Listing 1 Listing 2

```
library VBDTool;
uses WsDos, WinDos, wintypes, winprocs,   vbapi_, strings;
{$R VBDTool.RES}
{$D Micro System Solutions - MS VB3.0 dTool}
```

Creating Visual Basic VBXs:

The Visual Basic Disk Tool

by Fred C. Hill

The fifth in a series of articles on the secrets of writing VBX controls in Borland Pascal.

In this issue we conclude the series on writing VBX's using Borland Pascal. To finish it out we will be building a control which will provide many of the tools which the various releases of Visual Basic up through version 3.0 have failed to provide in an adequate manner. These tools will provide information about the disk drives on your system. In addition, a Visual Basic program has been provided which will give you substantial information about the rest of your system.

The VBX is called **VBDTOOL.VBX** for 'Visual Basic Disk Tool' and the VB program is called **SysStat** for 'System Status'. Each of these tools are useful in there own right and while copyrighted code they may be freely modified, used in your programs and distributed to others. If changes are made I would appreciate a copy of the result.

The VBX has one apparent bug in it. It returns an incorrect value when testing for a CDROM. I have explored many options in code but the Visual Basic environment does not seem to return the same values as the Windows 3.11, the Windows for Workgroups 3.11, and MSDOS systems do. The referenced code in the WsDOS unit works fine in all of the mentioned environments, however, In Visual Basic the test for a CDROM always returns an invalid drive status. If anyone can identify the problem I'd appreciate a message outlining the solution. Please include a return address or phone number so I may contact you.

This control uses the WsDOS unit from Turbo Power Software's Win/Sys library. Three routines are used from the unit, the GetVolumeLabel routine which returns the disk label, the GetDiskClass routine which returns the disk type, and GetDiskInfo which returns all of the statistical information about the disk. The statistic information is then used to calculate the data not directly available.

So, lets get to the code. As in all of the previous articles the setup for a VBX remains the same. The BitMap resources must be created and defined as constants. These are the Up, Down, Mono and EGA presentations of the tool. These are used in the VB Toolbox and in representing the control on the VB Form. The VBDTool is an invisible control and has no visual presence during run-time.

The control will actually be a DLL so we start it with library rather than program. After we compile the DLL we will rename it to be VBDTOOL.VBX from the default of VBDTOOL.DLL. This isn't really necessary except that Visual Basic will automatically include a VBX in the add-a-file list where a DLL will have to be specifically asked for. In other words you'll save about 10 key strokes by making the change now. The next statement identifies the resource file and is followed by the copyright statement ($D).

The Uses clause includes the WsDOS unit from Turbo Power Software and the vbAPI_ unit provided on the enclosed disk.

Listing 3

```
type
   pdTool = ^tdTool;
   tdTool = record
      usPathLen:   integer;
      hszPathString:    Hsz;
      hszDiskType:   Hsz;
      hszDrive:      Hsz;
      hszVolume:   Hsz;
      ulSize:        longInt;
      hszDate:         Hsz;
      hszTime:         Hsz;
      ulBytesPerCluster:   longint;
      ulDiskCapacity:   longInt;
      ulFreeSpace:         longInt;
      usClustersAvail:   longint;
      usTotalClusters:   Longint;
      usBytesPerSector:   longint;
      usSectorsPerCluster:   longint;
      usAction:   Integer;
 end;

const
      cVbxUsers:             integer = 0;
```

The record definition holds all of the data passed between the control and the VB program. Each of these is further defined in the property list which follows.

The Property table will include each of the standard properties as well as the custom properties for this control. This control has very few standard properties and many custom ones.

The standard properties used are the Name, Tag, Left, and Top. The name, of course, is needed to give the control a unique name and is required by Visual Basic. The tag is used by the developer to store data about the control. The left and top are required to allow VB to position the on-screen control somewhere visible during development time.

The custom properties are by-far the longest list. They include:

PathLen:	Length of a passed string.
PathStr:	A passed string.
DiskType:	The type of disk found. (See discussion below)
Drive:	The drive letter. Does not include the colon.
Volume:	The disk volume name.
Size:	The total space used on the volume.
Date:	The date the volume was created.
Time:	The time of day the volume was created.
BytesPerCluster:	Number of bytes in a single disk cluster.
DiskCapacity:	Total number of bytes allowed on the disk.
FreeSpace:	Total number of bytes available on the disk.
ClustersAvailable:	Total disk clusters available.
TotalClusters:	Total number of clusters on the disk.
BytesPerSector:	Total number of bytes per disk sector.
SectorsPerCluster:	Number of disk sectors per disk cluster.
Action:	Tells the VBX what action to perform. (see discussion below)

Most of these are self-explanatory and don't need further discussion but a few do require further explanation as to their use.

Drive:
One of two custom properties which can be written to. This property identifies the disk you which checked. Only the drive letter gets placed here, colons are not required nor allowed.

Action:
The second and last writeable property. This is used to initiate an action. While this method is used for a lot of controls, I could just as easily have used the Drive letter to trigger the controls actions. Any reference to a control would suffice. The allowable actions are: 1: Test the length of a string passed in Path String. and 2: Test the disk drive whose drive letter is passed in Drive.

PathLen and PathStr:
These two properties are included to show how data can be passed between the control and Visual Basic. In VB, place a string in PathStr and a 1 in Action. PathLen will contain the length of the string passed.

