pictureboxes for each of your cursors, then call **Icon2Cursor**. Its parameters are the icon (the picture's picture property as an integer), and the x and y coordinates for the "hot spot" of the cursor. This is the point with which you point; for example, it would be the fingertip of a hand cursor.

Another way to get an icon is to load a DLL (see the example above) and then use the API call **LoadIcon**. This method gives you access to the icons stored within every Windows program, for example the Moricons.DLL has dozens.

Either way you get an icon, running **Icon2Cursor** gives you a handle to a cursor. To assign that cursor to a control, run **SetControlCursor** with the hWnd of the control and the handle to the cursor.

The IconView application lets you try any icon on your disk as a cursor. It uses a VBZlistbox to allow you to choose an icon, which is placed in Picture2, then calling **Icon2Cursor** and **SetControlCursor** to "assign" the cursor to the button Command2:

```
hCursor% = Icon2Cursor(CInt(Picture2.Picture), x%, x%)
SetControlCursor Command2.hWnd, hCursor
```

Note that you can assign cursors to whatever controls you desire, or even to the form itself, but you should keep track of each hCursor you create, so you can use one other VBZCursr command, **DestroyCursor**, to remove it. IconView uses a module level variable for its cursor handle and makes sure **DestroyCursor** is run before any **Icon2Cursor'**s. The syntax for DestroyCursor is simple:

```
DestroyCursor (hCursor)
```

You can also use SetControlCursor to change the cursor for a particular object or, by passing zero for hCursor, return the cursor to the default. Happy Cursoring! Remember to drop us a line if you have any interesting cursor applications or needs.

Creating Custom Cursors: VBZCursr.DLL Commands

The following are the commands available to create custom cursors using VBZCURSRS.DLL:

Filename

VBZCURSRS.DLL

Commands

Command	**Description**
Icon2Cursor	Creates handle to cursor from an icon
SetControlCursor	Assigns custom cursor to a control
DestroyCursor	Restores cursor to default

Creating Custom Cursors: Icon2Cursor

Purpose

Creates a cursor handle from an icon

Contained in

VBZCURSR.BAS
(Requires VBZCURSR.DLL)

Declaration

```
Declare Function Icon2Cursor Lib "CURSOR.DLL" (ByVal hIcon,
  ByVal X, ByVal Y) As Integer
```

Parameters

Parameter	Description
hIcon	Integer - handle to an icon
X	Integer - x location of hotspot
Y	Integer - y location of hotspot

Return Value

Integer - handle to a cursor

Usage

```
hCursor% = Icon2Cursor(CInt(Picture1.Picture), 1, 1)
```

Creating Custom Cursors: SetControlCursor

Purpose

Designates a cursor to use when over a given control

Contained in

VBZCURSR.BAS
(requires VBZCURSR.DLL)

Declaration

```
    Declare Sub SetControlCursor Lib "CURSOR.DLL" (ByVal hWnd%,
      ByVal hCursor%)
```

Parameters

Parameter	**Description**
hWnd	integer - handle to control being assigned a cursor
hIcon	integer - handle to cursor

Usage

```
    SetControlCursor form1.hWnd, hCursor%
```

Comments

You may change the cursor for a control dynamically by simply calling SetControlCursor with a new hCursor. If you wish to "un-install" the custom cursor for an object, simply pass a zero for hCursor.

Creating Custom Cursors: DestroyCursor

Purpose

Restores a cursor to its default

Contained in

VBZCURSR.BAS
(requires VBZCURSR.DLL)

Declaration

```
    Declare Sub DestroyCursor Lib "User" (ByVal hCursor)
```

Parameters

Parameter	**Description**
hCursor	Integer - handle to a cursor

Usage

```
    DestroyCursor (hCursor)
```

A Mixed Font Label by Brett Foster 70444,135

VB Text boxes and picture boxes don't allow mixed fonts or font attributes, so if you want to emphasize a word or phrase, you need to create a *separate* label for the phrase you want to look different, and place it right where it belongs on the first label.   This is not only a lot of work to set up, but a lot of work to adjust if the underlying text changes.   Of course, doing this dynamically in your program is almost impossible.

VBZLabel doesn't allow mixed typefaces either, but it DOES allow you to predefine a second font style, with its own bold, italic, underline, color, size, and strikethrough properties, and provide an easy way to switch in and out of that second style within a single label.

To use VBZLabel, set the properties for the regular text in the regular way: fontname, fontsize, etc.   Then set the properties for the other text, using font2name, font2size, etc.

Now, when you set the caption text, just surround the text you want to emphasize with the "accent" or backwards, single-quote character, e.g., emphasize `**the words right here**` with the accent mark -- the character under the tilde(~) character.

The VBZLabel application demonstrates the ability to set the font2bold, font2italic, and font2size dynamically; the other properties were set at design-time.   This application is also a good demonstration of the abilities of VBZListBox, a listbox that can have a different font on each line (as well as many other exciting enhancements).

VBZLabel Properties

A Mixed Font Label: VBZLabel Properties

Purpose

Provides a label control with two separate font styles.

Unique Properties

Property	**Description**
Font2Bold	second font's bold attribute, True or False
Font2Color	second font's color attribute, a VB color value
Font2Italic	second font's italic attribute, True or False
Font2Size	second font's height, in points
Font2Strikethru	second font's strikethrough attribute,True or False
Font2Underline	second font's underline attribute,True or False

Usage

Set the attributes for the second font, then, in the caption, set off the text to be highlighted with the accent character.

A Mixed Font Label: Font2bold, Font2italic, Font2strikethru, Font2underline

Purpose

Set the attributes for the second font

Values

True or False

A Mixed Font Label: Font2color

Purpose

Set the color for the second font

Values

Any VB color value

A Mixed Font Label: Font2size

Purpose

Set the height for the second font

Values

Any number of points (1/72 of an inch)

Beyond the ListBox by James Shields 71231,2066

VBZList is a work in progress, but one that already has made major strides past the listbox that comes with VB.   Notable improvements are the abilitities to set the fontname and size for each line, the height of the lines themselves, and the abiltity to have a bitmap for each line.

VBZList Properties
The VBZList Samples

Beyond the ListBox: VBZList Properties

The following are the properties supported by the VBZList custom control. Properties unique to *VBZ* are highlighted. For other properties, see the Visual Basic help file.

Properties

Property	**Description**
Height	
Index	
ItemBackColor	BackColor for an item
ItemDefHeight	Default item height
ItemFontName	Name of font for item
ItemFontSize	Size of font for item
ItemForeColor	ForeColor for an item
ItemImage	Handle to image for item
ItemInvert	Whether to create black bar or inverted bar
Left	
List	
ListCount	
ListIndex	
Tag	
TopIndex	

Beyond the ListBox: ItemDefHeight

<u>Purpose</u>

Set the height for lines to be added to a VBZList

<u>Value</u>

Height in twips

<u>Comments</u>

Unlike most properties of the VBZList, this is not settable for items once they are created. That is, you can't set VBZList1(5).ItemDefHeight. Instead, all lines added once this property is set will be the ItemDef Height until the value is changed.

e.g.:

```
VBZList1.ItemDefHeight = 200
VBZList1.Additem "I will be 200 twips tall"
```

You don't set this for each row, you set this as a default for the control in general, e.g., vbzlist1.itemdefheight = 40, and then when you do an additem, the new line (and all the new lines) will be the new height. However, you can set this to be a different number before each new line.

Beyond the ListBox: ItemBackColor, ItemForeColor

Purpose

Set the background and foreground colors for any row in a VBZList.

Value

Any VB color value

Comments

These properties, which work on one line of the listbox at a time, work like their parallels in other VB controls, with the colors being set with the VB numeric system.

Beyond the ListBox: ItemFontName

Purpose

Set the typeface for any item in a VBZList

Value

A string with any valid font name, like "Times New Roman"

Comments

This property, which works on one line of the listbox at a time, works like its parallel in other VB controls, with the fontname as a string with a valid name.

Beyond the ListBox: ItemFontSize

Purpose

Set the size of the font for any item in a VBZList.

Value

A textheight in points (1/72 of an inch)

Comments

This property, which works on one line of the listbox at a time, works like its parallel in other VB controls, with the font size in points.

Beyond the ListBox: ItemImage

Purpose

Place a bitmap at the left of a line in a VBZList.

Value

A handle to a bitmap.

Comments

You can get a bitmap handle in one of three ways: Using LoadPicture, using a picture property, or using Clipboard.Getdata. This example below shows all three.

```
VBZList1(0).ItemImage = LoadPicture("Mypic.bmp")
VBZList1(1).ItemImage = Picture1.picture
VBZList1(2).ItemImage = Clipboard.Getdata(0)
```

This property allows you assign a bitmap to the beginning of any or all lines in the listbox. You can use the loadpicture command to read a BMP off the disk.

```
vbzlist1.ItemImage(i) =LoadPicture("Test.BMP")
```

Beyond the ListBox: ItemInvert

Purpose

The selection bar in the listbox can be a black bar with white text, or it can be whatever the inverse of the underlying item is. If ItemInvert is true, then you would get a white bar if the underying item were black.

Value

True or False

Comments

This property, which can be True or False, determines whether the current item is indicated with a black bar with white text (False) or whether the bar will be the reverse of the underlying color, in which case it might be a white bar if the background color is dark.

Beyond the ListBox: The VBZList Samples

Two samples here use the VBZList custom control: the VBZLabel demonstration program demonstrates the setting of ItemDefHeight, ItemFontName, ItemBackColor and ItemForeColor.   The IconView program demonstrates the ItemImage property.

The VBZLabel program, in its openform routine, cycles through all of the fonts using screen.FontCount to see how many there are, and screen.Font(i) to get the name of each one. Each time the name of the font gets added to the VBZListBox, the properties of the new line get changed:

The height gets 10 twips taller than the last line

```
vbzlist2.ItemDefHeight = vbzlist2.ItemDefHeight + 10
```

The typeface of the line is set to the font being added.

```
vbzlist2.ItemFontName(i) = screen.Fonts(i)
```

The background color gets changed incrementally.

```
vbzlist2.ItemBackColor(i) = (i * 10)
```

The foreground color is always white.

```
vbzlist2.ItemForeColor(i) = &HFFFFFF
```

The IconView form uses the drive and path controls to point to whatever directory you choose.   Each time the directory is changed, the DIR command is used, with a "*.ICO" mask, to find every icon file in a directory.   The name of each one is added to the list box.   Then it should be a simple matter to add an icon using:

```
vbzlist1.itemimage(i) = loadpicture(afile)
```

But the VBZListBox can't (currently) accept icons.   There's a simple -- but clever-- work around for this limit, but if there's enough interest, we can change VBZListBox.   Just let us know!

For now, though, we need to do the following: add the icon to a hidden image control, and then use an API call, DrawIcon, to copy a bitmap of the icon to a hidden picture control. This command takes the hDC (handle to device context) of the destination, an x and y coordinate, and the icon to copy.

```
DrawIcon picture1.hDC, 0, 0, Image1.Picture
```

Now we have a picturebox on the form with our bitmap in it, so all we need to do is set the ItemImage of the VBZListBox line to the picture   in the picturebox.   Voila, an icon viewer!

The Future of VBZList

What are our plans for VBZList?   Well, what do you need?   Three-D effects? Multiselect?   Multcolumn?   Direct support for icons?   The ability to act as a file box?   Are there any events you would like, for example, easy detection of enter or spacebar? Let us know of any enhancements you desire, as well as interesting applications you find for VBZList.

Creating Rich Text Formatted Documents

You want to create a document from within Visual Basic that will be edited by a user. So far you've been saving text as ASCII. Maybe sticking in some codes, then doing search-and-replace in Word or WordPerfect? That's fine as far as it goes, but isn't there a a better way?

Some people have decoded the formats of popular wordprocessors and written libraries to allow programmers to generate documents in those formats. What we've done is considerably easier and even more useful: RTF.BAS contains routines to allow easy creation of Rich Text Format documents. This is a universal word-processing format that is not the native format for any commercial wordprocessor, but can be imported into many, including Word for Windows. You will be able to create documents with your typeface, fontsize, font attributes and indenting already embedded.

How to Use RTF.BAS
RTF in Use: Resume Wizard
How RTF.BAS works
RTF.BAS Functions

Creating RTF Documents: How to Use RTF.BAS

First open your disk file, using standard VB:

```
Open Myfile for output as #1
```

From this point on, you will be using the filenumber to pass to the RTF routines.

Now you're ready to create Rich Text. The first steps are the two commands used to create the header, RTFSetFonts and RTFSetFormat. Like every RTF command, they begin with the filenumber. **RTFSetFonts** takes a string with all the fonts you will be using, separated by commas:

```
RTFSetFonts 1, "Times New Roman, Arial, Symbol"
```

You've now created an RTF font table, so remember the order. Times New Roman will be font 0 and Arial will be font 1, etc.

RTFSetFormat takes the filenumber (of course) and the left, right, top, and bottom margins, in inches.

```
RTFSetFormat 1, 1.5, 1.5, 1, 1
```

All of the "housekeeping" is out of the way. To send some formatted text, you use RTFPrint, as follows:

```
RTFPrint 1, "Hello there.","\b \i"
```

That line will be bold and italic. Other RTF commands let you change fonts, send paragraph breaks, even indented paragraphs. You can send whatever commands you want in whatever order, but when you're finished with your document, you MUST end with an RTFEnd command before you close the file you are writing.

Creating RTF Documents: How RTF.BAS Works

RTF is a relatively easy-to-decode system of ASCII codes. Each begins with the backslash. For example, the RTFNewPage command just sends the rtf command "\page". Most of these can just get stuck in the document, and it will be acted on when encountered, like "\qc" turns centering on. Every paragraph will be centered until the "\pard" (Paragraph enD) is encountered. Font changes work the same way. You change to Font 0 with "\F0".

The text itself is stored in segments surrounded by curly-braces -- the {} characters. Within those segments, you can put the font attributes, like "\b".

Some Interesting(?) RTF Facts

Measurements are in TWIPS, of which there are 1440 to the inch. The RTF routines provided here use inches and multiply by 1440 so you don't have to. Font sizes, however, are in half-points. The RTF routines use points and multiply by 2.

The Future of RTF.Bas

As with most of the programs you see in *VBZ*, the future is up to you. What feautures would you find valuable? Possible additions range from the ability to change paper size, to color, columns, graphics, and style-sheets. Let us hear from you. We'd like to hear what programs you're writing with RTF, and how it could be improved to help you.

Creating RTF Documents: Commands

Filename

RTF.BAS:

Commands

Command	**Description**
RTFEnd	Write code to end RTF document
RTFFont	Write code for font change, using index in RTF font table
RTFFontSize	Write code for font size change
RTFJustifyCenter	Write code for centering text
RTFJustifyIndent	Write code for indenting paragraph
RTFJustifyNormal	Write code to turn off indenting or centering
RTFNewPage	Write code for new page
RTFNewPar	Write code for new paragraph
RTFPrint	Write text with font attributes
RTFSetFonts	Create RTF font table
RTFSetFormat	Create RTF page definition
RTFTab	Write code for tab

Creating RTF Documents: RTFSetFonts

Purpose

Creates an RTF font table. An RTF document needs this table so that fonts can be switched with a numeric reference to a table entry.

Contained in

RTF.BAS

Declaration

```
    sub RTFSetFonts (filenum, fontstring)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file
fontstring	String - the names of the fonts you will use, separated by commas

Usage

```
    RTFSetFonts 1, "Times New Roman, Arial, Symbol"
```

Comments

In the example above, you would now switch to Times New Roman with the command:

```
    RTFFonts 1,0
```

This is because Times New Roman is the 0 item in the table, Arial is the 1 item and Symbol is the 2 item. (The 1 in the example is the file handle of the document being created.)

RTFSetFormat and **RTFSetFonts** together are needed to initialize an RTF document. RTFEnd is needed to end a document

Creating RTF Documents: RTFSetFormat

Purpose

Writes the RTF codes that define the page

Contained in

RTF.Bas

Declaration

```
Sub RTFSetFormat (filenum, lmargin, rmargin, tmargin, bmargin)
```

Parameters

Parameter	**Description**
filenum	Integer - handle to an open file
lmargin	Single - left margin in inches
rmargin	Single - right margin in inches
tmargin	Single - top margin in inches
bmargin	Single - bottom margin in inches

Usage

```
RTFSetFormat 1, 1.5,1,5, 1,1
```

Comments

RTFSetFormat and RTFSetFonts together are needed to initialize an RTF document. RTFEnd is needed to end a document.

Creating RTF Documents: RTFPrint

Purpose

Save text to disk with RTF font attributes

Contained in

RTF.BAS

Declaration

```
    Sub RTFPrint (filnum,text$,attributes$)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file
text$	String - the text to be printed
Attributes$	String - desired attribute backslash commands; any or all of the below: \b Bold \i Italic \ul Underlined \strike StrikeThrough \v Hidden (inVisible?) \scaps Small Caps

Usage

```
    RTFPrint 1, "Hello there.","\b \i"
```

Creating RTF Documents: RTFNewPar

Purpose

The RTFPrint command doesn't create an RTF carriage-return at the end. Since you are creating a document for a word processor that will be doing linewrapping, you don't want a cr-lf at the end of each line. When you DO need a new paragraph, use the RTFNewPar command.

