Get a 32×32 icon for a given file

```
//Microsoft Reciprocal License (Ms-RL)
//http://bmcommons.codeplex.com/license
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Drawing;
using System.Text;

namespace BlueMirror.Commons
{

public static class Win32
{

public const uint SHGFI_ICON = 0x100;
public const uint SHGFI_DISPLAYNAME = 0x200;
public const uint SHGFI_TYPENAME = 0x400;
public const uint SHGFI_ATTRIBUTES = 0x800;
public const uint SHGFI_ICONLOCATION = 0x1000;
public const uint SHGFI_EXETYPE = 0x2000;
public const uint SHGFI_SYSICONINDEX = 0x4000;
public const uint SHGFI_LINKOVERLAY = 0x8000;
public const uint SHGFI_SELECTED = 0x10000;
public const uint SHGFI_LARGEICON = 0x0;
public const uint SHGFI_SMALLICON = 0x1;
public const uint SHGFI_OPENICON = 0x2;
public const uint SHGFI_SHELLICONSIZE = 0x4;
public const uint SHGFI_PIDL = 0x8;
public const uint SHGFI_USEFILEATTRIBUTES = 0x10;

private const uint FILE_ATTRIBUTE_NORMAL = 0x80;
private const uint FILE_ATTRIBUTE_DIRECTORY = 0x10;
```

```
[DllImport(“comctl32.dll”)]
private static extern int ImageList_GetImageCount(int himl);

[DllImport(“comctl32.dll”)]
private static extern int ImageList_GetIcon(int HIMAGELIST,
int ImgIndex, int hbmMask);

[DllImport(“shell32.dll”)]
private static extern int SHGetFileInfo(string pszPath, uint
dwFileAttributes, ref SHFILEINFO psfi, int cbfileInfo, uint
uFlags);

private struct SHFILEINFO
{
public IntPtr hIcon;
public int iIcon;
public int dwAttributes;
public string szDisplayName;
public string szTypeName;
}

public enum FileIconSize
{
Small, // 16×16 pixels
Large // 32×32 pixels
}

// get a 32×32 icon for a given file

public static Image GetFileIconAsImage(string fullpath) {
return GetFileIconAsImage(fullpath, FileIconSize.Large);
}

public static Image GetFileIconAsImage(string fullpath,
FileIconSize size) {
SHFILEINFO info = new SHFILEINFO();

uint flags = SHGFI_USEFILEATTRIBUTES | SHGFI_ICON;
if (size == FileIconSize.Small) {
```

```
flags |= SHGFI_SMALLICON;
}

int retval = SHGetFileInfo(fullpath, FILE_ATTRIBUTE_NORMAL,
ref info, System.Runtime.InteropServices.Marshal.SizeOf(info),
flags);
if (retval == 0) {
return null; // error occured
}

System.Drawing.Icon icon =
System.Drawing.Icon.FromHandle(info.hIcon);

//ImageList imglist = new ImageList();
//imglist.ImageSize = icon.Size;
//imglist.Images.Add(icon);

//Image image = imglist.Images[0];
//icon.Dispose();
//return image;
return icon.ToBitmap();
}

public static Icon GetFileIcon(string fullpath, FileIconSize
size) {
SHFILEINFO info = new SHFILEINFO();

uint flags = SHGFI_USEFILEATTRIBUTES | SHGFI_ICON;
if (size == FileIconSize.Small) {
flags |= SHGFI_SMALLICON;
}

int retval = SHGetFileInfo(fullpath, FILE_ATTRIBUTE_NORMAL,
ref info, System.Runtime.InteropServices.Marshal.SizeOf(info),
flags);
if (retval == 0) {
return null; // error occured
}
```

```
System.Drawing.Icon icon =
System.Drawing.Icon.FromHandle(info.hIcon);
return icon;
}

public static Icon GetFolderIcon(string fullPath, FileIconSize
size) {
SHFILEINFO info = new SHFILEINFO();

uint flags = SHGFI_USEFILEATTRIBUTES | SHGFI_ICON;
if (size == FileIconSize.Small) {
flags |= SHGFI_SMALLICON;
}

int retval = SHGetFileInfo(fullPath, FILE_ATTRIBUTE_DIRECTORY,
