



CODE
PROJECT®
For those who code

[articles](#)[Q&A](#)[forums](#)[lounge](#) 

A class to load and use file associated icons

Vitaly Zayko, 24 Apr 2008



4.83 (13 votes)

Rate:



An article about using SHGetFileInfo in C# (a simple class and demo are included).



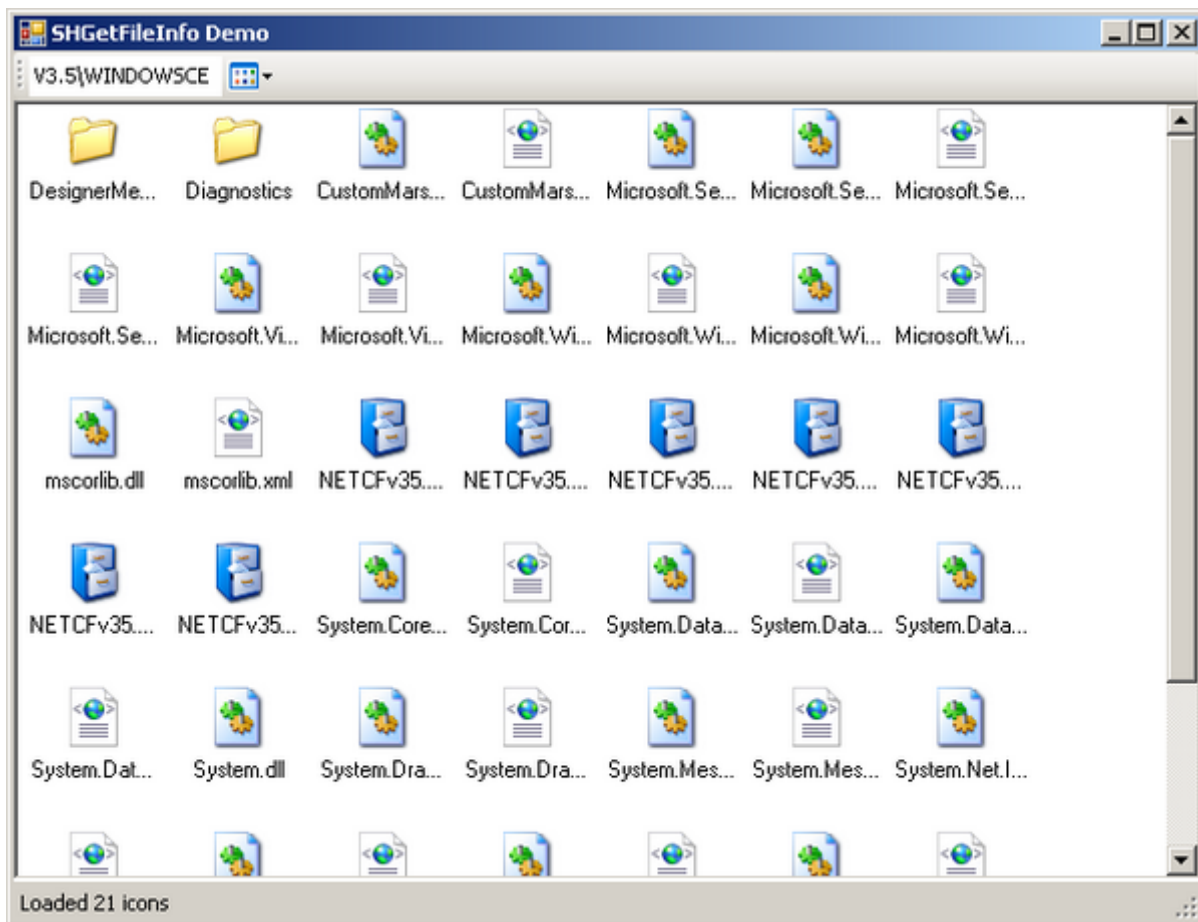
Is your email address OK? You are signed up for our newsletters but your email address is either unconfirmed, or has not been reconfirmed in a long time. Please [click here to have a confirmation email sent](#) so we can confirm your email address and start sending you newsletters again. Alternatively, you can [update your subscriptions](#).