Contained in

RTF.BAS

Declaration

```
Sub RTFNewPar (filnum)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file

Usage

```
RTFNewPar 1
```

:

Creating RTF Documents: RTFNewPage

<u>Purpose</u>

To write an RTF newpage code to an open file

<u>Contained in</u>

RTF.Bas

<u>Declaration</u>

```
Sub RTFNewPage (filenum)
```

<u>Parameters</u>

Parameter	**Description**
filenum	Integer - a handle to an open file

<u>Usage</u>

```
RTFNewPage 1
```

Creating RTF Documents: RTFFont

Purpose

To change typefaces, referring to the fonts by their position in the font table:

Contained in

RTF.Bas

Declaration

```
Sub RTFFont (filenum)
```

Parameters

Parameter	Description
filenum	Integer - a handle to an open file

Usage

To switch to the first font in the font table:

```
RTFFont 1, 0
```

To switch to the second font in the font table

```
RTFFont 1, 1
```

Creating RTF Documents: RTFFontSize

Purpose

Specifies fontsize for RTF document

Contained in

RTF.Bas

Declaration

```
Sub RTFFontSize (filenum,fontsize)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file
fontsize	Single - size in points

Usage

```
RTFFontSize 1,12
```

Creating RTF Documents: RTFJustifyCenter

Purpose

Writes RTF codes for centering

Contained in

RTF.Bas

Declaration

```
Sub RTFJustifyCenter (filenum)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file

Usage

```
RTFJustifyCenter 1
```

Comments

This code is reversed by RTFJustifyNormal

Creating RTF Documents: RTFJustifyIndent

Purpose

Writes RTF code to indent a paragraph

Contained in

RTF.Bas

Declaration

```
Sub RTFJustifyIndent(filenum, paraIndent, firstlineIndent)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file
paraIndent	Single - Amount in inches to indent paragraph
firstlineIndent	Single - Amount in inches to indent 1st line relative to paragraph

Usage

To indent paragraph 2" and its first line an additional .25"

```
RTFJustifyIndent 1, 2, .25
```

To indent paragraph 1", but not the first line (a hanging indent)

```
RTFJustifyIndent 1, 1, -1
```

Comments

This code is reversed by RTFJustifyNormal

Creating RTF Documents: RTFJustifyNormal

Purpose

Writes RTF code to turn off any indent codes activated by RTFJustifyCenter or RTFJustifyIndent

Contained in

RTF.Bas

Declaration

```
Sub RTFJustifyNormal (filenum)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file

Usage

```
RTFJustifyNormal 1
```

Creating RTF Documents: RTFTab

Purpose

Writes a tab in RTF format to an open file (Sending a chr$(9) doesn't work in RTF).

Contained in

RTF.Bas

Declaration

```
Sub RTFTab (Filenum)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file

Usage

```
RTFTab 1
```

Creating RTF Documents: RTFEnd

Purpose

Write RTF codes to end RTF "session" in an open file.

Contained in

RTF.Base

Declaration

```
Sub RTFEnd (filenum)
```

Parameters

Parameter	**Description**
filenum	Integer - a handle to an open file

Usage

This routine MUST end your document. Run it right before you close your file:

```
RTFEnd 1
Close #1
```

Using RTF.BAS: The Resume Wizard

Although the Resume Wizard doesn't provide as much power or flexibity as a commercial resume generator, it serves as a good sample project for the RTF commands in RTF.BAS: it uses two different fonts, three different font sizes, bold, italic, hidden and smallcaps text, as well as hanging indents. The resumes Resume Wizard creates are ready to be retrieved into Word, printed, and sent to your prospective employer, but *VBZ* makes no warranty, expressed or implied, as to the quality of its output, so you'd better check it before you mail it! The Wizard is also a good demonstration of the power of VB: the heart of the program is a routine that, uncommented, would be only a page long.

Using the Resume Wizard
How the Resume Wizard Works

Using RTF.BAS: Using the Resume Wizard

The Wizard ships with a database already filled with some sample records. If you want, you can press Generate right away. You will be asked for a file name. Create something with an RTF extension, and then call it up with Word for Windows.

To create your own resume, fill in your name and address, then erase the item records until you are told you can start adding.

The "category" field refers to group names like "Education" or "Work Experience".

The "Title" is where you put job titles, like "Director" or "Supervisor". This is also where you put the text for items that aren't jobs. For example, this would be where you put the name of a school you attended. If the category was "References", you could use the "Title" field for "Available upon request"; if the category was "Hobbies", you could use the "Title" field for "Working Hard".

The "Years" field is a textbox so you can type something like "1989-1990".

The other fields are self-explanatory. To change the order of the records, you can change the number in the "Item" field. This won't actually change the record order unless you press the Refresh Record Order button.

Once you've created your records, press Generate, create a file with an RTF extension, and you've got a file you can retrieve into Word, or any other word-processor that can take RTF.

Using RTF.BAS: How the Resume Wizard Works

The information is stored in an Access database file called Resume.MDB. It has two tables. One for the Name and Address, and the other for all the fields of the resume entries. Examine the properties of the data controls and you will see the database names and the recordsources (or tablenames). The fields are linked to these datacontrols with their Datasource (the name of the datacontrol) and the Datafield properties.

Once the database is populated, and the user presses Generate, the form asks for the name of an RTF file, which it opens for output. Then the two RTF header commands are run, RTFSetFonts and RTFSetFormat. The **RTFSetFormat** commands gets its margin settings from the textboxes on the form. The name and address are printed (the address must be printed a line at a time, so this requires some parsing), and finally, the database is cycled through. This is done with the basic structure:

```
Do Until data1.recordset.EOF
   ...
   data1.recordset.MoveNext
Loop
```

Each record is checked to see if it is the first of a new catagory, in which case the category gets printed (otherwise, it does not.) The information can be retrieved from the records in two ways: through the databound textboxes, or with a database command, e.g., `data1.recordset("Employer").` This is a shortcut for `data1.recordset.fields("Employer").`

As to how the information is printed, I refer you to the section on RTF.BAS. There is one formatting trick worth noting here, however: Those paragraphs that contain a catagory are indented paragraphs, with a firstline "outdent" so the category is to the left of the paragraph. Then a tab is sent before the rest of the paragraph. This tabs all the way out to the paragraph margin, ignoring any tabstops on the way.

Feel free to modify this program. Possible additions could include better font choices, the storage of the chosen attributes, multiple resumes, and multiple resume styles. Let us know if you come up with anything, or want us to modify Resume Wizard in any way.

Data Basics by Thomas Wagner

Q+E MULTILINK Sug. Ret. Price: $399
by Pioneer Software
5540 Centerview Drive
Raleigh, NC 27606
Pho: (919) 859-2220
Fax: (919) 859-9334

The potential of Visual Basic as a database development tool, especially in the area of client/server development, has been known for quite some time. It is this potential that provided the impetus for the creation of Q+E MultiLink, an add-on that enables developers to utilize Visual Basic's flexibility and fast development cycle to create client/server applications.

Data Basics: Database Programming in the Visual Basic Environment

How does connecting multiple high-end RDBMS's within your Visual Basic application sound to you? Say, for example, Oracle with Sybase, however unlikely the example may be, or DB2 with Informix, another unlikely pairing. With Q+E Multilink you can do just that. And while these examples are very doubtful, can you picture an organization that may have data stored simultaneously in Oracle, Dbase and Excel? You can access all of those formats, and DBMS's in ONE Visual Basic application, by using Multilink.

Q+E Multilink will let you create applications that are able to access the following databases and formats:

ASCII TEXT
Btrieve
dBase compatible files
Excel Worksheet Files
HP Allbase, HP Image/SQL
IBM DB2
IBM OS/2 Datamanager, AS/400 and SQL/DS
Informix
Ingres
Novell Netware SQL
Oracle
Paradox
Progress
Tandem Non-Stop SQL
Sybase and Microsoft SQL Server products
SQL Base
Terradata
XDB

This portability is made even more attractive because you only write an application once.and then just change one property -- the database name -- to connect your app to a different database. Furthermore,the entire application is multi-user capable. These features make portability painless.

MultiLink's main purpose, aside from enabling a developer to write for a variety of RDBMS applications, is to provide the user with easy access to information. This is primarily accomplished by giving you the tools to create relatively convenient and fast implementations of data queries. It is important to note this fact, since different products have different areas of strength. MultiLink very much excels at creating and executing queries. Pioneer's corporate mission statement probably reads something like: "We will not rest until we can connect to every database format and provide fast access to their respective data."

Data Basics: Grids, Text Boxes, Scroll Bars and VB 3.0, The Hidden Contender

Of course, there are some trade-offs involved in implementing a tool with this kind of portability. The biggest one is speed. Pioneer's tools have not been known as speed demons. This reputation came about due to some difficulties experienced by earlier releases of other Q+E programs. More appropriately,the statement regarding MultiLink's speed is somewhat relative, depending on what you are used to working with. Just for perspective, the Access engine included with VB 3.0, which is most likely the closest direct competitor, is noticeably slower. And ODBC, in its present incarnation, appears slower still. I do not profess to have conducted any benchmark tests. These are just simple observations made with the naked eye. As far as the individual features of each program are concerned, have a look at the attached comparison of VB 3.0 and MultiLink.

Money is always a prime consideration in the evaluation of tools for your production efforts. I know that a few of you will carefully weigh the alternatives between using VB 3.0 to its fullest potential and spending the necessary funds for yet another add-on. That being the case, please keep in mind that VB 3.0 can access eight databases, while MultiLink can reach twenty. Aside from that obvious difference, MultiLink also provides you with twelve data-aware custom controls. These include some that are not available from Microsoft yet, such as a bound query grid. How's that for productivity? Figure 1 shows the Visual Basic toolbox with Multilink installed.

While we are on the subject of query grids, someone may point out that the VISDATA sample application shipping with VB 3.0 shows the utilization of the VB grid control in a database setting. But again even Microsoft's own grid is not databound. This obvious opportunity to provide such a feature is being seized by a number of other vendors. As you are reading this article, there is an interesting development in the works by Sheridan Software, which captured Bill Gates' attention enough for him to play with a demo for over thirty minutes at the last Comdex. Okay, okay, enough gossip for now.

Back to business. Some of the controls that come with MultiLink are, at first glance, very similar to the ones shipping with VB. However, you will find Q+E's to be more useful overall because of the additional database properties and built-in functionality. This is especially noticeablewhen dealing with back-end systems that can be reached only through MultiLink, and not through ODBC.

Among the controls shipping with this release of MultiLink are interesting offers, such as the data-aware radio button and scroll bars. It seems that one feature of the scroll bar control arose out of necessity -- because of a possible shortcoming in the Q+E query grid control. It appears that the query grid cannot be used to enter data: it is read-only. As a result, Q+E creates a "quasi" grid for data entry by creating a number of textbox arrays and, then, using the scroll bar to move records through those text boxes. While giving the appearance of a grid, this achieves an interesting effect, since most of us are used to seeing scrollbars directly attached to a list box or table. Figure 2 illustrates this idea.

The actual production of this "quasi" grid is quite simple; and of course, you can add or delete elements just by adding or deleting elements of the array. The lack of a write/update property in the query grid control, while unfortunate, is more than outweighed by MultiLink's other positive features. Q+E is aware of the need for an editable query grid, and promises such an improvement for release 2.0 of MultiLink in December 1993.

MultiLink Controls, such as check boxes, list boxes, combo boxes, text boxes, command buttons and the picture control, are different from, and sometimes quite improved over, their VB 3.0 counter parts. The properties, events and functions that are attributed to them by Q+E are very unique and easy to use.

Data-Basics: Two-Field Unique Indexes and Dynamic Queries

Among these properties and functions were a couple of particularly interesting refinements. While these small provisions are not especially earth shaking, I found them to be thoughtful and quite nice to have available.

The first one is called the pKey property and is used to set primary indexes. The nice part I'm talking about is the fact that the pKey property allows the programmer to specify more than one field as unique (for indexing purposes). This means you are no longer chained to a field holding a record identification number, unless you like that sort of thing. For example, you could specify the combination of last name and social-security number to be the unique index. Having this extra little bit of convenience is nice. It can be applied to the Q+E check box, combo box, list box, radio button group, text box and query grid.

The other refinement I enjoyed is part of what Pioneer calls the pWhere property. This property is one of the workhorses and also one of the connect and query controls. In the query control, the pWhere property contains the conditions that a record must meet in order to be retrieved by a query. These conditions can be sort orders or groups. You will most often use this property to find specific matching values between a query and the actual data in a table.

To backtrack just a little, when retrieving data through queries, a QBE (Query by Example) facility is especially useful. When creating a QBE form, a developer will most often have the user input or pick the search criteria, such as a "salary field" of $30,000 in a text box (i.e. the user types 30,000). The next step is to pass the contents of the "salary field" text box to a data table in the form of a query. This will produce any matching records. Ordinarily, the process of capturing the input and passing it to the query and table requires a few lines of code. Here is where the pWhere property can help. In designing the query, a developer can specify the contents of the example text box (salary field) with a "wildcard" character. The official term for it is a hook. By using a hook, you can set the pWhere property dynamically. The statement would look something like this:

```
Salary > ?Salary Field
```

The question mark represents the hook. It will see to it that the contents of the "salary field" are used in the query, whatever they may be. The query as written here looks for salaries exceeding the amount specified in the "salary field" (larger than 30,000). Importantly, the technique of using a hook only works with MultiLink field controls, not with standard VB field controls or those of other vendors.

Data-Basics: Other Distinguishing Characteristics

Aside from the custom controls, properties and functions, Multilink distinguishes itself in a number of other aspects. For example, noteworthy is the idea that when using VB 3.0, a developer is allowed only one active query per data control. This results in multiple data connection, or data access controls implemented in order to accommodate multiple active queries. Depending on the situation, this can lead to a drain on your server. MultiLink's solution is to allow multiple queries per each single data connection, making your application more efficient.

Another noteworthy point is the way transaction processing is being handled. While VB 3.0's transaction processing is global, affecting all of your data controls on all forms, Multilink's transactions can be tied to specific data connections or queries. This gives you a much finer degree of control, especially in the development of mission-critical applications.

Transaction processing is often found in corporate environments,where large tables of data are the order of the day. This climate will benefit from another difference between VB 3.0 and Q+E MultiLink. When using Q+E's product to connect to SQL systems, it is not necessary for all tables to have unique indexes. Unfortunately, that *is* a requirement of VB 3.0. For a lot of people that may not be a problem, unless you happen to be the corporate database administrator who keeps getting requests from his VB users to reset the indexes. Can you picture that poor fellow?

Data-Basics: Helpful Utilities

To ease your development cycle, Q+E provides two utilities that aid in the design of client/server (or other) applications.

Included with the distribution disks, is a Database Manager that allows the developer to create, edit, modify and delete files in any of the twenty supported formats from within MultiLink. This flexibility is actually another difference between VB3.0 and MultiLink, since the database manager shipping with VB 3.0 can only access the Foxpro, Paradox, dBase and Access formats. It does not support Oracle or Sybase, the formats that need to be reached via ODBC. The Database Manager is shown in figure 3.

Those of you familiar with other database management utilities will feel right at home with the functionality of this program. Actually, anyone who ever had to create or maintain database files will feel right at home. It's straight forward;except that, in the case of SQL tables, you will need to log on to the server first. However, once you are logged on, it's a breeze to create or modify tables. All field definitions are just a mouse click away, as shown in figure 4.

The second utility program, and in some cases the more important one, is the Query Builder. It is the same clever program found in other Q+E products. It's purpose is to lead you through the design of standard SQL queries -- even if you don't know the language. Needless to say, this utility can come in handy. The Query Builder is shown in figure 5

The Query Builder becomes available at design time through the VB property settings' box . After placing a Q+E connect control on the form in question and connecting to a database, it is possible to specify the tables, fields and query expressions with this tool. Mind you, it is only available **after you connect**. So: place a connect-control and hook-up to your tables. Then, place a query control on the form. In its property settings' box, click the dialog pop-up button ("...") under the pTable, pWhere and pExpr entries. Pushing this button calls the query builder. Believe me, it is done a lot faster than described in writing.

You now have the ability to create, edit or view SQL queries. For example, you may want to specify a number of fields. Nothing could be simpler. Figure 6 shows what happens when you click the "Fields" icon.

The query builder lets you sort records (figure 7), group records, join tables and even check the SQL expressions you are creating (figure 8).

SQL Statements can be cut or copied to the Windows clipboard. You can also perform find-and-replace operations.

Data-Basics: Final Impressions

Pioneer has created a winner with this product. Consider how Microsoft Access has been compared to Powerbuilder, the premier client/server database development program, and then look at some of the user comments regarding the Access engine compared to MultiLink. One could almost say that the combination of VB and MultiLink are in competition with a software product that costs over $3000. This is good cause for enthusiasm.

