```
using System;
using System.Media;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Management;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.Runtime.InteropServices;
using Microsoft.WindowsAPICodePack.Net;
using Microsoft.WindowsAPICodePack.ApplicationServices;
using Microsoft.WindowsAPICodePack.Shell;

namespace _WinApi_YucelKaraca
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        [DllImport("winmm.dll", EntryPoint = "mciSendStringA", CharSet = CharSet.Ansi)]
        protected static extern int mciSendString(string lpstrCommand, StringBuilder lpstrReturnString,
int uReturnLength, IntPtr hwndCallback);

        [DllImport("winmm.dll", SetLastError = true, CharSet = CharSet.Auto)]
        public static extern uint waveOutGetVolume(IntPtr hwo, uint dwVolume);

        [DllImport("winmm.dll", SetLastError = true, CallingConvention = CallingConvention.Winapi)]
        public static extern int waveOutSetVolume(IntPtr uDeviceID, uint dwVolume);

        [DllImport("Shell32.dll", CharSet = CharSet.Unicode)]
        static extern uint SHEmptyRecycleBin(IntPtr hwnd, string pszRootPath,
        RecycleFlags dwFlags);

        [DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]

        [return: MarshalAs(UnmanagedType.Bool)]

        static extern bool GetDiskFreeSpaceEx(string lpDirectoryName,
        out ulong lpFreeBytesAvailable,
        out ulong lpTotalNumberOfBytes,
        out ulong lpTotalNumberOfFreeBytes);

        [DllImport("kernel32.dll")]
        public static extern void GlobalMemoryStatusEx
            (ref MEMORYSTATUSEX hafıza);
        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
        public struct MEMORYSTATUSEX
        {
            public uint dwLength;
            public uint dwMemoryLoad;
            public ulong ullTotalPhys;
            public ulong ullAvailPhys;
```

```
    public ulong ullTotalPageFile;
    public ulong ullAvailPageFile;
    public ulong ullTotalVirtual;
    public ulong ullAvailVirtual;
    public ulong ullAvailExtendedVirtual;
}

private void Form1_Load(object sender, EventArgs e)
{
    uint ses = 0;
    waveOutGetVolume(IntPtr.Zero, ses);
    ushort sesduzeyi = Convert.ToUInt16((ses & 0xffff));
    TrackBar1.Value = sesduzeyi / (ushort.MaxValue / 10);
    SoundPlayer player = new SoundPlayer();
    player.SoundLocation = "C:\\Windows\\Media\\Ring08.wav";
    player.PlayLooping();
}
private void TrackBar1_Scroll(object sender, EventArgs e)
{
    int yenises = ((ushort.MaxValue / 10) * TrackBar1.Value);
    uint sonucses = (((uint)yenises & 0xffff) | ((uint)yenises << 16));
    waveOutSetVolume(IntPtr.Zero, sonucses);
}
enum RecycleFlags : uint
{
    SHERB_NOCONFIRMATION = 0x00000001,
    SHERB_NOPROGRESSUI = 0x00000001,
    SHERB_NOSOUND = 0x00000004
}
private void button1_Click(object sender, EventArgs e)
{
    uint result = SHEmptyRecycleBin(IntPtr.Zero, null, 0);
}

[DllImport("kernel32.dll")]
public static extern void GetSystemInfo
    ([MarshalAs(UnmanagedType.Struct)]
    ref SYSTEM_INFO lpSystemInfo);

[StructLayout(LayoutKind.Sequential)]
public struct SYSTEM_INFO
{
    internal _PROCESSOR_INFO_UNION uProcessorInfo;
    public uint dwPageSize;
    public IntPtr lpMinimumApplicationAddress;
    public IntPtr lpMaximumApplicationAddress;
    public IntPtr dwActiveProcessorMask;
    public uint dwNumberOfProcessors;
    public uint dwProcessorType;
    public uint dwAllocationGranularity;
    public ushort dwProcessorLevel;
    public ushort dwProcessorRevision;
}

[StructLayout(LayoutKind.Explicit)]
public struct _PROCESSOR_INFO_UNION
{
    [FieldOffset(0)]
    internal uint dwOemId;
    [FieldOffset(0)]
    internal ushort wProcessorArchitecture;
    [FieldOffset(2)]
```

```
    internal ushort wReserved;
}
private void btnHDDDurumu_Click(object sender, EventArgs e)
{
    ulong toplamalan;
    ulong ulasilanbosalan;
    ulong bosalan;
    double alan;

    bool sonuc = GetDiskFreeSpaceEx(txtDiskAdi.Text, out ulasilanbosalan,
        out toplamalan, out bosalan);
    if (!sonuc)
    {
        throw new System.ComponentModel.Win32Exception();
    }

    alan = bosalan / (1024 * 1024);
    lblBosAlan.Text = lblBosAlan.Text + " " + alan.ToString() + " MB";

    alan = toplamalan / (1024 * 1024);
    lblToplamAlan.Text = lblToplamAlan.Text + " " + alan.ToString() + " MB";
}

private void tabPage4_Click(object sender, EventArgs e)
{

}

private void btnAgBaglantisi_Click(object sender, EventArgs e)
{
    NetworkCollection networks = NetworkListManager.
