Visual Basics

By Barry Seymour

Mouse Patrol

Remember the old "Outer Limits" TV show? Okay, you don't have to remember it from when it was originally **on**, you just have to remember a rerun from somewhere. (That's all **I** remember). At the opening of the show, the announcer intones "we control the video -- we control the audio."

Well, I'm not sure about the video, and the audio definitely has to wait for a future column, but this week we can show you how to control the mouse. Not the mouse as the USER moves it, but the mouse as YOU move it, in Visual Basic code. You can move the mouse yourself, or limit where the mouse is allowed to go on the screen. This is a small but necessary step towards world domination.

Say you want your program to demonstrate some technique to the user. You'll want your **code** to move the mouse, won't you? Using the Windows API this is a two step technique. First you specify the area the mouse is allowed to be in -- this is known as 'clipping' the cursor -- and then you move the mouse cursor there. You can put the mouse exactly where you want it by specifying only one point as being allowable for the mouse cursor. A call to move the mouse **anywhere** puts it right where you want it!

Putting this procedure in a loop allows you to move the mouse any way you want to; you're only limited by your ability to write hip algorithms describing the mouse's movement.

The two API calls you'll need are **ClipCursor** and **SetCursorPos**. We'll show you the global declarations you'll need for these functions, as well as the related user-defined type RECT; then we'll then explain them one at a time...

```
Declare Sub ClipCursor Lib "User" (lpRect As Any)
Declare Sub SetCursorPos Lib "User" (ByVal X As Integer, <+>
  ByVal Y As Integer)
Type RECT
    Left As Integer
    Top As Integer
    right As Integer
    Bottom As Integer
End Type
```

The ClipCursor sub 'clips' the area the mouse is allowed to be in. It's sort of like the invisible line you used to draw across your room to keep your little brother out (or in), only this one **works**. You use the user-defined type RECT to create a variable holding the bounds of the area allowed. You then pass that variable to the API call, and the mouse is locked into the area

you defined.

There are some things to keep in mind when using ClipCursor...

 The variables in RECT refer to an area defined in pixels. Since the Screen object in Visual Basic is defined in twips, you need to allow for the conversion between twips and pixels. (There are 15 twips to a pixel.)

 Also, **remember to restore the clipped area to the full screen when you're done!** It is supremely irritating for a user to discover that he can't move his/her mouse onto the menu, or to the quit button. You don't want to antagonize the user, do you?

The SetCursorPos call is used to put the mouse cursor where you want to. Parameters passed to this call should also be in pixels. You can use this any way you want to, but for this exercise all you need to know is that **any** call to move the mouse cursor will reposition the mouse inside the currently clipped area. It's not enough to call ClipCursor; you have to call SetCursorPos to actually get that little mouse pointer inside the invisible lines you've established. Parameters passed to SetCursorPos are also in pixels.

Putting it all together, we get this: Pass a RECT variable to ClipCursor to define a **single point** where your mouse can be; then call SetCursorPos with **any** values (0,0 will do) and the mouse will obediently jump into that one-pixel area...

```
Dim CursorRect as RECT
CursorRect.Top = 200 'these numbers are arbitrary.
CursorRect.Left = 300 'these numbers are arbitrary.
CursorRect.Right = CursorRect.Left ' same as left!
CursorRect.Bottom = CursorRect.Top ' same as top!
ClipCursor CursorRect
SetCurSorPos 0, 0
```

Note how we set the allowable region for the cursor to an area one pixel high by one pixel wide. The call to SetCursorPos places the mouse pointer at that exact spot, since it's not allowed anywhere else. The result is **total control.**

You can easily place this code snippet inside a loop or some other algorithm to modify the values of RECT and make the mouse pointer move around on the screen. You can also insert DoEvents() calls inside the loop to control the speed of the mouse and allow Windows to process other events.

In fact, a DoEvents() inside the loop is probably a good idea; if you have enough of them, the mouse cursor will change to a bar over a text box, a resizing arrow over a window border -- in fact, it'll behave just like it always does when the user moves it. If you want smoother motion and greater speed, and also wish to prevent Windows from processing other events (including

the redraw of your form after you click the button) leave DoEvents() out of the loop.

This week's example...

VBEX017 moves the mouse up and down, left and right on the screen at various speeds. A control array of buttons is used to initiate mouse movement, and the index of the button pushed determines how many DoEvents() calls are put in the loop. The more calls, the slower the cursor movement.

To duplicate this example, create the form VB017EX with the following controls...

CONTROL	CAPTION	PROPERTIES
Command1()	Slow	Control Array - elements 0,1,2
	Medium	
	Fast	
Command2	Quit	

Save the form as VB017EX.FRM. Create a global form called VB017EX.GBL. Paste code for each file in from the example as indicated and run it. When you run the program you'll see the mouse move right and left, down and up on your screen at various speeds.

As always, this column plus sample code is available on the Windows Online BBS in Danville, California, phone 1 510 736-8343. This column in Windows HELP format (VB017EX.HLP) plus the Visual Basic source code is in VB017EX.ZIP, and may be distributed as freeware.