[Download demo - 7.04 KB](#)



[Download sources \(VS2008\) - 13.6 KB](#)



Introduction

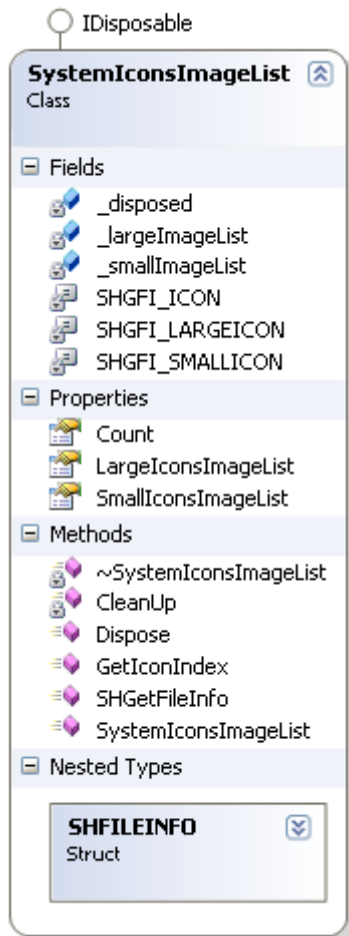
Everyone knows the classical linkage of the two members of Windows Common Controls – **ListView/ImageList** or **TreeView/ImageList**. The left part represents data in several formats, and the right provides the list of images for a better look. In almost all cases, it is your or your designer's job to create and load images that you need. But, what if you need to implement an app with the Windows Explorer interface which should show disk folders, files, shortcuts etc. with the same icons as Explorer does? There is a function **System.Drawing.Icon.ExtractAssociatedIcon** which returns an icon associated with a given file. This is the simplest way, but unfortunately, there is no overloaded version where you can specify the size of the icons you want to get – it always returns 32x32 icons (**Note**: MSDN doesn't explain what size will be returned, so my assumption is it would be the system defined size for large icons, which can be checked by **SystemInformation.IconSize**). Alternatively, you can use a special Windows API function and .NET PInvoke to bring it into your code. To make it reusable, I decided to create a class to utilize this functionality.

About the SHGetFileInfo function

This function is part of the system shell, and is available in every Microsoft OS beginning from Windows 95 and Windows NT 4.0, and can give you a lot of information about objects in the file system, such as files, folders etc. A complete description can be obtained from [MSDN](#). But since we don't need all this functionality, let's focus on getting the file icons for our app.

Using the code

As I said, I created a class that you can find in the archive on the top of this article – **SystemIconsImageList**. This class is derived from **Object**, not from **ImageList** or **Component**. Thus, you have to declare, create, and initialize it manually in the code-behind of your form. I will get back to you later with an explanation of why I did it this way. Let's take a close look at it:



First of all, we need to perform Platform Invoke for the **SHGetFileInfo** function:

[Hide](#) [Copy Code](#)

```
[DllImport("shell32.dll")]
public static extern IntPtr SHGetFileInfo(string pszPath, uint dwFileAttributes,
    ref SHFILEINFO psfi, uint cbSizeFileInfo, uint uFlags);
```

and predefine some system constants and structures that this function uses:

[Hide](#) [Copy Code](#)

```
private const uint SHGFI_ICON = 0x100;
private const uint SHGFI_LARGEICON = 0x0;
private const uint SHGFI_SMALLICON = 0x1;

[StructLayout(LayoutKind.Sequential)]
public struct SHFILEINFO
{
    public IntPtr hIcon;
    public IntPtr iIcon;
    public uint dwAttributes;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
    public string szDisplayName;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
    public string szTypeName;
};
```

Also, this class contains two **System.Windows.Forms.ImageList** controls to store system icons: one for small and one for large icons. Although **ImageList** is a managed control, it holds unmanaged resources – icons or images. That's the reason why **ImageList** implements the **IDisposable** interface which enforces us to implement **IDisposable** for our class as well. Now, let's see how to use this class. First of all, let's declare and create an instance:

[Hide](#) [Copy Code](#)

```
private SystemIconsImageList sysIcons = new SystemIconsImageList();
```

Basically, this constructor just creates **ImageList** objects, and initializes their **ImageSize** properties to the current system values:

[Hide](#) [Copy Code](#)

```
_smallImageList.ColorDepth = ColorDepth.Depth32Bit;
_smallImageList.ImageSize = SystemInformation.SmallIconSize;
_largeImageList.ColorDepth = ColorDepth.Depth32Bit;
_largeImageList.ImageSize = SystemInformation.IconSize;
```

Now, we can attach properties that represent the actual **ImageList** objects to the corresponding **ListView** (or **TreeView**) properties in our form's code-behind:

[Hide](#) [Copy Code](#)

```
listView1.SmallImageList = sysIcons.SmallIconsImageList;
listView1.LargeImageList = sysIcons.LargeIconsImageList;
```

Beside that, this class has a function which loads icons and returns their indexes, which we need to specify in a **ListViewItem.ImageIndex**:

[Hide](#) [Copy Code](#)

```
public int GetIconIndex(string FileName);
```

It has only one input parameter – **FileName**, which is the full path to a folder or file, including its name, used to get the icon associated with it. This file or folder should exist; otherwise, an exception will be raised.

The first trick is to not load all the registered icons for all the file types at ones, but only when we need to show a particular icon. The second is to not load and store the same icons more than once; otherwise, we will have a bunch of duplicates, for example, folder icons, and waste memory for nothing. To make this possible, we will use the file extension as the icon index and check if we already have an icon for this file type or not.