        GetNetworks(NetworkConnectivityLevels.All);
    foreach (Network n in networks)
    {
        TreeNode ana_dugum = new TreeNode();
        ana_dugum.Text = string.Format("Network {0} ({1})",
            treeView1.Nodes.Count, "Bağlantı İsmi= " + n.Name);
        treeView1.Nodes.Add(ana_dugum);

        ana_dugum.Nodes.Add("Tanımlama= " + n.Description);
        ana_dugum.Nodes.Add("Domain türü=" + n.DomainType);
        if (n.IsConnected == true)
        {
            ana_dugum.Nodes.Add("Ağ Bağlantısı Açık");
        }
        else
        {
            ana_dugum.Nodes.Add("Ağ Bağlantısı Kapalı");
        }
        if (n.IsConnectedToInternet == true)
        {
            ana_dugum.Nodes.Add("İnternet Bağlantısı Var");
        }
        else
        {
            ana_dugum.Nodes.Add("İnternet Bağlantısı Yok");
        }
        ana_dugum.Nodes.Add("Network Id=" +
            n.NetworkId.ToString());
        ana_dugum.Nodes.Add("Kategori=" +
            n.Category.ToString());
        ana_dugum.Nodes.Add("Oluşturulma Zamanı=" +
```

```
            n.CreatedTime.ToString());
        ana_dugum.Nodes.Add("Bağlantı Zamanı=" +
            n.ConnectedTime.ToString());
        ana_dugum.Nodes.Add("Bağlantı Protokolleri=" +
            n.Connectivity.ToString());
    }
    treeView1.ExpandAll();
}

private void btnBatarya_Click(object sender, EventArgs e)
{
    lblBesleme.Text = PowerManager.PowerSource.ToString();

    BatteryState batteryState = PowerManager.GetCurrentBatteryState();
    progressBar1.Value = PowerManager.BatteryLifePercent;
    lblSarjYuzde.Text = PowerManager.BatteryLifePercent.ToString() + "%";
    lblMaksimumSarj.Text = batteryState.MaxCharge + " mWh";
    lblGecerliSarj.Text = batteryState.CurrentCharge + " mWh";
}

private void btnRamDurumu_Click(object sender, EventArgs e)
{
    MEMORYSTATUSEX hafıza = new MEMORYSTATUSEX();
    hafıza.dwLength = 64;
    //Windows 7 öncesi versiyonlar için
    //GlobalMemoryStatusEx(hafıza);
    GlobalMemoryStatusEx(ref hafıza);

    textBox1.Text = textBox1.Text + "Kullanılan bellek yüzdesi= " +
        (hafıza.dwMemoryLoad) + "\r\n";
    textBox1.Text = textBox1.Text + "Toplam bellek miktarı= " +
        (hafıza.ullTotalPhys / (1024 * 1024)) + " mb \r\n";
    textBox1.Text = textBox1.Text + "Boş bellek miktarı= " +
        (hafıza.ullAvailPhys / (1024 * 1024)) + " mb \r\n";
    textBox1.Text = textBox1.Text + "Toplam page file miktarı= " +
        (hafıza.ullTotalPageFile / (1024 * 1024)) + " mb \r\n";
    textBox1.Text = textBox1.Text + "Boş page file miktarı= " +
        (hafıza.ullAvailPageFile / (1024 * 1024)) + " mb \r\n";
    textBox1.Text = textBox1.Text + "Toplam sanal bellek miktarı= " +
        (hafıza.ullTotalVirtual / (1024 * 1024)) + " mb \r\n";
    textBox1.Text = textBox1.Text + "Boş sanal bellek miktarı= " +
        (hafıza.ullAvailVirtual / (1024 * 1024)) + " mb";
}

private void btnSistemBilgisi_Click(object sender, EventArgs e)
{
    SYSTEM_INFO sysinfo = new SYSTEM_INFO();
    GetSystemInfo(ref sysinfo);
    string islemciTipi = "";
    switch (sysinfo.uProcessorInfo.wProcessorArchitecture)
    {
        case 9:
            islemciTipi = "İşlemci Tipi = AMDx64";
            break;
        case 6:
            islemciTipi = "İşlemci Tipi = Itaniumx64";
            break;
        case 0:
            islemciTipi = "İşlemci Tipi = Intelx86";
            break;
        default:
            islemciTipi = "İşlemci Tipi bilinmiyor";
```

```
                    break;
            }
            MessageBox.Show(islemciTipi + "\nİşlemci Sayısı = " +
sysinfo.dwNumberOfProcessors.ToString() + "\nCache Bellek Miktarı = " +
                sysinfo.dwPageSize.ToString(),"Sistem
Bilgisi",MessageBoxButtons.OK,MessageBoxIcon.Information);
        }

        private void btnMesajVer_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Bu mesaj kutusu Windows API kullanımına basit bir örnektir!","Mesaj
Kutusu",MessageBoxButtons.OK,MessageBoxIcon.Information);
        }

        private void btnGorevler_Click(object sender, EventArgs e)
        {
            foreach (Process gorev in Process.GetProcesses())
            {
                listBox1.Items.Add(gorev.ProcessName);
            }
        }
        public bool ProcessCDTray(bool open)
        {
            int ret = 0;
            //do a switch of the value passed
            switch (open)
            {
                case true: //true = open the cd ret = mciSendString("set cdaudio door open", null, 0,
IntPtr.Zero); return true; break;
                case false: //false = close the tray ret = mciSendString("set cdaudio door closed",
null, 0, IntPtr.Zero); return true; break;
                default: ret = mciSendString("set cdaudio door open", null, 0, IntPtr.Zero); return
true; break;
            }
        }
        private void btnDVDSurucuAc_Click(object sender, EventArgs e)
        {
            ProcessCDTray(true);
        }

        private void btnDVDSurucuKapat_Click(object sender, EventArgs e)
        {
            ProcessCDTray(false);
        }

        private void tabPage1_Click(object sender, EventArgs e)
        {

        }


    }
}
```