Think about it. How many times do you wonder which development tools in use today that will be around tomorrow? For example, nobody would want to be in the shoes of those programmers who had spent time and energy learning the Paradox Application Language, only to be faced with a completely different syntax in ObjectPal, the Windows language of Paradox. I can't imagine such a debacle facing a VB developer who wants to standardize around Q+E MultiLink. As I've said in the beginning, the manufacturer will just continue to make this program as adaptable as possible. So if you are looking for a company and program around which you can standardize, you can't get much better than this one.

In the end, it all boils down to productivity. If a tool makes you more productive and hence operate more profitably, use it. If it doesn't do that, then don't use it.

Data-Basics: User Comments

Mike Oden

Mike Oden of Oden Industries, located in Pasadena, California, has used Q+E products for some time. He included MultiLink in a project that was actually showcased at the last dB Expo. This showcase application is a database used by a medical corporation specializing in pathological examinations. The program helps this client with several important aspects of day-to-day logistics, including the management of individual patient records, reports and the billing of services rendered. Mike's application distinguishes itself by being able to incorporate photographs of actual biopsies. In addition, the examining doctor is able to record his comments and observations right alongside the picture to be stored with the patient's information. This greatly improves efficiency and helps the pathologist to complete more work in the same amount of time, thus operating more profitably.

As part of a very competitive local development community, Mike must choose his tools carefully. When asked to comment about MultiLink, he stated, "The program has several almost equally important features. First of all, you can connect different database back-ends in one application. For example, the showcase program utilizes Microsoft SQL and dBase in twenty different files and tables. Secondly, in my own work, MultiLink was noticeably faster than ODBC and the Access engine, so much so that there was no question at all which program I would use. Then there are the actual features of the controls that come with MultiLink, such as the databound list box. With MultiLink, I don't have to write an AddItem loop that puts the results of queries into a list box, as I would have to in VB. Lastly, the company has been around a while and has been working with this technology. Consequently, there are a lot fewer bugs than a newer vendor might experience."

Jim Thompson

Jim Thomson, an independent consultant presently contracting with the Fisher-Rosemount division of Emerson Electric, is using Q+E Multilink in the development of a sales and marketing decision-support system. The application is a Microsoft Windows client to a Microsoft SQL Server under OS/2.

When asked about his experience with the product,Jim said, "I initially selected MultiLink for its ability to connect to a PROGRESS back-end database. However, PROGRESS was ultimately deemed to be too slow, particularly with set queries. I called Q+E on the PROGRESS performance issues, and talked to the individual who wrote the PROGRESS interface. He explained that the performance issues were due to PROGRESS' host language interface, and gave me the name of a representative at PROGRESS Software. I contacted this individual who confirmed that the host language interface was indeed slow and would remain so until the next release of PROGRESS (version 7). Throughout this process, I found the Q+E personnel courteous, responsive, and knowledgeable." Because of these difficulties with PROGRESS, we changed our server database to Microsoft SQL Server, without the need to change any of the code we had developed so far, with the exception of a single connect statement."

With regard to the obvious competition between Q+E and VB 3.0, Jim stated, "Microsoft's inclusion of the Access 1.1 engine in VB 3.0 has brought into question the need for MultiLink. However, MultiLink's inclusion of Query-by-Example capability and bound grid control has proven to be a great tool for our development efforts. Also, we have found MultiLink to be more stable than Microsoft's ODBC drivers." Interestingly enough, Jim also mentioned that he found Q+E's tech support personnel more accessible and knowledgeable than Microsoft's.

Data-Basics: MultiLink vs. VB 3.0

Topic	Visual Basic 3.0	Q+E MultiLink/VB
Databases supported	MS Access, FoxPro, dBase, Paradox (3&3.5), Btrieve, SQL Server, Oracle(SQL Server and Oracle are only available in the Professional Edition)	Oracle, SQL Server, dBase, INGRES,Paradox (4.0), Sybase, AS/400 (SQL400)*, Btrieve, OS/2 DBM, DB2*, INFORMIX, Netware SQL, PROGRESS, SQLBase, XDB, SQL/DS*, Tandem NonStop SQL*, Teradata, Excel .XLS files, Text files, HP ALLBASE/SQL* and HP IMAGE/SQL and gateways, IBM DDC/2, Micro Decisionware and Sybase Net-Gateway* requires Gateway
Database Control Types	Text, Check Box, Picture, Data Label, Masked Edit	Text, Check Box, Picture, Query, List Box, Combo Box, Radio Button, Query Grid, Command Buttons, Scroll Bars and Connect Controls
Control Arrays	No - you can only display one record at a time on a form using the database controls.	Yes - you can display any number of records on a form at one time without writing code. Fully supports control arrays.
DB Command Button Controls to perform database operations directly	No	Yes - any database operation can be assigned to a button without writing code.
DB Scroll Bar Controls	No - must use Data control and arrow buttons on Data control; no stand alone scroll bar.	Yes - scroll bar can be used to move through records in a database without writing code.
Database Control Features	Can link Controls to database fields.	Can link Controls to database fields, and can format numbers as well as date values using format strings without writing code.
Query by Example	No	Yes - allows end user of application to use QBE to query for specific records.
Query Builder Utility	No	Yes - allows the developer to build queries and view/edit their SQL statements without knowing SQL language.

Picture Controls	Picture must be stored in databaseOnly BMP format supported.	Pictures can be stored in the database itself, or the name of a specific field containing the pictures can be stored in the database.BMP and MetaFile formats are supported.
Cross Form Control Support	No	Yes - all controls support cross-form referencing, allowing developers to split records and share connections and data across forms
Combo and List Box Controls filled via bound database fields	No	Yes - the combo and list boxes can be filled with directly bound database fields.
Database Manger Utility	Yes - however it does not support SQL databases.	Yes - all database formats fully supported
Transaction Processing Support	Yes - however transactions are only supported for SQL databases which themselves need to support transactions.	Yes - transactions supported for all formats including dBase, Btrieve and all SQL databases.
Transaction Mechanism	Transactions are global to all active databases in an application.	Transactions based on connection level, providing more control for client/server applications.

Provided by the Product Management of Q+E Software

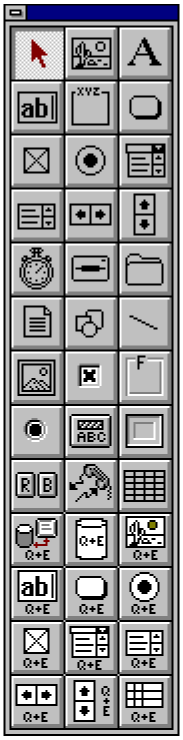
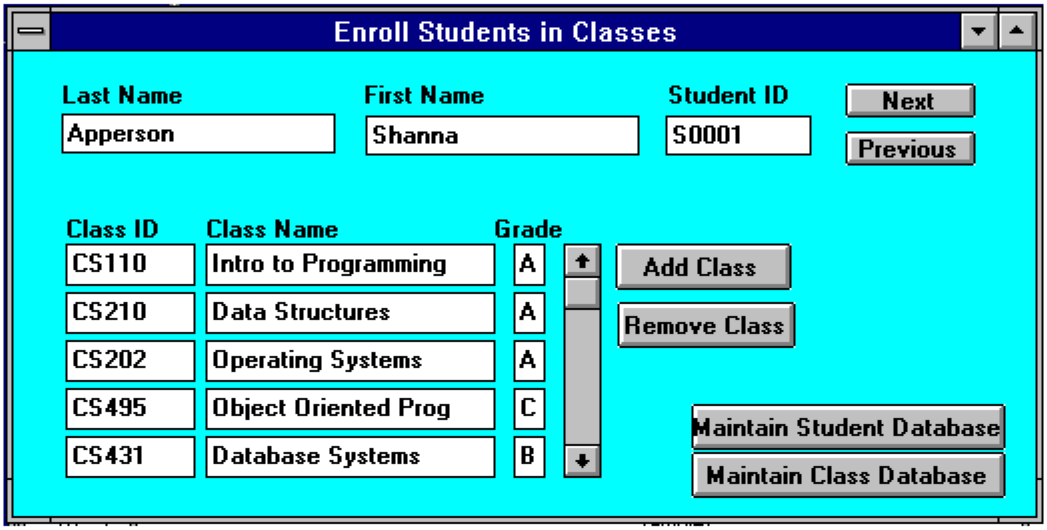


Figure 1: The Visual Basic Toolbox with MultiLink Installed



Last Name: Apperson

First Name: Shanna

Student ID: S0001

Class ID	Class Name	Grade
CS110	Intro to Programming	A
CS210	Data Structures	A
CS202	Operating Systems	A
CS495	Object Oriented Prog	C
CS431	Database Systems	B

Figure 2: The Text Array and Scrollbars in a Sample Application

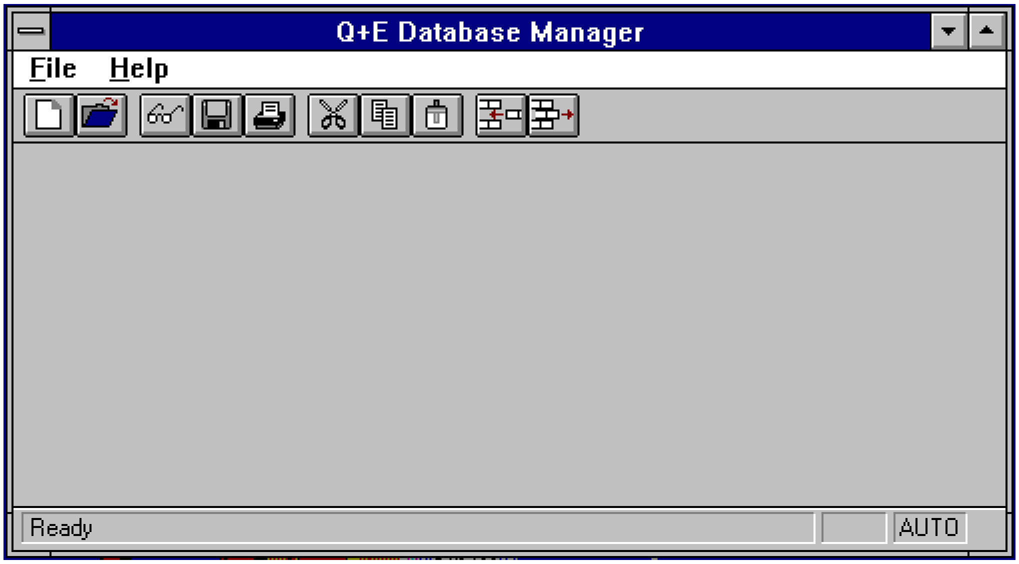
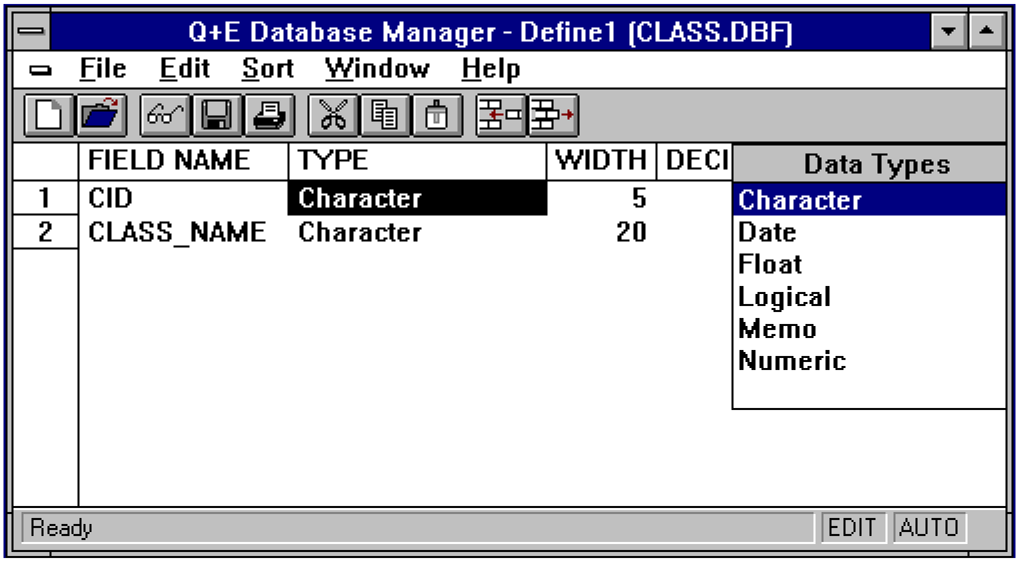
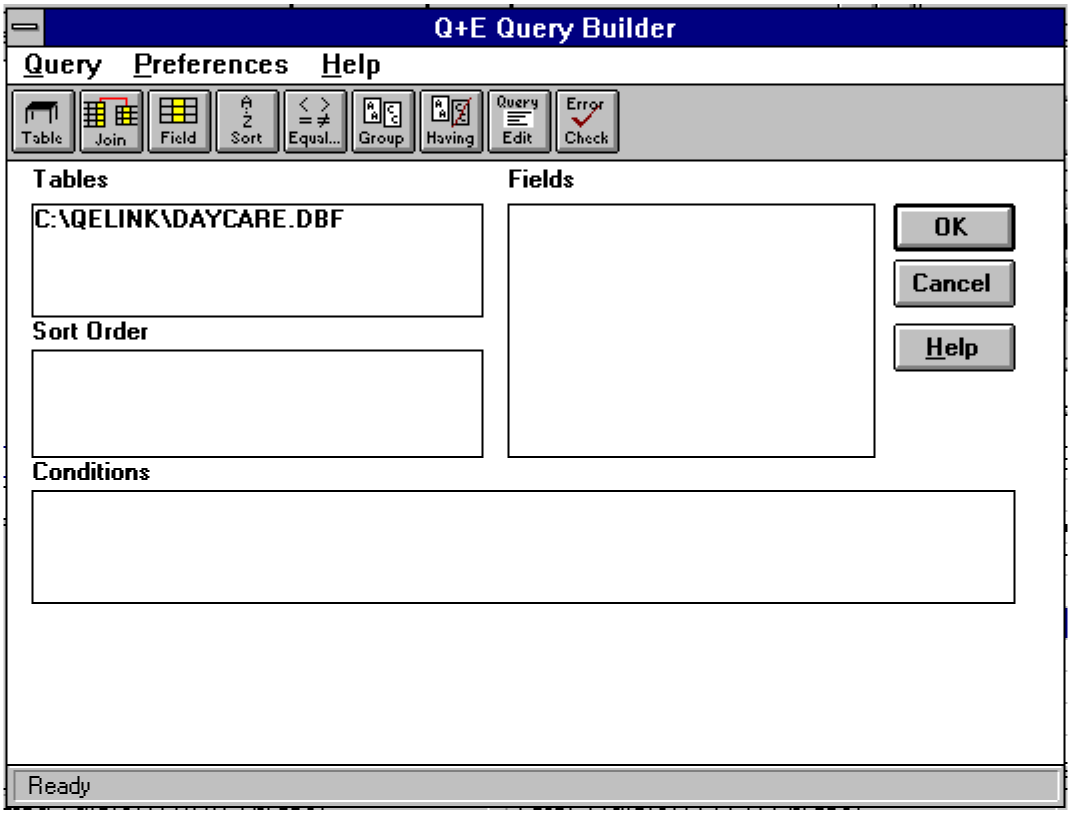


Figure 3: MultiLink Database Manager



	FIELD NAME	TYPE	WIDTH	DECI
1	CID	Character	5	
2	CLASS_NAME	Character	20	

Figure 4: Database Manager and Field Definition



C:\QELINK\DAYCARE.DBF

Figure 5: The Query Builder

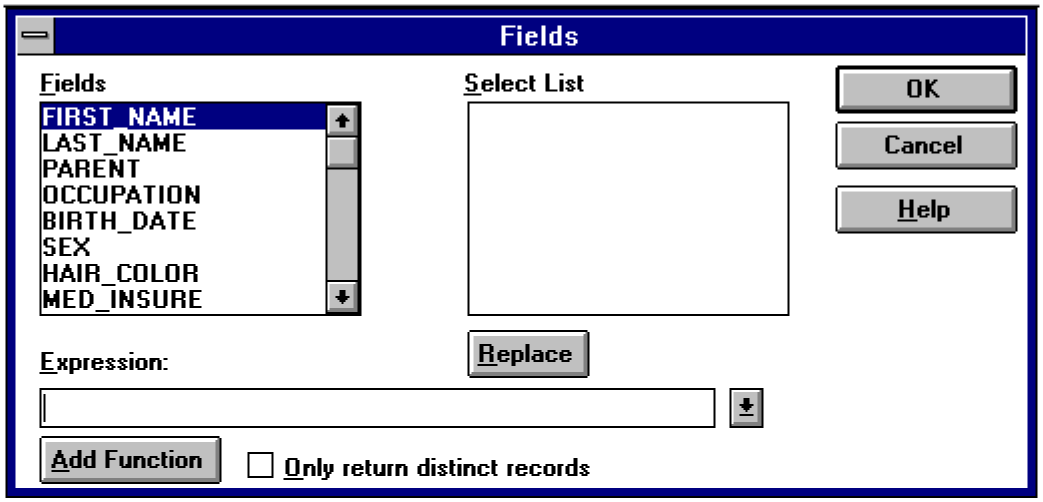


Figure 6: The Query Builder -- Selecting Fields

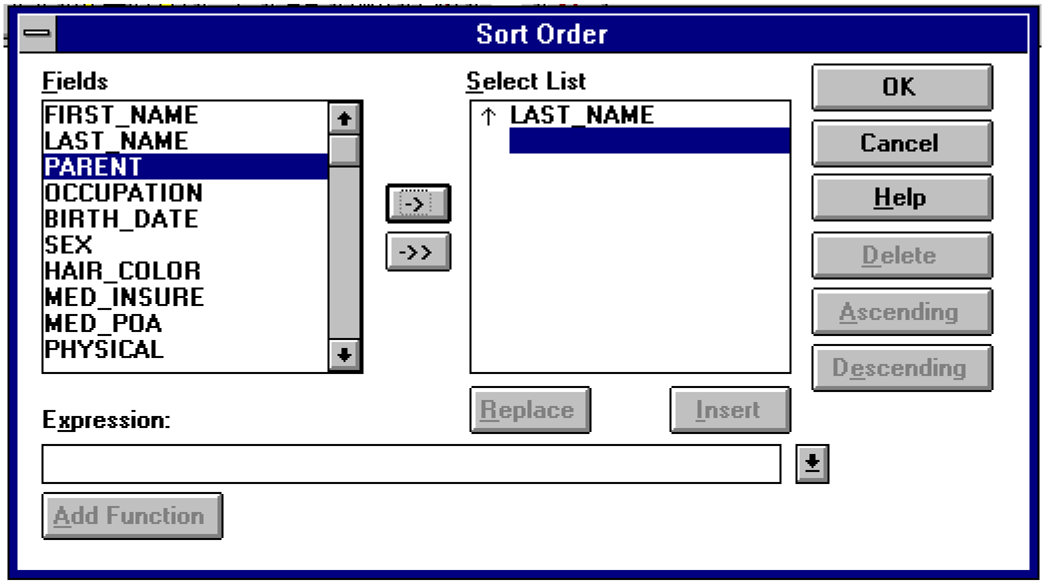


Figure 7: The Query Builder -- Set Sort Order

Figure 8: The Query Builder -- Check SQL Code

What's Coming Up?