ref info, System.Runtime.InteropServices.Marshal.SizeOf(info),
flags);
if (retval == 0) {
return null; // error occured
}

System.Drawing.Icon icon =
System.Drawing.Icon.FromHandle(info.hIcon);
return icon;
}

public static int GetFileIconIndex(string fullpath,
FileIconSize size) {
SHFILEINFO info = new SHFILEINFO();

uint flags = SHGFI_USEFILEATTRIBUTES | SHGFI_SYSICONINDEX;
if (size == FileIconSize.Small) {
flags |= SHGFI_SMALLICON;
}

int retval = SHGetFileInfo(fullpath, FILE_ATTRIBUTE_NORMAL,
ref info, System.Runtime.InteropServices.Marshal.SizeOf(info),
flags);
if (retval == 0) {
```

```
return -1; // error
}

return info.iIcon;
}

public static int GetFolderIconIndex(string fullpath,
FileIconSize size) {
SHFILEINFO info = new SHFILEINFO();

uint flags = SHGFI_USEFILEATTRIBUTES | SHGFI_SYSICONINDEX;
if (size == FileIconSize.Small) {
flags |= SHGFI_SMALLICON;
}

int retval = SHGetFileInfo(fullpath, FILE_ATTRIBUTE_DIRECTORY,
ref info, System.Runtime.InteropServices.Marshal.SizeOf(info),
flags);
if (retval == 0) {
return -1; // error
}

return info.iIcon;
}

public static void UpdateSystemImageList(ImageList imageList,
FileIconSize size, bool isSelected, Image deletedImage) {
SHFILEINFO info = new SHFILEINFO();
uint flags = SHGFI_SYSICONINDEX;

if (size == FileIconSize.Small)
flags |= SHGFI_SMALLICON;

if (isSelected == true)
flags |= SHGFI_OPENICON;

int imageHandle = SHGetFileInfo("C:", 0, ref info,
System.Runtime.InteropServices.Marshal.SizeOf(info), flags);
int iconCount = ImageList_GetImageCount(imageHandle);
for (int i = imageList.Images.Count; i < iconCount; i++) {
```

IntPtr iconHandle = (IntPtr)ImageList_GetIcon(imageHandle, i, 0); try { if (iconHandle.ToInt64() != 0) { Icon icon = Icon.FromHandle(iconHandle); imageList.Images.Add(icon); icon.Dispose(); DestroyIcon(iconHandle); } } catch { imageList.Images.Add(deletedImage); } } } [DllImport("user32.dll", CharSet = CharSet.Auto)] public extern static bool DestroyIcon(IntPtr handle); [DllImport("user32.dll", CharSet = CharSet.Unicode)] public extern static int SendMessage(IntPtr hWnd, UInt32 Msg, int wParam, int lParam); [DllImport("winmm.dll", EntryPoint = "mciSendStringA")] extern static void mciSendStringA(string lpstrCommand, string lpstrReturnString, long uReturnLength, long hwndCallback); public static void Eject(string driveLetter) { string returnString = ""; mciSendStringA("set cdaudio!" + driveLetter + " door open", returnString, 0, 0); } public static void Close(string driveLetter) { string returnString = ""; mciSendStringA("set cdaudio!" + driveLetter + " door closed", returnString, 0, 0); } [DllImport("user32.dll", SetLastError = true, CharSet = CharSet.Auto)] public extern static uint RegisterWindowMessage(string lpString); [DllImport("user32.dll", CharSet = CharSet.Unicode)] public extern static void SetWindowLong(IntPtr hWnd, int nIndex, int dwNewLong); [DllImport("user32.dll", CharSet = CharSet.Unicode)] public extern static int GetWindowLong(IntPtr hWnd, int nIndex); // Constants and structs defined in DBT.h public const int WM_DEVICECHANGE = 0x0219; public const int DBT_DEVICEARRIVAL = 0x8000; public const int DBT_DEVICEREMOVECOMPLETE = 0x8004; public const int DBT_DEVNODES_CHANGED = 0x0007; public enum DeviceType : int { OEM = 0x00000000, //DBT_DEVTYP_OEM DeviceNode = 0x00000001, //DBT_DEVTYP_DEVNODE Volume = 0x00000002, //DBT_DEVTYP_VOLUME Port = 0x00000003, //DBT_DEVTYP_PORT Net = 0x00000004 //DBT_DEVTYP_NET } public struct BroadcastHeader //_DEV_BROADCAST_HDR { public int Size; //dbch_size public DeviceType Type; //dbch_devicetype public int Reserved; //dbch_reserved } public struct Volume //_DEV_BROADCAST_VOLUME

