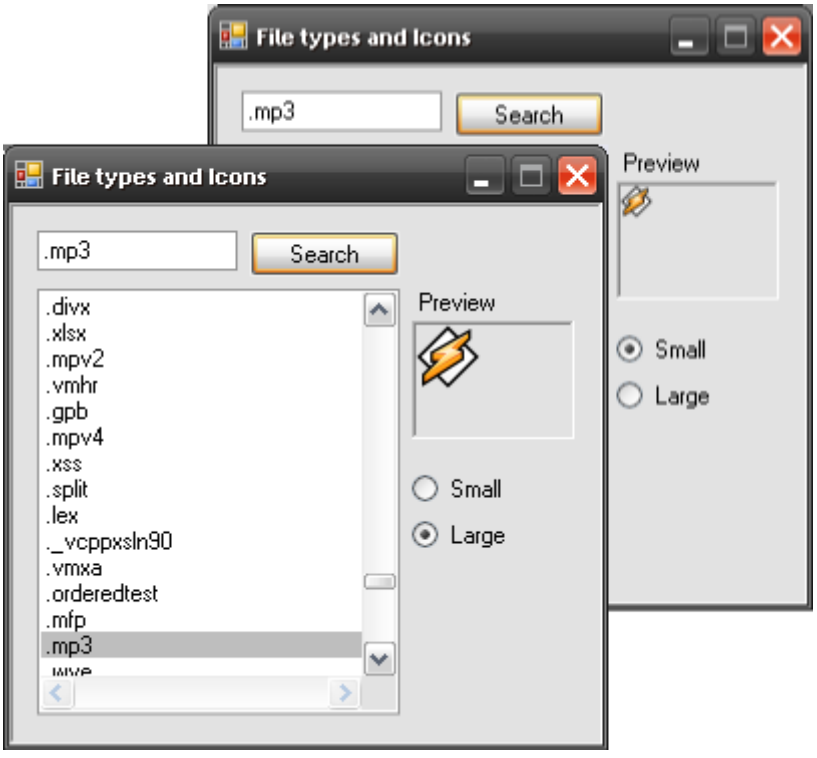
Get Registered File Types and Their Associated Icons in C#

Man Vuong, 19 Oct 2009

Get registered file types and their associated icons by reading Registry in C#

Download source - 14.11 KB



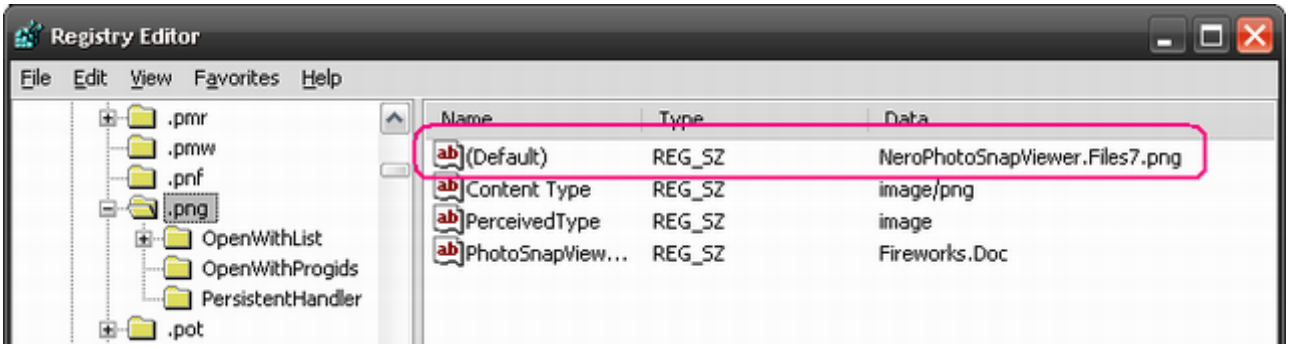
Introduction

How can we get the icon of a file if it does not exist? If we only know its name or even its extension, how do we get the icon associated with it? This is a simple project of mine to get the registered file types and the associated icons by reading from Windows Registry. The idea belongs to Aaron Young in his article "Listing the Registered File Extensions and Their Associated Icons" at CodeGuru, but the original source code was in Visual Basic. I know a little bit of VB, so I tried to rewrite it in C#. Thanks very much to Aeron Young!