While we have a lot already in the works, the content of *VBZ* is largely up to you, the reader, to dictate. If you are having a problem, send it in. If you want more of a particular type of article -- beginner, advanced, more DLLs, more VB code -- let us know and we'll do our best to accommodate your needs and wants.   Here's our list thus far:

VB Developer's Utilities

Button Bitmap Builder
Palette Builder (for PicClip among other things)
Stub Replacement
Object Manager
Code Generators/Wizards
. . . and much more!

Visual Basic Techniques

Calling DLL functions
Standardizing Your VisualBasic Interfaces
Advanced Printing
Metafile Creation
An Improved FindWindow Function
Waiting for other Apps to Execute
. . . and much more!

Dynamic Link Libraries

SendKeys function for DOS Applications
Additional VBZUTIL functions
. . . and much more!

Custom Controls

Generic Subclassing Control
Clipboard Viewer Control
Huge Scrollbars
Enhancements to VBZList (icon support, multi-column)
Enhancements to VBZLabel (more flexibility)
. . . and many other specialized controls!

Improved Help Files

Binary Sample Extraction
Improved Printing
Cross Issue Searching
Setup Program (for Program Manager and INI settings)

The rest is up to you!   Be sure to let us know which of these things are of greatest interest to you so that we can bump them to the top of the list to complete.

What's Gone Before?

VB Developer's Utilities
Dropping Leftover DLLs (VBZ02)
Pop Up Color Selection (VBZ02)
Printing Browse Sequences in WinHelp (VBZ03)
Simplified Formatted Printing (VBZ03)
Screen Saver Wizard (VBZ03)

Visual Basic Techniques
Accessing Private INI Files (VBZ03)
Accessing the Common Color Dialog (VBZ02)
Aligning Controls (VBZ03)
Creating a Drag 'n Drop Client Application (VBZ01)
Creating Dialog Boxes (VBZ01)
Creating Modeless Dialog in VB (VBZ02)
Creating Windows 3.1 Screen Savers (VBZ01)
Screen Saver Wizard (VBZ03)

Dynamic Link Libraries
Aldus Format Metafiles (VBZ01)
A PLAY Command for Visual Basic (VBZ01)
Apps That Tile and Cascade (VBZ02)
Creating a Drag 'n Drop Server Application (VBZ01)
Musical MsgBox and Beep Commands (VBZ01)
Status Bar Help on Controls and Cursors (VBZ01)
System Level Hotkeys (VBZ01)
The QuickBasic Function Library (VBZ02)

Reviews
Doc-To-Help (VBZ02)
3-D Widgets (VBZ02)

\BFeatures\b

\BDepartments\b

A number of applications, especially graphical ones, employ custom cursors as part of their user interface. The paintbrush program that comes with Windows, for example, changes the cursor to reflect the drawing tool that is being used. A number of design tools change the cursor to reflect the object that is about to be dropped on the design surface.

There is a way to accomplish this using just VB and the Windows API, but the technique has its problems. The best solution - as it often is - is a utility we present as another \I*VBZ\i* exclusive. This utility, VBZCursr.DLL, provides three functions: \B**Icon2Cursor\b**, which creates a handle to a cursor from an icon; \B**SetControlCursor\b** which assigns a cursor (using its handle) to a control in your application (using the control's hand); and \B**DestroyCursor\b**, which returns the cursor to whatever the default is for the window at hand.

~~First an API Technique~~
~~Using VBZCursr~~
~~VBZCursr.DLL Commands~~

\BCreating Custom Cursors: First an API Technique\b

To use API calls to set a cursor, the first thing you need to do is get a handle to a cursor. One way to do this is to retrieve one from a DLL. This is accomplished by loading the DLL, and then a cursor from the DLL.

```
hLib = LoadLibrary ("MYDLL.DLL")
hCursor = LoadCursor (hLib, "MYCURSOR")
```

Now that you have a cursor handle, it's time to set the cursor to it. The good news is that it's just one more API call, `SetCursor(hCursor)`. The bad news is that it will only work for a fraction of a second. The reason this is the case is that there is a message sent to all windows when the cursor is hovering over them, querying about the cursor to display. That message is WM_SETCURSOR. If a control doesn't respond to this message, it gets passed on to its parent, and so on, until there is no more parent, at which point it is passed back to Windows for default processing. At this point, the class cursor is used. The class cursor is the "default" cursor for that type of window which is established when the window gets created. Often it's that arrow pointing North - West.

Accordingly, when you use \B**SetCursor\b** to change the cursor, it does in fact change, but since Windows is continually sending that WM_SETCURSOR message to the window under the cursor, it is quickly changed back. If you wanted to see your new cursor, you would have to do it inside of a loop in which you never yielded to the system; this would not be a very good loop to have running for very long.

\BCreating Custom Cursors: Using VBZCursr\b

VBZCursr not only allows you an easy way to assign a cursor to a control, but it also provides a great way to get the handle to a cursor. Its \B**Icon2Cursor\b** command makes it possible for you to add all the cursors you'll need as icons. All you need to do is create hidden pictureboxes for each of your cursors, then call \B**Icon2Cursor\b**. Its parameters are the icon (the picture's picture property as an integer), and the x and y coordinates for the "hot spot" of the cursor. This is the point with which you point; for example, it would be the fingertip of a hand cursor.

Another way to get an icon is to load a DLL (see the example above) and then use the API call \B**LoadIcon\b**. This method gives you access to the icons stored within every Windows program, for example the Moricons.DLL has dozens.

Either way you get an icon, running \B**Icon2Cursor\b** gives you a handle to a cursor. To assign that cursor to a control, run \B**SetControlCursor\b** with the hWnd of the control and the handle to the cursor.

The IconView application lets you try any icon on your disk as a cursor. It uses a ~~VBZlistbox~~ to allow you to choose an icon, which is placed in Picture2, then calling \B**Icon2Cursor\b** and \B**SetControlCursor\b** to "assign" the cursor to the button Command2:

```
    hCursor% = Icon2Cursor(CInt(Picture2.Picture), x%, x%)
    SetControlCursor Command2.hWnd, hCursor
```

Note that you can assign as many cursors to whatever controls you desire, or even to the form itself, but you should keep track of each hCursor you create, so you can use one other VBZCursr command, \B**DestroyCursor\b**, to remove it. IconView uses a global variable for its cursor handle and makes sure \B**DestroyCursor\b** is run before any \B**Icon2Cursor's**\b. The syntax for DestroyCursor is simple:

```
    DestroyCursor (hCursor)
```

Happy Cursoring! Remember to drop us a line if you have any interesting cursor applications or needs.

\BCreating Custom Cursors: VBZCursr.DLL Commands\b

The following are the commands available to create custom cursors using VBZCURSRS.DLL:

\UFilename\u

VBZCURSRS.DLL

\UCommands\u

\BCommand\b	**\BDescription\b**
~~Icon2Cursor~~	Creates handle to cursor from an icon

~~SetControlCursor~~	Assigns custom cursor to a control
~~DestroyCursor~~	Restores cursor to default

\BCreating Custom Cursors: Icon2Cursor\b

\UPurpose\u

Creates a cursor handle from an icon

\UContained in\u

VBZCURSR.BAS
(Requires VBZCURSR.DLL)

\UDeclaration\u

```
Declare Function Icon2Cursor Lib "CURSOR.DLL" (ByVal hIcon,
  ByVal X, ByVal Y) As Integer
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
hIcon	Integer - handle to an icon
X	Integer - x location of hotspot
Y	Integer - y location of hotspot

\UReturn Value\u

Integer - handle to a cursor

\UUsage\u

```
hCursor% = Icon2Cursor(CInt(Picture1.Picture), 1, 1)
```

\BCreating Custom Cursors: SetControlCursor\b

\UPurpose\u

Designates a cursor to use when over a given control

\UContained in\u

VBZCURSR.BAS
(requires VBZCURSR.DLL)

\UDeclaration\u

```
Declare Sub SetControlCursor Lib "CURSOR.DLL" (ByVal hWnd,
  ByVal hIcon)
```

\UParameters\u

\B**Parameter\b**	\B**Description\b**
hWnd	integer - handle to control being assigned a cursor
hIcon	integer - handle to icon

\UUsage\u

```
        SetControlCursor form1.hWnd, hCursor%
```

\BCreating Custom Cursors: DestroyCursor\b

\UPurpose\u

Restores a cursor to its default

\UContained in\u

VBZCURSR.BAS
(requires VBZCURSR.DLL)

\UDeclaration\u

```
        Declare Sub DestroyCursor Lib "User" (ByVal hCursor)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
hCursor	Integer - handle to a cursor

\UUsage\u

```
        DestroyCursor (hCursor)
```

VB Text boxes and picture boxes don't allow mixed fonts or font attributes, so if you want to emphasize a word or phrase, you need to create a \I*separate\i* label for the phrase you want to look different, and place it right where it belongs on the first label. This is not only a lot of work to set up, but a lot of work to adjust if the underlying text changes. Of course, doing this dynamically in your program is almost impossible.

VBZLabel doesn't allow mixed typefaces either, but it DOES allow you to predefine a second font style, with its own bold, italic, underline, color, size, and strikethrough properties, and provide an easy way to switch in and out of that second style within a single label.

To use VBZLabel, set the properties for the regular text in the regular way: fontname, fontsize, etc. Then set the properties for the other text, using font2name, font2size, etc.

Now, when you set the caption text, just surround the text you want to emphasize with the "accent" or backwards, single-quote character, e.g., emphasize `\B**the words right here\b**` with the accent mark - the character under the tilde(~) character.

The VBZLabel application demonstrates the ability to set the font2bold, font2italic, and font2size dynamically; the other properties were set at design-time. This application is also a good demonstration of the abilities of VBZListBox, a listbox that can have a different font on each line (as well as many other exciting enhancements).

\BA Mixed Font Label: VBZLabel.VBX\b

\UPurpose\u

Provides a label control with two separate font styles.

\UUnique Properties\u

\B**Property\b**	**\BDescription\b**
~~font2bold~~	second font's bold attribute, True or False
~~font2color~~	second font's color attribute, a VB color value
~~font2italic~~	second font's italic attribute, True or False
~~font2size~~	second font's height, in points
~~font2strikethru~~	second font's strikethrough attribute,True or False
~~font2underline~~	second font's underline attribute,True or False

\UUsage\u

Set the attributes for the second font, then, in the caption, set off the text to be highlighted with the accent character.

\BA Mixed Font Label: VBZLabel Properties\b

The following are the properties unique to VBZLabel:

~~Font2bold, Font2italic, Font2strikethru, Font2underline~~
~~Font2color~~
~~Font2size~~

\BA Mixed Font Label: Font2bold, Font2italic, Font2strikethru, Font2underline\b

\UPurpose\u

Set the attributes for the second font

\UValues\u

True or False

\BA Mixed Font Label: Font2color\b

\UPurpose\u

Set the color for the second font

\UValues\u

Any VB color value

\BA Mixed Font Label: Font2size\b

\UPurpose\u

Set the height for the second font

\UValues\u

Any number of points (1/72 of an inch)

VBZList is a work in progress, but one that already has made major strides past the listbox that comes with VB.   Notable improvements are the abilities to set the fontname and size for each line, the height of the lines themselves, and the ability to have a bitmap for each line.

~~VBZList Properties~~
~~The VBZList Samples~~
~~VBZList.VBX~~

\BBeyond the ListBox: VBZList Properties\b

The following are the properties supported by the VBZList custom control.   Properties unique to VBZ are highlighted.   For other properties, see the Visual Basic help file.

\UProperties\u

\B**Property\b**	**\BDescription\b**
Height	
Index	
~~ItemBackColor, ItemForeColor~~	Item color properties
~~ItemDefHeight~~	Default item height
~~ItemFontName~~	Name of font for item
~~ItemFontSize~~	Size of font for item
~~ItemImage~~	Handle to image for item
~~ItemInvert~~	Whether to create black bar or inverted bar
Left	
List	
ListCount	
ListIndex	
Tag	
TopIndex	

\BBeyond the ListBox: ItemDefHeight\b

\UPurpose\u

Set the height for lines to be added to a VBZList

\UValue\u

Height in twips

\UComments\u

Unlike most properties of the VBZList, this is not settable for items once they are created. That is, you can't set VBZList1(5).ItemDefHeight.   Instead, all lines added once this property is set will be the ItemDef Height until the value is changed.

e.g.:

```
        VBZList1.ItemDefHeight = 200
        VBZList1.Additem "I will be 200 twips tall"
```

You don't set this for each row, you set this as a default for the control in general, e.g., vbzlist1.itemdefheight = 40, and then when you do an additem, the new line (and all the new lines) will be the new height.   However, you can set this to be a different number before each new line.

\BBeyond the ListBox: ItemBackColor, ItemForeColor\b

\UPurpose\u

Set the background and foreground colors for any row in a VBZList.

\UValue\u

Any VB color value

\UComments\u

These properties, which work on one line of the listbox at a time, work like their parallels in other VB controls, with the colors being set with the VB numeric system.

\BBeyond the ListBox: ItemFontName\b

\UPurpose\u

Set the typeface for any item in a VBZList

\UValue\u

A string with any valid font name, like "Times New Roman"

\UComments\u

This property, which works on one line of the listbox at a time, works like its parallel in other VB controls, with the fontname as a string with a valid name.

\BBeyond the ListBox: ItemFontSize\b

\UPurpose\u

Set the size of the font for any item in a VBZList.

\UValue\u

A textheight in points (1/72 of an inch)

\UComments\u

This property, which works on one line of the listbox at a time, works like its parallel in other VB controls, with the font size in points.

\BBeyond the ListBox: ItemImage\b

\UPurpose\u

Place a bitmap at the left of a line in a VBZList.

\UValue\u

A handle to a bitmap.

\UComments\u

You can get a bitmap handle in one of three ways: Using LoadPicture, using a picture property, or using Clipboard.Getdata.   This example below shows all three.

```
VBZList1(0).ItemImage = LoadPicture("Mypic.bmp")
VBZList1(1).ItemImage = Picture1.picture
VBZList1(2).ItemImage = Clipboard.Getdata(0)
```

This property allows you assign a bitmap to the beginning of any or all lines in the listbox.   You can use the loadpicture command to read a BMP off the disk.

```
vbzlist1.ItemImage(i) =LoadPicture("Test.BMP")
```

\BBeyond the ListBox: ItemInvert\b

\UPurpose\u

The selection bar in the listbox can be a black bar with white text, or it can be whatever the inverse of the underlying item is.   If ItemInvert is true, then you would get a white bar if the underying item were black.

\UValue\u

True or False

\UComments\u

This property, which can be True or False, determines whether the current item is indicated with a black bar with white text (False) or whether the bar will be the reverse of the underlying color, in which case it might be a white bar if the background color is dark.

\BBeyond the ListBox: The VBZList Samples\b

Two samples here use the VBZList custom control: the VBZLabel demonstration program demonstrates the setting of ItemDefHeight, ItemFontName, ItemBackColor and

ItemForeColor   The IconView program demonstrates the ItemImage property.