{ public int Size; //dbcv_size public DeviceType Type; //dbcv_devicetype public int Reserved; //dbcv_reserved public int Mask; //dbcv_unitmask public int Flags; //dbcv_flags } [DllImport("kernel32.dll")] public extern static long GetVolumeInformation(string PathName, StringBuilder VolumeNameBuffer, int VolumeNameSize, ref uint VolumeSerialNumber, ref uint MaximumComponentLength, ref uint FileSystemFlags, StringBuilder FileSystemNameBuffer, int FileSystemNameSize); public static string GetVolumeSerialNumber(string drive) { uint serNum = 0; uint maxCompLen = 0; StringBuilder volLabel = new StringBuilder(256); uint volFlags = 0; StringBuilder fileSystemName = new StringBuilder(256); /* long ret = */ GetVolumeInformation(drive, volLabel, volLabel.Capacity, ref serNum, ref maxCompLen, ref volFlags, fileSystemName, fileSystemName.Capacity); string serialNumberAsString = serNum.ToString("X"); serialNumberAsString.PadLeft(8, '0'); serialNumberAsString = serialNumberAsString.Substring(0, 4) + "-" + serialNumberAsString.Substring(4); return serialNumberAsString; } // List View public const int LVS_EX_DOUBLEBUFFER = 0x10000; public const int LVM_FIRST = 0x1000; public const int LVM_SETITEMSTATE = LVM_FIRST + 43; public const int LVM_SETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 54; public const int LVM_GETEXTENDEDLISTVIEWSTYLE = LVM_FIRST + 55; public const int LVM_SETCOLUMNORDERARRAY = LVM_FIRST + 58; public const int LVM_GETCOLUMNORDERARRAY = LVM_FIRST + 59; public const int LVIF_STATE = 0x0008; public const int LVIS_SELECTED = 0x0002; public const int LVIS_FOCUSED = 0x0001; [StructLayout(LayoutKind.Sequential)] public struct LVITEM { public uint mask; public int iItem; public int iSubItem; public uint state; public uint stateMask; public string pszText; public int cchTextMax; public int iImage; public int lParam; public int iIndent; public int iGroupId; public uint cColumns; public uint puColumns; } [DllImport("user32.dll", CharSet = CharSet.Unicode)] public extern static int SendMessage(IntPtr hWnd, UInt32 Msg, int wParam, ref LVITEM lvItem); // Tree View Styles public const

int TV_FIRST = 0x1100; public const int TVM_SETEXTENDEDSTYLE = TV_FIRST + 44; public const int TVM_GETEXTENDEDSTYLE = TV_FIRST + 45; public const int TVM_SETAUTOSCROLLINFO = TV_FIRST + 59; public const int TVS_NOHSCROLL = 0x8000; public const int TVS_EX_MULTISELECT = 0x0002; public const int TVS_EX_DOUBLEBUFFER = 0x0004; public const int TVS_EX_AUTOHSCROLL = 0x0020; public const int TVS_EX_FADEINOUTEXPANDOS = 0x0040; public const int GWL_STYLE = -16; [DllImport("uxtheme.dll", CharSet = CharSet.Unicode)] public extern static int SetWindowTheme(IntPtr hWnd, string pszSubAppName, string pszSubIdList); } } [/csharp]