Barry Seymour
Marquette Computer Consultants
San Rafael, CA 415/459-0835
for Windows Online "the Weekly"

'Code Sample

Paste the following code into your GLOBAL module...
Join all lines broken by the LINE JOIN code (**<+>**)

```
Declare Sub ClipCursor Lib "User" (lpRect As Any)
Declare Sub SetCurSorPos Lib "User" (ByVal X As Integer, <+>
  ByVal Y As Integer)

Type RECT
    Left As Integer
    Top As Integer
    right As Integer
    Bottom As Integer
End Type

Global CursorRect As RECT

Global ScreenHeight As Integer
Global ScreenWidth As Integer
```

Paste the following code into your VB017EX.FRM form...

```
Sub Form_MouseUp (Button As Integer, Shift As Integer, X As Single, Y
As Single)
    MousePressed = 0
End Sub

Sub Command1_Click (Index As Integer)

    Speed% = (Index * 10) + 1
    ' INDEX SETS THE SPEED OF THE MOUSE.
    ' the higher the index, the fewer the
    ' DoEvents() calls, hence the higher the speed.
    ' We also add one to avoid a DIVIDE BY ZERO error.
```

```
'RESTORE CURSOR TO FULL SCREEN, Just in case..
CursorRect.Top = 0
CursorRect.Left = 0
CursorRect.right = ScreenWidth
CursorRect.Bottom = ScreenHeight
ClipCursor CursorRect

SetCurSorPos 400, 300

For X% = 0 To ScreenWidth ' left to right
    CursorRect.Top = 200
    CursorRect.Left = X%
    CursorRect.right = CursorRect.Left
    CursorRect.Bottom = CursorRect.Top
    ClipCursor CursorRect
    SetCurSorPos 0, 0
    If X% Mod Speed% = 0 Then D% = DoEvents()
Next X%

For X% = ScreenWidth To 0 Step -1 'Right To Left
    CursorRect.Top = 200
    CursorRect.Left = X%
    CursorRect.right = CursorRect.Left
    CursorRect.Bottom = CursorRect.Top
    ClipCursor CursorRect
    SetCurSorPos 0, 0
    If X% Mod Speed% = 0 Then D% = DoEvents()
Next X%

For Y% = 0 To ScreenHeight 'top to bottom
    CursorRect.Top = Y%
    CursorRect.Left = 250
    CursorRect.right = CursorRect.Left
    CursorRect.Bottom = CursorRect.Top
    ClipCursor CursorRect
    SetCurSorPos 0, 0
    If X% Mod Speed% = 0 Then D% = DoEvents()
Next Y%

For Y% = ScreenHeight To 0 Step -1
    CursorRect.Top = Y%
    CursorRect.Left = 250
    CursorRect.right = CursorRect.Left
    CursorRect.Bottom = CursorRect.Top
    ClipCursor CursorRect
    SetCurSorPos 0, 0
    If X% Mod Speed% = 0 Then D% = DoEvents()
Next Y%

CursorRect.Top = 0
CursorRect.Left = 0
CursorRect.right = ScreenWidth
```

```
    CursorRect.Bottom = ScreenHeight
    ClipCursor CursorRect

    SetCurSorPos ScreenWidth / 2, ScreenHeight / 2
End Sub

Sub Form_Load ()
    'init global vars...
    ScreenHeight = screen.Height / 15
    ScreenWidth = screen.Width / 15
End Sub

Sub Command2_Click ()
    CursorRect.Top = 0
    CursorRect.Left = 0
    CursorRect.right = screen.Width / 15
    CursorRect.Bottom = screen.Height / 15
    ClipCursor CursorRect
    End
End Sub
```

Marquette Computer Consultants

Visual Solutions for Business Computing

Marquette Computer Consultants provides a broad range of services for business computing in the San Francisco Bay Area using IBM compatible desktop computers in the Microsoft Windows 3.x environment. **MCC** offers the following services...

- Windows Programming in Microsoft Visual Basic
- Windows installation, optimization training and support
- Database programming for Windows / DOS applications in standalone, networked and / or SQL Server environments
- Hardware support, assistance and troubleshooting

Barry Seymour of Marquette Computer Consultants has extensive experience in Visual Basic, having worked exclusively in that environment ever since Microsoft distributed pre-release evaluation copies of "Thunder" to selected developers in May of 1991. He has also had experience in developing Visual Basic programs in the client-server environment, using Microsoft's Visual Basic/SQL Server link.

Mr. Seymour also has supported Windows in a multiple LAN environment and has provided hardware and software support for Novell and LAN Manager file servers and workstations in complex, client-server oriented development environments.

Mr. Seymour is also the author of *TaskTracker for Windows,* a shareware time management and reporting program currently available on Windows OnLine, CompuServe, BIX and many other bulletin board services.

Mr. Seymour writes the VISUAL BASICS column for Windows OnLine and WUGNET.

Marquette Computer Consultants
22 Sirard Lane
San Rafael, CA 94901-1066
(415) 459-0835