DiskType:
The utilities provided by Turbo Power Software are very general in nature, meaning that they don't make any assumptions about the platform they'll be running on. As a result they can provide fairly complete information about the system. The disk type returned by this library is no exception. It identifies drives as floppy drives (360K, 720K, 1.2M, 1.44M and Other), Removeable drives (Bernoulli), Hard Drives, Ram Drives, Lan Drives (Novell) and CDROM drives. It also returns a type of Substitute drive when the end user uses the MSDOS SUBST command. The last two types it returns are 'Invalid drive' when the drive type appears invalid and 'unknown drive' when a drive is found but cannot be identified.

Disk information: The information returned by the control is that provided by the GetDiskInfo routine in the WsDOS unit as well as information calculated in the control.

Since the control executes at a much higher 'compiled' speed it is to your advantage to perform as many operations in the VBX as possible, even if it is 'easier' to perform the same task in VB.

Events
This control uses no custom events although, using the code presented in issue 3, you could easily add custom events to pass significant disk information between VB and the control. An Event would also be a convenient place to provide error handling.

Control Procedure
As with all controls, this one needs a control procedure. (See Listing #) As you remember, the control procedure is the message loop which every Windows program needs to handle messages passed from one operating event to another. Only the messages directly relating to our control will get referenced in the message loop. These will be: WM_SIZE to resize the visual control. Since we do not have a visual presence in the run-time we restrict the development control to the size of the bitmap; WM_PAINT to place the control on the form and to re-paint it whenever Windows feels the control needs repainting; VBM_SETPROPERTY which will handle all of the property transfers. Within the VBM_SETPROPERTY code we examine the action code to see what this pass of the control should be doing. We have limited this loop to either checking the size of the PathStr (Gen_StrLen) or getting the disk information (ReadVolume). Within ReadVolume, all of the real important work get done. Each of the properties will be initialized, read or set and prepared for returning a value to the VB environment. The WsDos unit will be called to read the disk in the requested drive and return its status. The data is then converted to the appropriate data type for use by Visual Basic.

Near the beginning of the VBM_SETPROPERTY message loop we Dereference the control memory address and begin looking for data. The pointer to the disk letter (lpDrv^) is passed to the DiskType routine which in turn calls the WsDOS routine GetDiskClass to test the drive letter passed to it. On the return the disk type is assigned a string with the disk type name in it. This string is then converted to an Hsz string for use by Visual Basic. This happens for each of the string items being passed through the control.

After the volume name has been read and converted for use by VB the disk information is read using the TPS unit WsDOS and the function GetDiskInfo. The data collected by this routine is either converted for VB or used in

Listing 4

```
const
ModelDefCtlName:        array[0..8] of Char = 'DiskTool'#0;  { default control name prefix}
ModelClassName:                 array[0..15] of Char = 'ThunderDiskTool'#0;{ Visual Basic class name}
ModelParentClassName:           array[0..8] of Char = #0;   { Parent window class if subclassed}
ModelFmtTool: TMODEL = (
        usVersion:                      VB_VERSION;             { VB version used by control}
        fl:                 {Model_fInvisAtRun or }Model_fInitMsg or Model_fLoadMsg;   { Bitfield structure}
        ctlproc:            TFarProc(@CtlProc);                 { The control proc.}
        fsClassStyle:       cs_VRedraw or cs_HRedraw;       { window class style}
        flWndStyle:         WS_Child or WS_Border;{ default window style}
        cbCtlExtra:         sizeof(tdTool);                     { # bytes alloc'd for HCTL structure}
        idBmpPalette:       UpTool;                             { BITMAP id for tool palette}
        DefCtlName:         tOffset(@ModelDefCtlName);      { default control name prefix}
        ClassName:          tOffset(@ModelClassName);       { Visual Basic class name}
        ParentClassName:            0 {tOffset(@ModelParentClassName)};     { Parent window class if subclassed}
        proplist:           ofs(PropertyList);                  { Property list}
        eventlist:          0;          { Event list}
        nDefProp:           0;          { index of default property}
        nDefEvent:          0;          { index of default event}
        nValueProp:         0           { default value }
);
```

calculations and converted. This is a lot of information to collect from the disk but actually taken very little time for the disk to provide it. The longest time is getting the disk to answer the page to begin with.

The Model Table

While rather unassuming, the Model table is one of the more important items in the control. This table points Visual Basic to all of the other components.

The VBINITCC and VBTERMCC routines finish out the control and are pretty standard. The VBINITCC routine can test the environment to see where it is running and reacte appropriately. For example, if you wanted to restrict the use of the control to the development environment and you discovered you were in the run-time, you could pop up a message box alerting the user than the control was not available in that environment and then abnormally terminate. This would preclude the Visual Basic program from running due to a error loading a control. Thsi is all explained in the CDK documentation from Microsoft Inc.

To create a loadable VBX, unzip the code from the disk into a directory on your hard drive. Compile the VBDTOOL.PAS file using BPW 7.0. When the code is compiled you'll have a file named VBDTOOL.VBX in your directory. Rename this file to VBDTOOL.VBX and move it to your \WINDOWS\SYSTEM directory. Run Visual Basic and load the VB program using the DSTAT.MAK file. Change the disk drive letter and click on the Examine button to see the info from the drive.

Reference: Professional Features Book 1, Microsoft Visual Basic 3.0 Custom Control Guide

Fred Hill is president of Micro System Solutions. His company produces DOS and Windows applications in Borland Pascal 7.0 and Visual Basic 3.0. He may be reached on Compuserve 76060,102