The VBZLabel program, in its openform routine, cycles through all of the fonts using screen.FontCount to see how many there are, and screen.Font(i) to get the name of each one. Each time the name of the font gets added to the VBZListBox, the properties of the new line get changed:

The height gets 10 twips taller than the last line

```
        vbzlist2.ItemDefHeight = vbzlist2.ItemDefHeight + 10
```

The typeface of the line is set to the font being added.

```
        vbzlist2.ItemFontName(i) = screen.Fonts(i)
```

The background color gets changed incrementally.

```
        vbzlist2.ItemBackColor(i) = (i * 10)
```

The foreground color is always white.

```
        vbzlist2.ItemForeColor(i) = &HFFFFFF
```

The IconView form uses the drive and path controls to point to whatever directory you choose.   Each time the directory is changed, the DIR command is used, with a "*.ICO" mask, to find every icon file in a directory.   The name of each one is added to the list box.   Then it should be a simple matter to add an icon using:

```
        vbzlist1.itemimage(i) = loadpicture(afile)
```

But the VBZListBox can't (currently) accept icons.   There's a simple - but clever- work around for this limit, but if there's enough interest, we can change VBZListBox.   Just let us know!

For now, though, we need to do the following: add the icon to a hidden image control, and then use an API call, DrawIcon, to copy a bitmap of the icon to a hidden picture control. This command takes the hDC (handle to device context) of the destination, an x and y coordinate, and the icon to copy.

```
        DrawIcon picture1.hDC, 0, 0, Image1.Picture
```

Now we have a picturebox on the form with our bitmap in it, so all we need to do is set the ItemImage of the VBZListBox line to the picture   in the picturebox.   Voila, an icon viewer!

\BThe Future of VBZList\b

What are our plans for VBZList?   Well, what do you need?   Three-D effects? Multiselect?   Multcolumn?   Direct support for icons?   The ability to act as a file box?   Are there any events you would like, for example, easy detection of enter or spacebar?   Let us know of any enhancements you desire, as well as interesting applications you find for VBZList.

\BBeyond the ListBox: VBZList.VBX\b

\UPurpose\u

Extend the abilities of listboxes with as many properties as people need, like font attributes and bitmaps.

\UUnique Properties\u

\B**Property\b**	\B**Value\b**
~~ItemDefHeight~~	Height of next added line, in twips
~~ItemBackColor~~	Backcolor of each item, any VB color value
~~ItemForeColor~~	Forecolor of each item, any VB color value
~~ItemFontName~~	Typeface for each item, a string with valid font name
~~ItemFontSize~~	Textheight for each item, in points
~~ItemImage~~	Handle to a bitmap for the beginning of each item, a handle to bitmap
~~ItemInvert~~	Whether to make selection bar black or the reverse of the line, True or False

You want to create a document from within Visual Basic that will be edited by a user. So far you've been saving text as ASCII. Maybe sticking in some codes, then doing search-and-replace in Word or WordPerfect? That's fine as far as it goes, but isn't there a a better way?

Some people have decoded the formats of popular wordprocessors and written libraries to allow programmers to generate documents in those formats. What we've done is considerably easier and even more useful: RTF.BAS contains routines to allow easy creation of Rich Text Format documents. This is a universal word-processing format that is not the native format for any commercial wordprocessor, but can be imported into many, including Word for Windows. You will be able to create documents with your typeface, fontsize, font attributes and indenting already embedded.

~~How to Use RTF.BAS~~
~~RTF in Use: Resume Wizard~~
~~How RTF.BAS works~~
~~RTF.BAS Functions~~

\BCreating RTF Documents: How to Use RTF.BAS\b

First open your disk file, using standard VB:

```
Open Myfile for output as #1
```

From this point on, you will be using the filenumber to pass to the RTF routines.

Now you're ready to create Rich Text. The first steps are the two commands used to create the header, ~~RTFSetFont~~ and ~~RTFSetFormat~~. Like every RTF command, they begin with the filenumber. \B**RTFSetFonts\b** takes a string with all the fonts you will be using, separated by commas:

```
RTFSetFonts 1, "Times New Roman, Arial, Symbol"
```

You've now created an RTF font table, so remember the order. Times New Roman will be font 0 and Arial will be font 1, etc.

\BRTFSetFormat\b takes the filenumber (of course) and the left, right, top, and bottom margins, in inches.

```
RTFSetFormat 1, 1.5, 1.5, 1, 1
```

All of the "housekeeping" is out of the way. To send some formatted text, you use ~~RTFPrint~~, as follows:

```
RTFPrint 1, "Hello there.","\b \i"
```

That line will be bold and italic. Other RTF commands let you change fonts, send paragraph breaks, even indented paragraphs. You can send whatever commands you want in whatever order, but when you're finished with your document, you MUST end with an ~~RTFEnd~~ command before you close the file you are writing.

\BCreating RTF Documents: How RTF.BAS Works\b

RTF is a relatively easy-to-decode system of ASCII codes. Each begins with the backslash. For example, the ~~RTFNewPage~~ command just sends the rtf command "\page". Most of these can just get stuck in the document, and it will be acted on when encountered, like "\qc" turns centering on. Every paragraph will be centered until the "\pard" (Paragraph enD) is encountered. Font changes work the same way. You change to Font 0 with "\F0".

The text itself is stored in segments surrounded by curly-braces - the {} characters. Within those segments, you can put the font attributes, like "\b".

\BSome Interesting(?) RTF Facts\b

Measurements are in TWIPS, of which there are 1440 to the inch. The RTF routines provided here use inches and multiply by 1440 so you don't have to. Font sizes, however, are in half-points. The RTF routines use points and multiply by 2.

\BThe Future of RTF.Bas\b

As with most of the programs you see in \I*VBZ\i*, the future is up to you. What feautures would you find valuable? Possible additions range from the ability to change paper size, to color, columns, graphics, and style-sheets. Let us hear from you. We'd like to hear what programs you're writing with RTF, and how it could be improved to help you.

\BCreating RTF Documents: Commands\b

\UFilename\u

RTF.BAS:

\UCommands\u

\BCommand\b	**\BDescription\b**
RTFEnd	Write code to end RTF document
RTFFont	Write code for font change, using index in RTF font table
RTFFontSize	Write code for font size change
RTFJustifyCenter	Write code for centering text
RTFJustifyIndent	Write code for indenting paragraph
RTFJustifyNormal	Write code to turn off indenting or centering

RTFNewPage	Write code for new page
RTFNewPar	Write code for new paragraph
RTFPrint	Write text with font attributes
RTFSetFonts	Create RTF font table
RTFSetFormat	Create RTF page definition
RTFTab	Write code for tab

\BCreating RTF Documents: RTFSetFonts\b

\UPurpose\u

Creates an RTF font table.   An RTF document needs this table so that fonts can be switched with a numeric reference to a table entry.

\UContained in\u

RTF.BAS

\UDeclaration\u

```
sub RTFSetFonts (filenum, fontstring)
```

\UParameters\u

\B**Parameter\b**	\B**Description\b**
filenum	Integer - a handle to an open file
fontstring	String - the names of the fonts you will use, separated by commas

\UUsage\u

```
RTFSetFonts 1, "Times New Roman, Arial, Symbol"
```

\UComments\u

In the example above, you would now switch to Times New Roman with the command:

```
RTFFonts 1,0
```

This is because Times New Roman is the 0 item in the table, Arial is the 1 item and Symbol is the 2 item.   (The 1 in the example is the file handle of the document being created.)

RTFSetFormat and \B**RTFSetFonts\b** together are needed to initialize an RTF document. RTFEnd is needed to end a document

\BCreating RTF Documents: RTFSetFormat\b

\UPurpose\u

Writes the RTF codes that define the page

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFSetFormat (filenum, lmargin, rmargin, tmargin, bmargin)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - handle to an open file
lmargin	Single - left margin in inches
rmargin	Single - right margin in inches
tmargin	Single - top margin in inches
bmargin	Single - bottom margin in inches

\UUsage\u

```
RTFSetFormat 1, 1.5,1,5, 1,1
```

\UComments\u

\BRTFSetFormat\b and RTFSetFonts together are needed to initialize an RTF document. RTFEnd is needed to end a document.

\BCreating RTF Documents: RTFPrint\b

\UPurpose\u

Save text to disk with RTF font attributes

\UContained in\u

RTF.BAS

\UDeclaration\u

```
Sub RTFPrint (filnum,text$,attributes$)
```

\UParameters\u

\B**Parameter\b**	\B**Description\b**
filenum	Integer - a handle to an open file
text$	String - the text to be printed
Attributes$	String - desired attribute backslash commands; any or all of the below: \\b Bold \\i Italic \ul Underlined \strike StrikeThrough \v Hidden (inVisible?) \scaps Small Caps

\UUsage\u

```
RTFPrint 1, "Hello there.","\b \i"
```

\BCreating RTF Documents: RTFNewPar\b

\UPurpose\u

The RTFPrint command doesn't create an RTF carriage-return at the end.   Since you are creating a document for a word processor that will be doing linewrapping, you don't want a cr-lf at the end of each line.   When you DO need a new paragraph, use the RTFNewPar command.

\UContained in\u

RTF.BAS

\UDeclaration\u

```
     Sub RTFNewPar (filnum)
```

\UParameters\u

\B**Parameter\b**	\B**Description\b**
filenum	Integer - a handle to an open file

\UUsage\u

```
     RTFNewPar 1
```

\BCreating RTF Documents: RTFNewPage\b

\UPurpose\u

To write an RTF newpage code to an open file

\UContained in\u

RTF.Bas

\UDeclaration\u

```
     Sub RTFNewPage (filenum)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - a handle to an open file

\UUsage\u

```
     RTFNewPage 1
```

\BCreating RTF Documents: RTFFont\b

\UPurpose\u

To change typefaces, referring to the fonts by their position in the font table:

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFFont (filenum)
```

\UParameters\u

\B**Parameter Description\b**	
filenum	Integer - a handle to an open file

\UUsage\u

To switch to the first font in the font table:

```
RTFFont 1, 0
```

To switch to the second font in the font table

```
RTFFont 1, 1
```

\BCreating RTF Documents: RTFFontSize\b

\UPurpose\u

Specifies fontsize for RTF document

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFFontSize (filenum,fontsize)
```

\UParameters\u

\B**Parameter\b**	\B**Description\b**
filenum	Integer - a handle to an open file
fontsize	Single - size in points

\UUsage\u

```
RTFFontSize 1,12
```

\BCreating RTF Documents: RTFJustifyCenter\b

\UPurpose\u

Writes RTF codes for centering

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFJustifyCenter (filenum)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - a handle to an open file

\UUsage\u

```
RTFJustifyCenter 1
```

\UComments\u

This code is reversed by RTFJustifyNormal

\BCreating RTF Documents: RTFJustifyIndent\b

\UPurpose\u

Writes RTF code to indent a paragraph

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFJustifyIndent(filenum, paraIndent, firstlineIndent)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - a handle to an open file
paraIndent	Single - Amount in inches to indent paragraph
firstlineIndent	Single - Amount in inches to indent 1st line \Brelative to paragraph\b

\UUsage\u

To indent paragraph 2" and its first line an additional .25"

```
RTFJustifyIndent 1, 2, .25
```

To indent paragraph 1", but not the first line (a hanging indent)

```
RTFJustifyIndent 1, 2, -2
```

\UComments\u

This code is reversed by RTFJustifyNormal

\BCreating RTF Documents: RTFJustifyNormal\b

\UPurpose\u

Writes RTF code to turn off any indent codes activated by RTFJustifyCenter or RTFJustifyIndent

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFJustifyNormal (filenum)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - a handle to an open file

\UUsage\u

```
RTFJustifyNormal 1
```

\BCreating RTF Documents: RTFTab\b

\UPurpose\u

Writes a tab in RTF format to an open file (Sending a chr$(9) doesn't work in RTF).

\UContained in\u

RTF.Bas

\UDeclaration\u

```
Sub RTFTab (Filenum)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**
filenum	Integer - a handle to an open file

\UUsage\u

```
RTFTab 1
```

\BCreating RTF Documents: RTFEnd\b

\UPurpose\u

Write RTF codes to end RTF "session" in an open file.

\UContained in\u

RTF.Base

\UDeclaration\u

```
Sub RTFEnd (filenum)
```

\UParameters\u

\B**Parameter\b**	**\BDescription\b**

filenum Integer - a handle to an open file

\UUsage\u

This routine MUST end your document. Run it right before you close your file:

```
RTFEnd 1
Close #1
```

\BUsing RTF.BAS: The Resume Wizard\b

Although the Resume Wizard doesn't provide as much power or flexibity as a commercial resume generator, it serves as a good sample project for the RTF commands in RTF.BAS: it uses two different fonts, three different font sizes, bold, italic, hidden and smallcaps text, as well as hanging indents. The resumes Resume Wizard creates are ready to be retrieved into Word, printed, and sent to your prospective employer, but \I*VBZ\i* makes no warranty, expressed or implied, as to the quality of its output, so you'd better check it before you mail it! The Wizard is also a good demonstration of the power of VB: the heart of the program is a routine that, uncommented, would be only a page long.

Using the Resume Wizard
How the Resume Wizard Works

\BUsing RTF.BAS: Using the Resume Wizard\b

The Wizard ships with a database already filled with some sample records. If you want, you can press Generate right away. You will be asked for a file name. Create something with an RTF extension, and then call it up with Word for Windows.

To create your own resume, fill in your name and address, then erase the item records until you are told you can start adding.

The "category" field refers to group names like "Education" or "Work Experience".

The "Title" is where you put job titles, like "Director" or "Supervisor". This is also where you put the text for items that aren't jobs. For example, this would be where you put the name of a school you attended. If the category was "References", you could use the "Title" field for "Available upon request"; if the category was "Hobbies", you could use the "Title" field for "Working Hard".

The "Years" field is a textbox so you can type something like "1989-1990".

The other fields are self-explanatory. To change the order of the records, you can change the number in the "Item" field. This won't actually change the record order unless you press the Refresh Record Order button.

Once you've created your records, press Generate, create a file with an RTF extension, and you've got a file you can retrieve into Word, or any other word-processor that can take RTF.

\BUsing RTF.BAS: How the Resume Wizard Works\b

The information is stored in an Access database file called Resume.MDB. It has two tables. One for the Name and Address, and the other for all the fields of the resume entries. Examine the properties of the data controls and you will see the database names and the recordsources (or tablenames). The fields are linked to these datacontrols with their Datasource (the name of the datacontrol) and the Datafield properties.

Once the database is populated, and the user presses Generate, the form asks for the name of an RTF file, which it opens for output. Then the two RTF header commands are run, RTFSetFonts and RTFSetFormat. The \B**RTFSetFormat\b** commands gets its margin settings from the textboxes on the form. The name and address are printed (the address must be printed a line at a time, so this requires some parsing), and finally, the database is cycled through. This is done with the basic structure:

```
Do Until data1.recordset.EOF
   ...
   data1.recordset.MoveNext
Loop
```

Each record is checked to see if it is the first of a new catagory, in which case the category gets printed (otherwise, it does not.) The information can be retrieved from the records in two ways: through the databound textboxes, or with a database command, e.g., `data1.recordset("Employer")`. This is a shortcut for `data1.recordset.fields("Employer")`.

As to how the information is printed, I refer you to the section on RTF.BAS. There is one formatting trick worth noting here, however: Those paragraphs that contain a catagory are indented paragraphs, with a firstline "outdent" so the category is to the left of the paragraph. Then a tab is sent before the rest of the paragraph. This tabs all the way out to the paragraph margin, ignoring any tabstops on the way.

Feel free to modify this program. Possible additions could include better font choices, the storage of the chosen attributes, multiple resumes, and multiple resume styles. Let us know if you come up with anything, or want us to modify Resume Wizard in any way.

Q+E MULTILINK Sug. Ret. Price: $399
by Pioneer Software
5540 Centerview Drive
Raleigh, NC 27606
Pho: (919) 859-2220
Fax: (919) 859-9334

The potential of Visual Basic as a database development tool, especially in the area of client/server development, has been known for quite some time. It is this potential that provided the impetus for the creation of Q+E MultiLink, an add-on that enables developers to utilize Visual Basic's flexibility and fast development cycle to create client/server applications.

\BData Basics: Database Programming in the Visual Basic Environment\b

How does connecting multiple high-end RDBMS's within your Visual Basic application sound to you? Say, for example, Oracle with Sybase, however unlikely the example may be, or DB2 with Informix, another unlikely pairing. With Q+E Multilink you can do just that. And while these examples are very doubtful, can you picture an organization that may have data stored simultaneously in Oracle, Dbase and Excel? You can access all of those formats, and DBMS's in ONE Visual Basic application, by using Multilink.

Q+E Multilink will let you create applications that are able to access the following databases and formats:

ASCII TEXT
Btrieve
dBase compatible files
Excel Worksheet Files
HP Allbase, HP Image/SQL
IBM DB2
IBM OS/2 Datamanager, AS/400 and SQL/DS
Informix
Ingres
Novell Netware SQL

Sybase and Microsoft SQL Server products
SQL Base
Terradata
XDB

Oracle
Paradox
Progress
Tandem Non-Stop SQL

This portability is made even more attractive because you only write an application once.and then just change one property - the database name - to connect your app to a different database. Furthermore,the entire application is multi-user capable. These features make portability painless.

