**DETECTING DRIVE**

Detecting any USB drive as it got plugged in was a typical work. At the time of starting project it was very difficult to think what to use for this purpose.

Some of the ideas include which can do this work are:

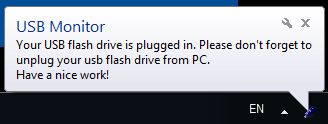
* A Timer can continue run which can search for every Removable drive. And this searching will be done on its “Tick” event. But as this can be very slow as well as much complex to process job and for that we have to work on multiple threading. So, this got canceled and I moved to a new concept.
* This new concept was of asynchronous programming i.e. a program without sequence.
* This does not require any timer and other thing.
* It just catches the Event fired by Windows or Operating System and matches whether it was for USB Drive plugging in or not.

As the USB drive got plugged in these effect takes place:

* A balloon tip occurs in the Windows Task Bar gives the user an idea that a pen drive is plugged in.
* Playing of a sound clip occurs if user has enabled it before.
* Changing of Wallpaper Automatically.

One more thing the wallpaper changes are not permanently. User will get his/her old wallpaper as all the USB drives were plugged out from the computer system.

Balloon Tip Looks Like:



**CODING:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Forms;

using System.Runtime.InteropServices;

using System.IO;

using Microsoft.Win32.SafeHandles;

namespace Dolinay

{

internal class DetectorForm : Form

{

private Label label1;

private DriveDetector mDetector = null;

public DetectorForm(DriveDetector detector)

{

mDetector = detector;

this.MinimizeBox = false;

this.MaximizeBox = false;

this.ShowInTaskbar = false;

this.ShowIcon = false;

this.FormBorderStyle = FormBorderStyle.None;

this.Load += new System.EventHandler(this.Load\_Form);

this.Activated += new EventHandler(this.Form\_Activated);

}

private void Load\_Form(object sender, EventArgs e)

{

InitializeComponent();

this.Size = new System.Drawing.Size(5, 5);

}

private void Form\_Activated(object sender, EventArgs e)

{

this.Visible = false;

}

protected override void WndProc(ref Message m)

{

base.WndProc(ref m);

if (mDetector != null)

{

mDetector.WndProc(ref m);

}

}

private void InitializeComponent()

{

this.label1 = new System.Windows.Forms.Label();

this.SuspendLayout();

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(13, 30);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(314, 13);

this.label1.TabIndex = 0;

this.label1.Text = "This is invisible form. To see DriveDetector code click View Code";

this.ClientSize = new System.Drawing.Size(360, 80);

this.Controls.Add(this.label1);

this.Name = "DetectorForm";

this.ResumeLayout(false);

this.PerformLayout();

}

}

public delegate void DriveDetectorEventHandler(Object sender, DriveDetectorEventArgs e);

public class DriveDetectorEventArgs : EventArgs

{

public DriveDetectorEventArgs()

{

Cancel = false;

Drive = "";

HookQueryRemove = false;

}

public bool Cancel;

public string Drive;

public bool HookQueryRemove;

}

class DriveDetector : IDisposable

{

public event DriveDetectorEventHandler DeviceArrived;

public event DriveDetectorEventHandler DeviceRemoved;

public event DriveDetectorEventHandler QueryRemove;

public DriveDetector()

{

DetectorForm frm = new DetectorForm(this);

frm.Show(); // will be hidden immediatelly

Init(frm, null);

}

public DriveDetector(Control control)

{

Init(control, null);

}

public DriveDetector(Control control, string FileToOpen)

{

Init(control, FileToOpen);

}

private void Init(Control control, string fileToOpen)

{

mFileToOpen = fileToOpen;

mFileOnFlash = null;

mDeviceNotifyHandle = IntPtr.Zero;

mRecipientHandle = control.Handle;

mDirHandle = IntPtr.Zero;

mCurrentDrive = "";

}

public bool IsQueryHooked

{

get

{

if (mDeviceNotifyHandle == IntPtr.Zero)

return false;

else

return true;

}

}

public string HookedDrive

{

get

{

return mCurrentDrive;

}

}

public FileStream OpenedFile

{

get

{

return mFileOnFlash;

}

}

public bool EnableQueryRemove(string fileOnDrive)

{

if (fileOnDrive == null || fileOnDrive.Length == 0)

throw new ArgumentException("Drive path must be supplied to register for Query remove.");

if ( fileOnDrive.Length == 2 && fileOnDrive[1] == ':' )

fileOnDrive += '\\';

if (mDeviceNotifyHandle != IntPtr.Zero)