But, wait a minute! This works only for non-executable files, but what about folders, executables which have their own icons with the same ".exe" at the end, shortcuts which should be represented by parent icons, and files without any extensions? We have to take care of all these cases, but differently. For example, the best way for executables and shortcuts probably is to use their names as icon indexes (to reduce memory usage, I decided to use just the name without the full path); for folders and files with no extension, we are going to use GUIDs as a unique key since they should be represented by an icon each.

[Hide](#) [Copy Code](#)

```
FileInfo info = new FileInfo(FileName);
string ext = info.Extension;
if (String.IsNullOrEmpty(ext))
{
    if ((info.Attributes & FileAttributes.Directory) != 0)
        ext = "5EEB255733234c4dBE9A128E896A1E";
        // for directories
    else
        ext = "F9EB930C78D2477c80A51945D505E9C4";
        // for files without extension
}
else
    if (ext.Equals(".exe", StringComparison.InvariantCultureIgnoreCase) ||
        ext.Equals(".lnk", StringComparison.InvariantCultureIgnoreCase))
        ext = info.Name;
```

Then, we just need to check if we have an icon for this file type, or we have to load a new one and add it the to internal **ImageList**:

[Hide](#) [Copy Code](#)

```
if (_smallImageList.Images.ContainsKey(ext))
{
    return _smallImageList.Images.IndexOfKey(ext);
}
else
{
    SHGetFileInfo(FileName, 0, ref shinfo,
        (uint)Marshal.SizeOf(shinfo), SHGFI_ICON | SHGFI_SMALLICON);
    Icon smallIcon;
    try
    {
```

```
        smallIcon = Icon.FromHandle(shinfo.hIcon);
    }
    catch (ArgumentException ex)
    {
        throw new ArgumentException(String.Format("File \"{0}\" does" +
            " not exist!", FileName), ex);
    }
    _smallImageList.Images.Add(ext, smallIcon);

    SHGetFileInfo(FileName, 0, ref shinfo,
        (uint)Marshal.SizeOf(shinfo), SHGFI_ICON | SHGFI_LARGEICON);
    Icon largeIcon = Icon.FromHandle(shinfo.hIcon);
    _largeImageList.Images.Add(ext, largeIcon);

    return _smallImageList.Images.Count - 1;
}
```

...and we are done!

Why didn't we implement this functionality based on **Component**? In this case, we will be able to put these into the Visual Studio Toolbox and set the required properties at design time.

First of all, Microsoft made **ImageList** as **sealed**, so no child components could be derived from it. Plus, I wanted to load both small and large icons at the same time, so I needed two **ImageList** objects inside. Since my component cannot be derived from **ImageList**, I won't be able to assign the **SmallImageList** and **LargeImageList** properties of **ListView/TreeView** at design time, so some small manipulations in code were required any way.

And one last thing: I feel much comfortable creating non-visual components manually in the code-behind than looking for them in the Visual Studio Toolbox :)

Enjoy!

History

- 1 May 2008 - Minor update.
- 24 April 2008 - Initial version.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)


Share



About the Author



Vitaly Zayko

Software Developer (Senior)
Russian Federation 

More than 15 years in the industry.
Delphi, C# (Win/WebForms), MS SQL

You may also be interested in...

[Public, Private, and Hybrid Cloud: What's the difference?](#)

[SAPrefs - Netscape-like Preferences Dialog](#)

[System File Association](#)

[Generate and add keyword variations using AdWords API](#)

[Get Registered File Types and Their Associated Icons in C#](#)

[Window Tabs \(WndTabs\) Add-In for DevStudio](#)

Comments and Discussions

Add a Comment or Question



Search Comments



First Prev Next

Get Icon Location from SHGetFileInfo

fer_cyberlinklabs 18-Dec-08 7:20

Nice One, but if you want to speed up your app, try this [modified]

Paw Jershaug 16-Oct-08 4:52

Wow!

hannahb 5-Oct-08 4:07

StateImageList

Jan.Seiffert 10-Jun-08 20:13

Non existing files

Member 1920274 25-Apr-08 19:53

Re: Non existing files

Vitaly Zayko 25-Apr-08 21:59

Re: Non existing files

Leung Yat Chun 7-Nov-09 23:43

Err..



The_Mega_ZZTer 25-Apr-08 11:47

Re: Err..

Vitaly Zayko 25-Apr-08 13:31

[Refresh](#)

1

 General  News  Suggestion  Question  Bug  Answer  Joke  Praise  Rant  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)
Web02 | 2.8.170927.1 | Last Updated 24 Apr 2008

 选择语言 | ▼
Layout: [fixed](#) | [fluid](#)

Article Copyright 2008 by Vitaly Zayko
Everything else Copyright © [CodeProject](#), 1999-2017