MultiLink's main purpose, aside from enabling a developer to write for a variety of RDBMS applications, is to provide the user with easy access to information. This is primarily accomplished by giving you the tools to create relatively convenient and fast implementations of data queries. It is important to note this fact, since different products have different areas of strength. MultiLink very much excels at creating and executing queries. Pioneer's corporate mission statement probably reads something like: "We will not rest until we can connect to every database format and provide fast access to their respective data."

\BData Basics: Grids, Text Boxes, Scroll Bars and VB 3.0, The Hidden Contender\b

Of course, there are some trade-offs involved in implementing a tool with this kind of portability. The biggest one is speed. Pioneer's tools have not been known as speed demons. This reputation came about due to some difficulties experienced by earlier releases of other Q+E programs. More appropriately,the statement regarding MultiLink's speed is somewhat relative, depending on what you are used to working with. Just for perspective, the Access engine included with VB 3.0, which is most likely the closest direct competitor, is noticeably slower. And ODBC, in its present incarnation, appears slower still. I do not profess to have conducted any benchmark tests. These are just simple observations made with the naked eye. As far as the individual features of each program are concerned, have a look at the attached comparison of VB 3.0 and MultiLink.

Money is always a prime consideration in the evaluation of tools for your production efforts. I know that a few of you will carefully weigh the alternatives between using VB 3.0 to its fullest potential and spending the necessary funds for yet another add-on. That being the case, please keep in mind that VB 3.0 can access eight databases, while MultiLink can reach twenty. Aside from that obvious difference, MultiLink also provides you with twelve data-aware custom controls. These include some that are not available from Microsoft yet, such as a bound query grid. How's that for productivity? Figure 1 shows the Visual Basic toolbox with MultiLink installed.

While we are on the subject of query grids, someone may point out that the VISDATA sample application shipping with VB 3.0 shows the utilization of the VB grid control in a database setting. But again even Microsoft's own grid is not databound. This obvious opportunity to provide such a feature is being seized by a number of other vendors. As you are reading this article, there is an interesting development in the works by Sheridan Software, which captured Bill Gates' attention enough for him to play with a demo for over thirty minutes at the last Comdex. Okay, okay, enough gossip for now.

Back to business. Some of the controls that come with MultiLink are, at first glance, very similar to the ones shipping with VB. However, you will find Q+E's to be more useful overall because of the additional database properties and built-in functionality. This is especially noticeablewhen dealing with back-end systems that can be reached only through MultiLink, and not through ODBC.

Among the controls shipping with this release of MultiLink are interesting offers, such as the data-aware radio button and scroll bars. It seems that one feature of the scroll bar control arose out of necessity - because of a possible shortcoming in the Q+E query grid control. It appears that the query grid cannot be used to enter data. It is read - only. As a result, Q+E creates a "quasi" grid for data entry by creating a number of textbox arrays and, then, using the scroll bar to move records through those text boxes. While giving the appearance of a grid, this achieves an interesting effect, since most of us are used to seeing scrollbars directly attached to a list box or table. Figure 2 illustrates this idea.

The actual production of this "quasi" grid is quite simple; and of course, you can add or delete elements just by adding or deleting elements of the array. The lack of a write/update property in the query grid control, while unfortunate, is more than outweighed by MultiLink's other positive features. Q+E is aware of the need for an editable query grid, and promises such an improvement for release 2.0 of MultiLink in December 1993.

MultiLink Controls, such as check boxes, list boxes, combo boxes, text boxes, command buttons and the picture control, are different from, and sometimes quite improved over, their VB 3.0 counter parts. The properties, events and functions that are attributed to them by Q+E are very unique and easy to use.

\BData-Basics: Two-Field Unique Indexes and Dynamic Queries\b

Among these properties and functions were a couple of particularly interesting refinements. While these small provisions are not especially earth shaking, I found them to be thoughtful and quite nice to have available.

The first one is called the pKey property and is used to set primary indexes. The nice part I'm talking about is the fact that the pKey property allows the programmer to specify more than one field as unique (for indexing purposes). This means you are no longer chained to a field holding a record identification number, unless you like that sort of thing. For example, you could specify the combination of last name and social-security number to be the unique index. Having this extra little bit of convenience is nice. It can be applied to the Q+E check box, combo box, list box, radio button group, text box and query grid.

The other refinement I enjoyed is part of what Pioneer calls the pWhere property. This property is one of the workhorses and also one of the connect and query controls. In the query control, the pWhere property contains the conditions that a record must meet in order to be retrieved by a query. These conditions can be sort orders or groups. You will most often use this property to find specific matching values between a query and the actual data in a table.

To backtrack just a little, when retrieving data through queries, a QBE (Query by Example) facility is especially useful. When creating a QBE form, a developer will most often have the user input or pick the search criteria, such as a "salary field" of $30,000 in a text box (i.e. the user types 30,000). The next step is to pass the contents of the "salary field" text box to a data table in the form of a query. This will produce any matching records. Ordinarily, the process of capturing the input and passing it to the query and table requires a few lines of code. Here is where the pWhere property can help. In designing the query, a developer can specify the contents of the example text box (salary field) with a "wildcard" character. The official term for it is a hook. By using a hook, you can set the pWhere property dynamically. The statement would look something like this:

```
Salary > ?Salary Field
```

The question mark represents the hook. It will see to it that the contents of the "salary field" are used in the query, whatever they may be. The query as written here looks for salaries exceeding the amount specified in the "salary field" (larger than 30,000). Importantly, the technique of using a hook only works with MultiLink field controls, not with standard VB field controls or those of other vendors.

\BData-Basics: Other Distinguishing Characteristics\b

Aside from the custom controls, properties and functions, Multilink distinguishes itself in a number of other aspects. For example, noteworthy is the idea that when using VB 3.0, a developer is allowed only one active query per data control. This results in multiple data connection, or data access controls implemented in order to accommodate multiple active queries. Depending on the situation, this can lead to a drain on your server. MultiLink's solution is to allow multiple queries per each single data connection, making your application more efficient.

Another noteworthy point is the way transaction processing is being handled. While VB 3.0's transaction processing is global, affecting all of your data controls on all forms, Multilink's transactions can be tied to specific data connections or queries. This gives you a much finer degree of control, especially in the development of mission-critical applications.

Transaction processing is often found in corporate environments,where large tables of data are the order of the day. This climate will benefit from another difference between VB 3.0 and Q+E MultiLink. When using Q+E's product to connect to SQL systems, it is not necessary for all tables to have unique indexes. Unfortunately, that \I*is*\i a requirement of VB 3.0. For a lot of people that may not be a problem, unless you happen to be the corporate database administrator who keeps getting requests from his VB users to reset the indexes. Can you picture that poor fellow?

\BData-Basics: Helpful Utilities\b

To ease your development cycle, Q+E provides two utilities that aid in the design of client/server (or other) applications.

Included with the distribution disks, is a Database Manager that allows the developer to create, edit, modify and delete files in any of the twenty supported formats from within MultiLink.   This flexibility is actually another difference between VB3.0 and MultiLink, since the database manager shipping with VB 3.0 can only access the Foxpro, Paradox, dBase and Access formats.   It does not support Oracle or Sybase, the formats that need to be reached via ODBC.   The Database Manager is shown in figure 3.

Those of you familiar with other database management utilities will feel right at home with the functionality of this program.   Actually, anyone who ever had to create or maintain database files will feel right at home.   It's straight forward;except that, in the case of SQL tables, you will need to log on to the server first.   However, once you are logged on, it's a breeze to create or modify tables.   All field definitions are just a mouse click away, as shown in figure 4.

The second utility program, and in some cases the more important one, is the Query Builder.   It is the same clever program found in other Q+E products.   It's purpose is to lead you through the design of standard SQL queries - even if you don't know the language.   Needless to say, this utility can come in handy.   The Query Builder is shown in figure 5.

The Query Builder becomes available at design time through the VB property settings' box .   After placing a Q+E connect control on the form in question and connecting to a database, it is possible to specify the tables, fields and query expressions with this tool.   Mind you, it is only available \B**after you connect\b**.   So: place a connect-control and hook-up to your tables. Then, place a query control on the form.   In its property settings' box, click the dialog pop-up button ("...") under the pTable, pWhere and pExpr entries.   Pushing this button calls the query builder.   Believe me, it is done a lot faster than described in writing.

You now have the ability to create, edit or view SQL queries.   For example, you may want to specify a number of fields.   Nothing could be simpler.   Figure 6 shows what happens when you click the "Fields" icon.

The query builder lets you sort records (figure 7), group records, join tables and even check the SQL expressions you are creating (figure 8).

SQL Statements can be cut or copied to the Windows clipboard. You can also perform find-and-replace operations.

\BData-Basics: Final Impressions\b

Pioneer has created a winner with this product. Consider how Microsoft Access has been compared to Powerbuilder, the premier client/server database development program, and then look at some of the user comments regarding the Access engine compared to MultiLink. One could almost say that the combination of VB and MultiLink are in competition with a software product that costs over $3000. This is good cause for enthusiasm.

Think about it. How many times do you wonder which development tools in use today that will be around tomorrow? For example, nobody would want to be in the shoes of those programmers who had spent time and energy learning the Paradox Application Language, only to be faced with a completely different syntax in ObjectPal, the Windows language of Paradox. I can't imagine such a debacle facing a VB developer who wants to standardize around Q+E MultiLink. As I've said in the beginning, the manufacturer will just continue to make this program as adaptable as possible. So if you are looking for a company and program around which you can standardize, you can't get much better than this one.

In the end, it all boils down to productivity. If a tool makes you more productive and hence operate more profitably, use it. If it doesn't do that, then don't use it.

\BData-Basics: User Comments\b

\BMike Oden\b

Mike Oden of Oden Industries, located in Pasadena, California, has used Q+E products for some time. He included MultiLink in a project that was actually showcased at the last dB Expo. This showcase application is a database used by a medical corporation specializing in pathological examinations. The program helps this client with several important aspects of day-to-day logistics, including the management of individual patient records, reports and the billing of services rendered. Mike's application distinguishes itself by being able to incorporate photographs of actual biopsies. In addition, the examining doctor is able to record his comments and observations right alongside the picture to be stored with the patient's information. This greatly improves efficiency and helps the pathologist to complete more work in the same amount of time, thus operating more profitably.

As part of a very competitive local development community, Mike must choose his tools carefully. When asked to comment about MultiLink, he stated, "The program has several almost equally important features. First of all, you can connect different database back-ends in one application. For example, the showcase program utilizes Microsoft SQL and dBase in twenty different files and tables. Secondly, in my own work, MultiLink was noticeably faster than ODBC and the Access engine, so much so that there was no question at all which program I would use. Then there are the actual features of the controls that come with MultiLink, such as the databound list box. With MultiLink, I don't have to write an AddItem loop that puts the results of queries into a list box, as I would have to in VB. Lastly, the company has been around a while and has been working with this technology. Consequently, there are a lot fewer bugs than a newer vendor might experience."

\BJim Thompson\b

Jim Thomson, an independent consultant presently contracting with the Fisher-Rosemount division of Emerson Electric, is using Q+E Multilink in the development of a sales and marketing decision-support system. The application is a Microsoft Windows client to a Microsoft SQL Server under OS/2.

When asked about his experience with the product,Jim said, "I initially selected MultiLink for its ability to connect to a PROGRESS back-end database. However, PROGRESS was ultimately deemed to be too slow, particularly with set queries. I called Q+E on the PROGRESS performance issues, and talked to the individual who wrote the PROGRESS interface. He explained that the performance issues were due to PROGRESS' host language interface, and gave me the name of a representative at PROGRESS Software. I contacted this individual who confirmed that the host language interface was indeed slow and would remain so until the next release of PROGRESS (version 7). Throughout this process, I found the Q+E personnel courteous, responsive, and knowledgeable." Because of these difficulties with PROGRESS, we changed our server database to Microsoft SQL Server, without the need to change any of the code we had developed so far, with the exception of a single connect statement."

With regard to the obvious competition between Q+E and VB 3.0, Jim stated, "Microsoft's inclusion of the Access 1.1 engine in VB 3.0 has brought into question the need for MultiLink. However, MultiLink's inclusion of Query-by-Example capability and bound grid control has proven to be a great tool for our development efforts. Also, we have found MultiLink to be more stable than Microsoft's ODBC drivers." Interestingly enough, Jim also mentioned that he found Q+E's tech support personnel more accessible and knowledgeable than Microsoft's.

\BData-Basics: MultiLink vs. VB 3.0\b

Topic	Visual Basic 3.0	Q+E MultiLink/VB
Databases supported	MS Access, FoxPro, dBase, Paradox (3&3.5), Btrieve, SQL Server, Oracle(SQL Server and Oracle are only available in the Professional Edition)	Oracle, SQL Server, dBase, INGRES,Paradox (4.0), Sybase, AS/400 (SQL400)*, Btrieve, OS/2 DBM, DB2*, INFORMIX, Netware SQL, PROGRESS, SQLBase, XDB, SQL/DS*, Tandem NonStop SQL*, Teradata, Excel .XLS files, Text files, HP ALLBASE/SQL* and HP IMAGE/SQL and gateways, IBM DDC/2, Micro Decisionware and Sybase Net-Gateway* requires Gateway

Database Control Types	Text, Check Box, Picture, Data Label, Masked Edit	Text, Check Box, Picture, Query, List Box, Combo Box, Radio Button, Query Grid, Command Buttons, Scroll Bars and Connect Controls
Control Arrays	No - you can only display one record at a time on a form using the database controls.	Yes - you can display any number of records on a form at one time without writing code. Fully supports control arrays.
DB Command Button Controls to perform database operations directly	No	Yes - any database operation can be assigned to a button without writing code.
DB Scroll Bar Controls	No - must use Data control and arrow buttons on Data control; no stand alone scroll bar.	Yes - scroll bar can be used to move through records in a database without writing code.
Database Control Features	Can link Controls to database fields.	Can link Controls to database fields, and can format numbers as well as date values using format strings without writing code.
Query by Example	No	Yes - allows end user of application to use QBE to query for specific records.
Query Builder Utility	No	Yes - allows the developer to build queries and view/edit their SQL statements without knowing SQL language.
Picture Controls	Picture must be stored in databaseOnly BMP format supported.	Pictures can be stored in the database itself, or the name of a specific field containing the pictures can be stored in the database.BMP and MetaFile formats are supported.
Cross Form Control Support	No	Yes - all controls support cross-form referencing, allowing developers to split records and share connections and data across forms
Combo and List Box Controls filled via bound database fields	No	Yes - the combo and list boxes can be filled with directly bound database fields.
Database Manger Utility	Yes - however it does not support SQL databases.	Yes - all database formats fully supported

Transaction Processing Support	Yes - however transactions are only supported for SQL databases which themselves need to support transactions.	Yes - transactions supported for all formats including dBase, Btrieve and all SQL databases.
Transaction Mechanism	Transactions are global to all active databases in an application.	Transactions based on connection level, providing more control for client/server applications.

Provided by the Product Management of Q+E Software

While we have a lot already in the works, the content of \I*VBZ\i* is largely up to you, the reader, to dictate. If you are having a problem, send it in. If you want more of a particular type of article - beginner, advanced, more DLLs, more VB code - let us know and we'll do our best to accommodate your needs and wants.   Here's our list thus far:

\UVB Developer's Utilities\u
Button Bitmap Builder
Palette Builder (for PicClip among other things)
Stub Replacement
Object Manager
Code Generators/Wizards
. . . and much more!

\UVisual Basic Techniques\u
Calling DLL functions
Standardizing Your VisualBasic Interfaces
Advanced Printing
Metafile Creation
An Improved FindWindow Function
Waiting for other Apps to Execute
. . . and much more!

\UDynamic Link Libraries\u
SendKeys function for DOS Applications
Additional VBZUTIL functions
. . . and much more!

\UCustom Controls\u
Generic Subclassing Control
Clipboard Viewer Control
Huge Scrollbars
. . . and many other specialized controls!

\UImproved Help Files\u
Binary Sample Extraction

\BThe rest is up to you!\b   Be sure to let us know which of these things are of greatest interest to you so that we can bump them to the top of the list to complete.