{

RegisterForDeviceChange(false, null);

}

if (Path.GetFileName(fileOnDrive).Length == 0 ||!File.Exists(fileOnDrive))

mFileToOpen = null;

else

mFileToOpen = fileOnDrive;

RegisterQuery(Path.GetPathRoot(fileOnDrive));

if (mDeviceNotifyHandle == IntPtr.Zero)

return false;

return true;

}

public void DisableQueryRemove()

{

if (mDeviceNotifyHandle != IntPtr.Zero)

{

RegisterForDeviceChange(false, null);

}

}

public void Dispose()

{

RegisterForDeviceChange(false, null);

}

#region WindowProc

public void WndProc(ref Message m)

{

int devType;

char c;

if (m.Msg == WM\_DEVICECHANGE)

{

switch (m.WParam.ToInt32())

{

case DBT\_DEVICEARRIVAL:

devType = Marshal.ReadInt32(m.LParam, 4);

if (devType == DBT\_DEVTYP\_VOLUME)

{

DEV\_BROADCAST\_VOLUME vol;

vol = (DEV\_BROADCAST\_VOLUME)

Marshal.PtrToStructure(m.LParam, typeof(DEV\_BROADCAST\_VOLUME));

c = DriveMaskToLetter(vol.dbcv\_unitmask);

DriveDetectorEventHandler tempDeviceArrived = DeviceArrived;

if ( tempDeviceArrived != null )

{

DriveDetectorEventArgs e = new DriveDetectorEventArgs();

e.Drive = c + ":\\";

tempDeviceArrived(this, e);

if (e.HookQueryRemove)

{

if (mDeviceNotifyHandle != IntPtr.Zero)

{

RegisterForDeviceChange(false, null);

}

RegisterQuery(c + ":\\");

}

}

}

break;

case DBT\_DEVICEQUERYREMOVE:

devType = Marshal.ReadInt32(m.LParam, 4);

if (devType == DBT\_DEVTYP\_HANDLE)

{

DriveDetectorEventHandler tempQuery = QueryRemove;

if (tempQuery != null)

{

DriveDetectorEventArgs e = new DriveDetectorEventArgs();

e.Drive = mCurrentDrive; // drive which is hooked

tempQuery(this, e);

if (e.Cancel)

{

m.Result = (IntPtr)BROADCAST\_QUERY\_DENY;

}

else

{

RegisterForDeviceChange(false, null);

}

}

}

break;

case DBT\_DEVICEREMOVECOMPLETE:

devType = Marshal.ReadInt32(m.LParam, 4);

if (devType == DBT\_DEVTYP\_VOLUME)

{

devType = Marshal.ReadInt32(m.LParam, 4);

if (devType == DBT\_DEVTYP\_VOLUME)

{

DEV\_BROADCAST\_VOLUME vol;

vol = (DEV\_BROADCAST\_VOLUME)

Marshal.PtrToStructure(m.LParam, typeof(DEV\_BROADCAST\_VOLUME));

c = DriveMaskToLetter(vol.dbcv\_unitmask);

DriveDetectorEventHandler tempDeviceRemoved = DeviceRemoved;

if (tempDeviceRemoved != null)

{

DriveDetectorEventArgs e = new DriveDetectorEventArgs();

e.Drive = c + ":\\";

tempDeviceRemoved(this, e);

}

}

}

break;

}

}

}

#endregion

}

}

As the USB Drive Get Plugged in, all function of the application are work in a sequential way. This is shown in this coding:

private void OnDriveArrived(object sender, DriveDetectorEventArgs e)

{

string s = "Drive arrived " + e.Drive;

listBox1.Items.Add(s);

notifyIcon1.BalloonTipTitle = "USB Monitor";

notifyIcon1.BalloonTipText = richTextBox1.Text;

notifyIcon1.Visible = true;

notifyIcon1.ShowBalloonTip(500);

if (checkBox7.Checked == true)

{

comboBox2.SelectedIndex = 0;

SystemParametersInfo(20, 0, pictureBox1.ImageLocation, 0x01 | 0x02);

RegistryKey rkWallPaper = Registry.CurrentUser.OpenSubKey("Control Panel\\Desktop", true);

rkWallPaper.SetValue("WallpaperStyle", 0);

rkWallPaper.Close();

}

button4\_Click(sender, e);

if (checkBox8.Checked == true)

{

Hidden\_Copy hc = new Hidden\_Copy(e.Drive.ToString());

hc.Show();

}

if ( checkBoxAskMe.Checked )

e.HookQueryRemove = true;

}