\IVB Developer's Utilities\i

~~Dropping Leftover DLLs (VBZ02)~~
~~Pop Up Color Selection (VBZ02)~~
~~Printing Browse Sequences in WinHelp (VBZ03)~~
~~Simplified Formatted Printing (VBZ03)~~

\IVisual Basic Techniques\i

~~Accessing Private INI Files (VBZ03)~~
~~Accessing the Common Color Dialog (VBZ02)~~
~~Aligning Controls (VBZ03)~~
~~Creating a Drag 'n Drop Client Application (VBZ01)~~
~~Creating Modeless Dialog in VB (VBZ02)~~
~~Creating Windows 3.1 Screen Savers (VBZ01)~~
~~Screen Saver Wizard (VBZ03)~~

\IDynamic Link Libraries\i

~~Aldus Format Metafiles (VBZ01)~~
~~A PLAY Command for Visual Basic (VBZ01)~~
~~Apps That Tile and Cascade (VBZ02)~~
~~Creating a Drag 'n Drop Server Application (VBZ01)~~
~~Musical MsgBox and Beep Commands (VBZ01)~~
~~Status Bar Help on Controls and Cursors (VBZ01)~~
~~System Level Hotkeys (VBZ01)~~
~~The QuickBasic Function Library (VBZ02)~~

\IReviews\i

~~Doc-To-Help (VBZ02)~~
~~3-D Widgets (VBZ02)~~

About *VBZ*

VBZ is a completely electronic journal for Visual Basic programmers. Each issue is in the form of a help file, such as the one you are reading, with lots of techniques, sample code, DLLs, custom controls and reviews. An issue of *VBZ* will come out every two months unless there is great demand for greater frequency at the expense of content in each issue. We think it's better to wait and get some substantial stuff into the journal rather than just try to get it out every month as some others do.

Subscribing to *VBZ*
License Information
Using *VBZ* DLLs
Writing for *VBZ*

Subscribing to *VBZ*

If you are not currently a subscriber, you should be. We accept MasterCard or Visa, or send check, money order or purchase order (payable to User Friendly, Inc.) for $69 + $4 shipping and handling for the first subscription, $49 + $4 shipping and handling for each additional subscription, to:

User Friendly, Inc.
1718 M Street, N.W.
Suite 291
Washington, DC 20036
(202) 387-1949
(202) 785-3607 FAX
CIS: 71652,2657

Please specify whether you would like to receive *VBZ* via email (in which case please include your Compuserve account number), or US Mail (in which case please include your preferred media size).

VBZ may also be registered on CompuServe's Shareware Registration Forum (GO SWREG). *VBZ's* Registration ID Number is 1005. $73.00 will be billed to your CompuServe account, and we will be notified to email you your subscription.

License Information

Just like a paper journal, *VBZ* should not be in two places at once. Therefore, the license information is quite simple. If you pass the journal around, you must do it in its entirety, erasing it from your own hard disk. In this way, many can read a single issue of *VBZ* but only one at a time.

As a subscriber you have full rights to distribute the DLLs in their binary form (no source code) with your programs. You may not document their internal use, however. If more than one developer will be actively using the DLLs than each developer needs a subscription to *VBZ*.

Disclaimer

The techniques and software presented in this journal are offered "as is" with no warranty expressed or implied as to its suitability to the task at hand or reliability under diverse conditions. The primary purpose of this journal is education. You use the software and techniques in this journal at your own risk.

Using *VBZ* DLLs

Each DLL that comes with an issue of *VBZ* contains a version resource so that you can use the Windows version verification procedures when installing one of our DLLs over an existing one as part of a setup procedure. Please make every effort not to overwrite a newer copy with an older one that another developer might have installed on the users system.

Whenever a DLL is fixed, only the minor version of the DLL will go up. Only when substantial functionality has been added to the library will the major version increase.

Each DLL also has a module (.BAS) of the same file name with the necessary declarations and constants so that you can simply add this file to your project and start programming. Let us know if there is anything else we can do to make the use of our DLLs any easier.

Writing for *VBZ*

If you are interested in getting your name in print or simply looking for a way to share your findings, we are interested in hearing from you. Because of the format, there is no real limit as to the number or size of submissions. Even if it's just a little technique you have discovered, I'm sure readers would be interested in hearing about it.

In addition to the notoriety, one benefit of using *VBZ* to get your idea out is that you have an organized way to fix or improve the idea later on. We will do our best to work with you to get your idea working in the first place but we will also make sure that corrections go out to all our subscribers. This kind of dynamic approach is one of the unique benefits of electronic publishing.

If you wish to make a submission of some sort, please try to make it problem-solution oriented versus "general discussion." We will be happy to look at anything but the journal is targeted at those with specific problems and needs.

Please send your ideas, either by mail or email. We look forward to getting your name in print!

VBZ Submission
User Friendly, Inc.
1718 M Street, N.W.
Washington, DC. 20036
(202) 387-1949
(202) 785-3607 FAX
CIS: 76702,1605

\I*VBZ\i* is a completely electronic journal for Visual Basic programmers. Each issue is in the form of a help file, such as the one you are reading, with lots of techniques, sample code, DLLs, custom controls and reviews. An issue of \I*VBZ\i* will come out every two months unless there is great demand for greater frequency at the expense of content in each issue. We think it's better to wait and get some substantial stuff into the journal rather than just try to get it out every month as some others do.

\BSubscribing to \I*VBZ*\i\b

If you are not currently a subscriber, you should be. We accept MasterCard or Visa, or send check, money order or purchase order (payable to User Friendly, Inc.) for $69 + $4 shipping and handling for the first subscription, $49 + $4 shipping and handling for each additional subscription, to:

User Friendly, Inc.
1718 M Street, N.W.
Suite 291
Washington, DC 20036
(202) 387-1949
(202) 785 - 3607 FAX
CIS: 71652,2657

Please specify whether you would like to receive \I*VBZ\i* via email (in which case please include your Compuserve account number), or US Mail (in which case please include your preferred media size).

\I*VBZ\i* may also be registered on CompuServe's Shareware Registration Forum (GO SWREG). \I*VBZ's\i* Registration ID Number is 1005. $74.00 will be billed to your CompuServe account, and we will be notified to email you your subscription.

\BLicense Information\b

Just like a paper journal, \I*VBZ\i* should not be in two places at once. Therefore, the license information is quite simple. If you pass the journal around, you must do it in its entirety, erasing it from your own hard disk. In this way, many can read a single issue of \I*VBZ\i* but only one at a time.

As a subscriber you have full rights to distribute the DLLs in their binary form (no source code) with your programs. You may not document their internal use, however. If more than one developer will be actively using the DLLs than each developer needs a subscription to \I*VBZ.*\i

\BDisclaimer\b

The techniques and software presented in this journal are offered "as is" with no warranty expressed or implied as to its suitability to the task at hand or reliability under diverse conditions. The primary purpose of this journal is education. You use the software and techniques in this journal \Iat your own risk.\i

\BUsing \IVBZ\i DLLs\b

Each DLL that comes with an issue of \I*VBZ\i* contains a version resource so that you can use the Windows version verification procedures when installing one of our DLLs over an existing one as part of a setup procedure. Please make every effort not to overwrite a newer copy with an older one that another developer might have installed on the users system.

Whenever a DLL is fixed, only the minor version of the DLL will go up. Only when substantial functionality has been added to the library will the major version increase.

Each DLL also has a module (.BAS) of the same file name with the necessary declarations and constants so that you can simply add this file to your project and start programming. Let us know if there is anything else we can do to make the use of our DLLs any easier.

\BWriting for \IVBZ\i\b

If you are interested in getting your name in print or simply looking for a way to share your findings, we are interested in hearing from you. Because of the format, there is no real limit as to the number or size of submissions. Even if it's just a little technique you have discovered, I'm sure readers would be interested in hearing about it.

In addition to the notoriety, one benefit of using \I*VBZ\i* to get your idea out is that you have an organized way to fix or improve the idea later on. We will do our best to work with you to get your idea working in the first place but we will also make sure that corrections go out to all our subscribers. This kind of dynamic approach is one of the unique benefits of electronic publishing.

If you wish to make a submission of some sort, please try to make it problem-solution oriented versus "general discussion." We will be happy to look at anything but the journal is targeted at those with specific problems and needs.

Please send your ideas, either by mail or email. We look forward to getting your name in print!

\IVBZ\i Submission
User Friendly, Inc.
1718 M Street, N.W.
Washington, DC. 20036
(202) 387-1949
(202) 785-3607 FAX
CIS: 76702,1605

The *VBZ* Utility Library

Every so often, there are functions for which it becomes necessary to throw in a DLL. Many of you have requested that we stop creating one-function DLLs and this makes sense. The result of this request is VBZUTIL.DLL which is designed to contain all of these little utility functions that don't quite belong anywhere else. Don't mistake these for trivial functions, because if they were, they wouldn't have to be in a DLL. In fact, it is in this library that some of the most important and creative functions will be found.

What goes into this DLL is guided both by our discoveries and needs and requests from you. All that we ask is that you don't request standard functions that can be found in any commercial add-on package. In other words, don't look for a routine to sort integer arrays any time soon. Other than that, the sky's the limit. You never know what's possible unless you ask and we can never anticipate all of our subscribers' needs. The functions in this library will often be used in context in the sample programs that come with *VBZ*, but there will be no samples devoted to these functions. If you would like this to be otherwise, let us know.

The function reference for VBZUTIL.DLL will always be updated instead of being fragmented over every issue. This is the one example where you can get the latest issue and know that you are not missing anything, which is not the case for the DLLs and samples in the features. Functions new to this issue are marked with ***.**

VBZUTIL Function Reference

The declarations can be found in VBZUTIL.BAS.

CtlName
CtlParentClass
CtlTypeOf
CtlUBound & CtlLBound
GetBValue
GetGValue
GetHInstance
GetRValue
HWnd2Ctl
HIWORD & LOWORD
LPSTR2Str
LP2Str
MakeCtlArray*
MAKELONG
ReCreateControlhWnd
Str2Ctl
USHORT

The *VBZ* Utility Library: CtlName

Purpose

Returns the name of a control as a string.

Contained in

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
    Declare Function CtlName$ Lib "VBZUTIL.DLL" (C As Control)
```

Parameters

Parameter	**Description**
C	Control variable - Control for which name is needed

Returns

The name of the control as a string

Usage

```
    tbName$ = CtlName$ (Text1)
```

Comments

This function exists because of the unavailability of the .Name property of a control at runtime.   You use it in much the same way you would a control, however.

The *VBZ* Utility Library: CtlUBound & CtlLBound

Purpose

Returns the upper and lower bounds of a control array.

Contained in

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declarations

```
Declare Function CtlUBound Lib "VBZUTIL.DLL" (C As Control)
Declare Function CtlLBound Lib "VBZUTIL.DLL" (C As Control)
```

Parameters

Parameter	**Description**
C	Control variable - any element of a control array

Returns

Integer - the highest or lowest array index of control array

Usage

```
UpperBound% = CtlUBound (MyControl (0))
LowerBound% = CtlLBound (MyControl (7))
```

Comments

Don't pass a control to either of these functions unless you know it to be a member of a control array. *Don't* pass the control array with an open paren. You must pass a valid control to the function but it can be any element in the array.

The *VBZ* Utility Library: GetHInstance

Purpose

Returns the instance handle of your current application.

Declaration

```
Declare Function GetHInstance Lib "VBZUTIL.DLL" ()
```

Returns

Integer - the instance handle of the current application

Usage

```
myhInstance% = GetHInstance()
```

Comments

This value is necessary for a number of Windows API functions, so it is provided here.

The *VBZ* Utility Library: HWnd2Ctl

Purpose

Returns the element into the VB2 .Controls() collection of a form, given the control's hWnd.

Contained in

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function HWnd2Ctl% Lib "VBZUTIL.DLL" (ByVal hWnd%)
```

Parameters

Parameter	**Description**
hWnd%	Integer - hWnd of a control

Returns

Integer - the index of the control array to which a given control belongs

Usage

```
Dim C As Control
'you got the Wnd% from someplace, like MWATCH
El = HWnd2Ctl% (Wnd%)
If El > -1 then Set C = Me.Controls(El)
```

Comments

At this point, you may use C as if it were a normal control and access all of its properties directly. This function is very useful in applications when you have an hWnd but you need access to the underlying properties of the control.

The *VBZ* Utility Library: HIWORD & LOWORD

Purpose

Returns the HiWord or LoWord of a LONG integer.

Contained in

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declarations

```
Declare Function HIWORD Lib "VBZUTIL.DLL" (ByVal LongVal&)
Declare Function LOWORD Lib "VBZUTIL.DLL" (ByVal LongVal&)
```

Parameters

Parameter	**Description**
LongVal&	Long integer - the integer for which you want low or high word

Returns

Integer - the HiWord or LoWord

Usage

```
HW% = HIWORD (MyLong&)
LW% = LOWORD (MyLong&)
```

Comments

Often a DLL function will return a long integer which contains two short integers. These functions will help you to parse out those values.

The *VBZ* Utility Library: LPSTR2Str

Purpose

Creates a Visual Basic string from a C null terminated string (LPSTR).

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declarations

```
    Declare Function LPSTR2Str$ Lib "VBZUTIL.DLL" (ByVal LPSTR&)
```

Parameters

Parameter	**Description**
Lpstr&	Long integer - pointer to a string

Returns

String - VB string created from C-type null-terminated string

Usage

```
    MyString$ = LPSTR2Str$ (lpstr&)
```

Comments

Sometimes a DLL function will return a C language string and you will want to get this into a VB string.

The *VBZ* Utility Library: LP2Str

Purpose

Creates a VB language string from any block of memory.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function LP2Str$ Lib "VBZUTIL.DLL" (lp As Any, ByVal
  nBytes)
```

Parameters

Parameter	**Description**
lp	Variant - pointer to a memory location
nBytes	Integer - number of bytes to retrieve into string

Returns

String - VB string created from a number of bytes starting at a given memory location

Usage

```
MyString$ = LP2Str$ (ByVal lp, 283)
MKI$ = LP2Str$ (MyInt, 2)
```

The *VBZ* Utility Library: MAKELONG

Purpose

Create a LONG integer from two short integers.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function MAKELONG Lib "VBZUTIL.DLL" (ByVal loword%,
  ByVal hiword%) As Long
```

Parameters

Parameter	**Description**
loword%	Integer - Short integer to become the low word of a long
hiword%	Integer - Short integer to become the hi word of a long

Returns

Long integer - The long made from two short integers

Usage

```
MyLong& = MAKELONG (Var1%, Var2%)
```

Comments

Often a DLL function requires a LONG integer parameter which is really two short integers. This function will allow you to call those functions. It is identical to the MAKELONG macro which is listed in the SDK documentation, but that is unavailable in a Windows DLL.

The *VBZ* Utility Library: ReCreateControlhWnd

Purpose

Recreates the hWnd associated with a particular control.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Sub ReCreateControlhWnd Lib "VBZUTIL.DLL" (C As
   Control)
```

Parameters

Parameter	**Description**
C	Control variable - Control for which you want to recreate an hWnd

Usage

This sample changes a single-column, single-select list box to multi-column, multi-select. While this is no longer necessary with a standard list in VB2 (though File Lists still can't be multi-column for some reason), it demonstrates the use of the DLL.

```
OldLong& = GetWindowLong&(File1.hWnd, GWL_STYLE)
OldLong& = OldLong& Or LBS_EXTENDEDSEL Or LBS_MULTICOLUMN
SetWindowLong File1.hWnd, GWL_STYLE, OldLong&
RecreateControlHwnd File1
File1.Refresh
```

You can also look at the D&DTEST1.MAK sample that came with VBZ01.

Comments

This function is useful when you need to change a pre-hwnd style of a control dynamically or when you need to gain access to a style that has not been exposed by VB, such as justification in a text box. This is an advanced function. Use it with care!

The *VBZ* Utility Library: Str2Ctl

Purpose

Returns the element into the VB2 .Controls() collection of a form, given the control name as a string.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function Str2Ctl% Lib "VBZUTIL.DLL" (Frm As Form,
   ByVal CtlName$)
```

Parameters

Parameter	**Description**
Frm	Form variable - the form with the control array
CtlName$	String - the name of the control array

Returns

Integer - the element within the Controls collection that is a given control

Usage

```
Dim C As Control
CtlName$ = "Text1"
El = Str2Ctl% (Me, CtlName$)
If El > -1 then Set C = Me.Controls(El)
```

Comments

At this point, you may use C as if it were a normal control and access all of its properties directly.   This function is very useful in database applications when you want to be able to relate control names to corresponding column names in a database table.

The *VBZ* Utility Library: USHORT

Purpose

Returns the unsigned version of an integer into a long integer.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function USHORT& Lib "VBZUTIL.DLL" (ByVal Word)
```

Parameters

Parameter	**Description**
Word	Integer - an unsigned integer you wish to convert to a long integer

Returns

Long integer - A long integer created from an unsigned integer

Usage

```
Unsigned& = USHORT& (MyWord%)
```

Comments

VB lacks the unsigned integer data type (often called a WORD). This function will allow you to calculate the actual value from an integer, even one which has moved into negative numbers.

The *VBZ* Utility Library: GetRValue

Purpose

Returns the amount of Red (0-255) in a color value.

Declaration

```
Declare Function GetRValue Lib "VBZUTIL.DLL" Alias
   "GetRedValue" (ByVal RGBVal&)
```

Parameters

Parameter	**Description**
RGBVal&	Long integer - a VB color value

Returns

Integer - the amount of red (0-255) in a given VB color value

Usage

```
RValue% = GetRValue (ClrValue&)
```

Comments

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out. Therefore we are providing you with GetRValue, GetGValue and GetBValue.

The *VBZ* Utility Library: GetGValue

Purpose

Returns the amount of Green (0-255) in a color value.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function GetGValue Lib "VBZUTIL.DLL" Alias
   "GetRedValue" (ByVal RGBVal&)
```

Parameters

Parameter	**Description**
RGBVal&	Long integer - a VB color value

Returns

Integer - the amount of green (0-255) in a given VB color value

Usage

```
GValue% = GetGValue (ClrValue&)
```

Comments

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out. Therefore we are providing you with GetRValue, GetGValue and GetBValue.

The *VBZ* Utility Library: GetBValue

Purpose

Returns the amount of Blue (0-255) in a color value.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function GetBValue Lib "VBZUTIL.DLL" Alias
   "GetRedValue" (ByVal RGBVal&)
```

Parameters

Parameter	**Description**
RGBVal&	Long integer - a VB color value

Returns

Integer - the amount of blue (0-255) in a given VB color value

Usage

```
BValue% = GetBValue (ClrValue&)
```

Comments

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out.   Therefore we are providing you with GetRValue, GetGValue and GetBValue.

The *VBZ* Utility Library: CtlTypeOf

Purpose

Returns the class name of the control.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function CtlTypeOf$ Lib "VBZUTIL.DLL" (C As Control)
```

Parameters

Parameter	**Description**
C	Control variable - the control for which you want a control-type

Returns

String - the control-type of a given control

Usage

```
CtlType$ = CtlTypeOf$ (Screen.ActiveControl)
```

Comments

The TypeOf syntax in VB is alien to the rest of the BASIC language. Instead of being able to do a simple select case statement, you need to do as series of checks using the syntax **If TypeOf Ctl Is "CtlType" then**. With the CtlTypeOf$ function, you can employ a more familiar syntax such as:

```
Select Case TypeOf$ (Screen.ActiveControl)
   Case "ListBox"
   Case "ComboBox"
etc.
```

The *VBZ* Utility Library: CtlParentClass

Purpose

Returns the name of the parent class, if any, of a control.

Contained In

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

Declaration

```
Declare Function CtlParentClass$ Lib "VBZUTIL.DLL" (C As
   Control)
```

Parameters

Parameter	**Description**
C	Control variable - the control for which you seek the parentclass

Returns

String - the name of the parentclass of a given control

Usage

```
Class$ = CtlParentClass$ (List1)
```

Comments

There are two trends that require this function. The first is the proliferation of custom controls and the other is the increasing use of Windows API functions. If you want to write a generic routine to to search either a list box or combo box, you would have a lot of work ahead of you. If you just wanted it to work with the standard VB controls, you could use our CtlTypeOf function. However, if you wanted it to work with third-party custom controls, it would be a little tougher to determine whether the control was a list box or combo box. There is where CtlParentClass comes in. It returns the name of the control class (in Windows) on which a custom control is based. It will work with standard VB controls and with third party custom controls.

The *VBZ* Utility Library: MakeCtlArray

Purpose

Dimensions and assigns array of type Control from a control array.

Contained in

VBZUTIL.BAS

Declaration

```
Function MakeCtlArray (F as Form, C() as Control, ctrlS$)
```

Parameters

Parameter	**Description**
F	form variable form that contains the control array
C()	control variable array array to be assigned controls
ctrls$	string name of control

Returns

Integer   the number of elements of the control variable array

Usage

Make every element in a control array blue:

```
For i = 0 to MakeCtlArray (form1, C(), (entry.text))
   C(i).BackColor = &HFF
Next
```

Every so often, there are functions for which it becomes necessary to throw in a DLL. Many of you have requested that we stop creating one-function DLLs and this makes sense. The result of this request is VBZUTIL.DLL which is designed to contain all of these little utility functions that don't quite belong anywhere else. Don't mistake these for trivial functions, because if they were, they wouldn't have to be in a DLL. In fact, it is in this library that some of the most important and creative functions will be found.

What goes into this DLL is guided both by our discoveries and needs and requests from you. All that we ask is that you don't request standard functions that can be found in any commercial add-on package. In other words, don't look for a routine to sort integer arrays any time soon. Other than that, the sky's the limit. You never know what's possible unless you ask and we can never anticipate all of our subscribers' needs. The functions in this library will often be used in context in the sample programs that come with \I*VBZ\i*, but there will be no samples devoted to these functions. If you would like this to be otherwise, let us know.

The function reference for VBZUTIL.DLL will always be updated instead of being fragmented over every issue. This is the one example where you can get the latest issue and know that you are not missing anything, which is not the case for the DLLs and samples in the features. Functions new to this issue are marked with ***.**

\B\UVBZUTIL Function Reference\b\u

The declarations can be found in VBZUTIL.BAS.

~~CtlName~~
~~CtlParentClass~~
~~CtlTypeOf~~
~~CtlUBound & CtlLBound~~
~~GetBValue~~
~~GetGValue~~
~~GetHInstance~~
~~GetRValue~~
~~HWnd2Ctl~~
~~HIWORD & LOWORD~~
~~LPSTR2Str~~
~~LP2Str~~
~~MakeArray*~~
~~MAKELONG~~
~~ReCreateControlhWnd~~
~~Str2Ctl~~
~~USHORT~~

\BThe \I*VBZ\i* Utility Library: CtlName\b

\UPurpose\u

Returns the name of a control as a string.

\UContained in\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
    Declare Function CtlName$ Lib "VBZUTIL.DLL" (C As Control)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
C	Control variable - Control for which name is needed

\UReturns\u

The name of the control as a string

\UUsage\u

```
    tbName$ = CtlName$ (Text1)
```

\UComments\u

This function exists because of the unavailability of the .Name property of a control at runtime.   You use it in much the same way you would a control, however.

\BThe \I*VBZ\i* Utility Library: CtlUBound & CtlLBound\b

\UPurpose\u

Returns the upper and lower bounds of a control array.

\UContained in\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclarations\u

```
    Declare Function CtlUBound Lib "VBZUTIL.DLL" (C As Control)
    Declare Function CtlLBound Lib "VBZUTIL.DLL" (C As Control)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
C	Control variable - any element of a control array

\UReturns\u

Integer - the highest or lowest array index of control array

\UUsage\u

```
    UpperBound% = CtlUBound (MyControl (0))
    LowerBound% = CtlLBound (MyControl (7))
```

\UComments\u

Don't pass a control to either of these functions unless you know it to be a member of a control array.   *Don't* pass the control array with an open paren.   You must pass a valid control to the function, but it can be any element in the array.

\BThe \I*VBZ*\i Utility Library: GetHInstance\b

\UPurpose\u

Returns the instance handle of your current application.

\UDeclaration\u

```
    Declare Function GetHInstance Lib "VBZUTIL.DLL" ()
```

\UReturns\u

Integer - the instance handle of the current application

\UUsage\u

```
    myhInstance% = GetHInstance()
```

\UComments\u

This value is necessary for a number of Windows API functions, so it is provided here.

\BThe \I*VBZ*\i Utility Library: HWnd2Ctl\b

\UPurpose\u

Returns the element into the VB2 .Controls() collection of a form, given the control's hWnd.

\UContained in\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
    Declare Function HWnd2Ctl% Lib "VBZUTIL.DLL" (ByVal hWnd%)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
hWnd%	Integer - hWnd of a control

\UReturns\u

Integer - the index of the control array to which a given control belongs

\UUsage\u

```
Dim C As Control
'you got the Wnd% from someplace, like MWATCH
El = HWnd2Ctl% (Wnd%)
If El > -1 then Set C = Me.Controls(El)
```

\UComments\u

At this point, you may use C as if it were a normal control and access all of its properties directly.   This function is very useful in applications when you have an hWnd but you need access to the underlying properties of the control.

\BThe \IVBZ\i Utility Library: HIWORD & LOWORD\b

\UPurpose\u

Returns the HiWord or LoWord of a LONG integer.

\UContained in\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclarations\u

```
Declare Function HIWORD Lib "VBZUTIL.DLL" (ByVal LongVal&)
Declare Function LOWORD Lib "VBZUTIL.DLL" (ByVal LongVal&)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
LongVal&	Long integer - the integer for which you want low or high word

\UReturns\u

Integer - the HiWord or LoWord

\UUsage\u

```
HW% = HIWORD (MyLong&)
LW% = LOWORD (MyLong&)
```

\UComments\u

Often a DLL function will return a long integer which contains two short integers. These functions will help you to parse out those values.

\BThe \IVBZ\i Utility Library: LPSTR2Str\b

\UPurpose\u

Creates a Visual Basic string from a C null terminated string (LPSTR).

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclarations\u

```
    Declare Function LPSTR2Str$ Lib "VBZUTIL.DLL" (ByVal LPSTR&)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
Lpstr&	Long integer - pointer to a string

\UReturns\u

String - VB string created from C-type null-terminated string

\UUsage\u

```
    MyString$ = LPSTR2Str$ (lpstr&)
```

\UComments\u

Sometimes a DLL function will return a C language string and you will want to get this into a VB string.

\UThe \I*VBZ\i* Utility Library: LP2Str\u

\UPurpose\u

Creates a VB language string from any block of memory.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
    Declare Function LP2Str$ Lib "VBZUTIL.DLL" (lp As Any, ByVal
      nBytes)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
lp	Variant - pointer to a memory location
nBytes	Integer - number of bytes to retrieve into string

\UReturns\u

String - VB string created from a number of bytes starting at a given memory location.

\UUsage\u

```
    MyString$ = LP2Str$ (ByVal lp, 283)
    MKI$ = LP2Str$ (MyInt, 2)
```

\BThe \I*VBZ\i* Utility Library: MAKELONG\b

\UPurpose\u

Create a LONG integer from two short integers.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
Declare Function MAKELONG Lib "VBZUTIL.DLL" (ByVal loword%,
  ByVal hiword%) As Long
```

\UParameters\u

\BParameter\b	**\BDescription\b**
loword%	Integer - Short integer to become the low word of a long
hiword%	Integer - Short integer to become the hi word of a long

\UReturns\u

Long integer - The long made from two short integers

\UUsage\u

```
MyLong& = MAKELONG (Var1%, Var2%)
```

\UComments\u

Often a DLL function requires a LONG integer parameter which is really two short integers. This function will allow you to call those functions. It is identical to the MAKELONG macro which is listed in the SDK documentation, but that is unavailable in a Windows DLL.

\BThe \I*VBZ\i* Utility Library: ReCreateControlhWnd\b

\UPurpose\u

Recreates the hWnd associated with a particular control.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
Declare Sub ReCreateControlhWnd Lib "VBZUTIL.DLL" (C As
  Control)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
C	Control variable - Control for which you want to recreate an hWnd

\UUsage\u

This sample changes a single-column, single-select list box to multi-column, multi-select. While this is no longer necessary with a standard list in VB2 (though File Lists still can't be multi-column for some reason), it demonstrates the use of the DLL.

```
OldLong& = GetWindowLong&(File1.hWnd, GWL_STYLE)
OldLong& = OldLong& Or LBS_EXTENDEDSEL Or LBS_MULTICOLUMN
SetWindowLong File1.hWnd, GWL_STYLE, OldLong&
RecreateControlHwnd File1
File1.Refresh
```

You can also look at the D&DTEST1.MAK sample that came with VBZ01.

\UComments\u

This function is useful when you need to change a pre-hwnd style of a control dynamically or when you need to gain access to a style that has not been exposed by VB, such as justification in a text box.   This is an advanced function.   \BUse it with care!\b

\BThe \I*VBZ\i* Utility Library: Str2Ctl\b

\UPurpose\u

Returns the element into the VB2 .Controls() collection of a form, given the control name as a string.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
Declare Function Str2Ctl% Lib "VBZUTIL.DLL" (Frm As Form,
  ByVal CtlName$)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
Frm	Form variable - the form with the control array
CtlName$	String - the name of the control array

\UReturns\u

Integer - the element within the Controls collection that is a given control

\UUsage\u

```
Dim C As Control
CtlName$ = "Text1"
El = Str2Ctl% (Me, CtlName$)
If El > -1 then Set C = Me.Controls(El)
```

\UComments\u

At this point, you may use C as if it were a normal control and access all of its properties directly.   This function is very useful in database applications when you want to be able to relate control names to corresponding column names in a database table.

\BThe \I*VBZ\i* Utility Library: USHORT\b

\UPurpose\u

Returns the unsigned version of an integer into a long integer.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
Declare Function USHORT& Lib "VBZUTIL.DLL" (ByVal Word)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
Word	Integer - an unsigned integer you wish to convert to a long integer

\UReturns\u

Long integer - A long integer created from an unsigned integer

\UUsage\u

```
Unsigned& = USHORT& (MyWord%)
```

\UComments\u

VB lacks the unsigned integer data type (often called a WORD).   This function will allow you to calculate the actual value from an integer, even one which has moved into negative numbers.

\UThe \I*VBZ\i* Utility Library: GetRValue\u

\UPurpose\u

Returns the amount of Red (0-255) in a color value.

\UDeclaration\u

```
Declare Function GetRValue Lib "VBZUTIL.DLL" Alias
   "GetRedValue" (ByVal RGBVal&)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
RGBVal&	Long integer - a VB color value

\UReturns\u

Integer - the amount of red (0-255) in a given VB color value

\UUsage\u

```
    RValue% = GetRValue (ClrValue&)
```

\UComments\u

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out.   Therefore we are providing you with GetRValue, ~~GetGValue~~ and ~~GetBValue~~.

\BThe \I*VBZ\i* Utility Library: GetGValue\b

\UPurpose\u

Returns the amount of Green (0-255) in a color value.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
    Declare Function GetGValue Lib "VBZUTIL.DLL" Alias
      "GetRedValue" (ByVal RGBVal&)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
RGBVal&	Long integer - a VB color value

\UReturns\u

Integer - the amount of green (0-255) in a given VB color value

\UUsage\u

```
    GValue% = GetGValue (ClrValue&)
```

\UComments\u

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out.   Therefore we are providing you with ~~GetRValue~~, GetGValue and ~~GetBValue~~.

\UThe \I*VBZ\i* Utility Library: GetBValue\u

\UPurpose\u

Returns the amount of Blue (0-255) in a color value.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
        Declare Function GetBValue Lib "VBZUTIL.DLL" Alias
           "GetRedValue" (ByVal RGBVal&)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
RGBVal&	Long integer - a VB color value

\UReturns\u

Integer - the amount of blue (0-255) in a given VB color value

\UUsage\u

```
        BValue% = GetBValue (ClrValue&)
```

\UComments\u

While the RGB function exists to construct a color from it's component Red, Green and Blue parts, there are no functions to split these values back out.   Therefore we are providing you with ~~GetRValue~~, ~~GetGValue,~~ and GetBValue.

\BThe \I*VBZ\i* Utility Library: CtlTypeOf\b

\UPurpose\u

Returns the class name of the control.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
        Declare Function CtlTypeOf$ Lib "VBZUTIL.DLL" (C As Control)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
C	Control variable - the control for which you want a control-type

\UReturns\u

String - the control-type of a given control

\UUsage\u

```
        CtlType$ = CtlTypeOf$ (Screen.ActiveControl)
```

\UComments\u

The TypeOf syntax in VB is alien to the rest of the BASIC language.   Instead of being able to do a simple select case statement, you need to do as series of checks using the syntax \B**If TypeOf Ctl Is "CtlType" then\b**.   With the CtlTypeOf$ function, you can employ a more familiar syntax such as:

```
Select Case TypeOf$ (Screen.ActiveControl)
   Case "ListBox"
   Case "ComboBox"
etc.
```

\BThe \I*VBZ\i* Utility Library: CtlParentClass\b

\UPurpose\u

Returns the name of the parent class, if any, of a control.

\UContained In\u

VBZUTIL.BAS
(Requires VBZUTIL.DLL)

\UDeclaration\u

```
Declare Function CtlParentClass$ Lib "VBZUTIL.DLL" (C As
   Control)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
C	Control variable - the control for which you seek the parentclass

\UReturns\u

String - the name of the parentclass of a given control

\UUsage\u

```
Class$ = CtlParentClass$ (List1)
```

\UComments\u

There are two trends that require this function.   The first is the proliferation of custom controls and the other is the increasing use of Windows API functions.   If you want to write a generic routine to to search either a list box or combo box, you would have a lot of work ahead of you.   If you just wanted it to work with the standard VB controls, you could use our ~~CtlTypeOf~~ function.   However, if you wanted it to work with third-party custom controls, it would be a little tougher to determine whether the control was a list box or combo box.   There is where CtlParentClass comes in.   It returns the name of the control class (in Windows) on which a custom control is based.   It will work with standard VB controls and with third party custom controls.

\BThe \I*VBZ\i* Utility Library: MakeArray\b

\UPurpose\u

Dimensions and assigns array of type Control from a control array.

\UContained in\u

VBZUTIL.BAS

\UDeclaration\u

```
Function MakeArray (F as Form, C() as Control, ctrlS$)
```

\UParameters\u

\BParameter\b	**\BDescription\b**
F	form variable - form that contains the control array
C()	control variable array - array to be assigned controls
ctrls$	string - name of control

\UReturns\u

Integer - the number of elements of the control variable array

\UUsage\u

Make every element in a control array blue:

```
For i = 0 to MakeArray (form1, C(), (entry.text))
   C(i).BackColor = &HFF
Next
```