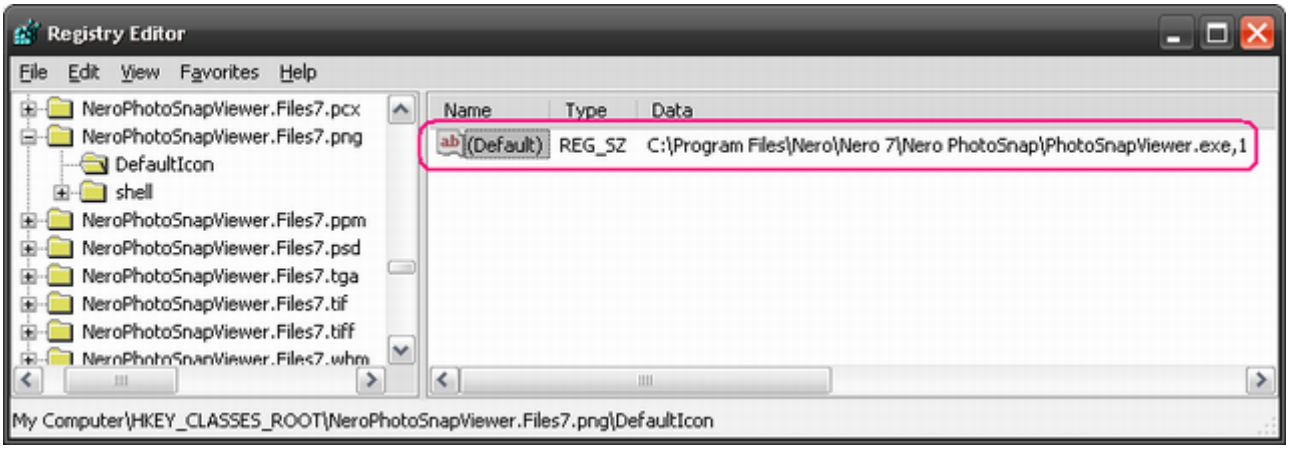
Background

Because this method reads information from the Windows registry, we need to know some basics.

Firstly, where are the registered file types? They are contained in the key `HKEY_CLASSES_ROOT` . Now, you launch the Regedit (some way can be: Start > Run > regedit), and go to that key. You may find a lot of sub keys like this: " *.zip* ", " *.mp3* ", etc. Those are the file types we need. Pay attention to the value named empty (Default) as in the picture below:



The value (Default) is not a file path, you can easily find out that the data of default value (here: *NeroPhotoSnapPreview.File7.png*) is the name of another sub key in `HKEY_CLASSES_ROOT` . Following is what we can find:



Now, what do you see for the value of `DefaultIcon`? That is the path to the file that contains the icon we need! Our tasks now are getting that file path and retrieving it to Windows Form.

Secondly, we need to interact with the Registry. .NET supports class `RegistryKey` in `Microsoft.Win32` namespace and makes it easy for us to open a registry key and get its values as well as its sub keys. Walking a registry tree is the same as walking a file/folder tree structure, it is very easy!

Thirdly, how to retrieve an icon from a file ? There is a little bit of difficulty here, we need to use an API function: `ExtractIcon`, but don't worry! All our works are redefining it in C#, and passing the correct parameters for it. Another choice is the function of `ExtractIconEx`, but we will consider the `ExtractIcon` first.

Using the Code

I packed all necessary methods and API functions in the class `RegisteredFileType`.

The ExtractIcon Method

Here is the `ExtractIcon` I redefined in C# by referencing *www.pinvoke.net*:

Hide Copy Code

```
[DllImport("shell32.dll", EntryPoint = "ExtractIconA",
    CharSet = CharSet.Ansi, SetLastError = true, ExactSpelling = true)]
private static extern IntPtr ExtractIcon
    (int hInst, string lpszExeFileName, int nIconIndex);
```

Be sure that you declared to use the `System.Runtime.InteropServices` namespace.

The GetFileTypeAndIcon Method

This is one of the core methods which gets the registered file types and icons from Windows Registry:

Hide Shrink Copy Code

```
/// <summary>
Gets registered file types and their associated icon in the system.
/// </summary>
/// <returns>Returns a hash table which contains the file extension as keys,
/// the icon file and param as values.</returns>
public static Hashtable GetFileTypeAndIcon()
{
    try
    {
    // Create a registry key object to represent the
    // HKEY_CLASSES_ROOT registry section
    RegistryKey rkRoot = Registry.ClassesRoot;
    //Gets all sub keys' names.
    string[] keyNames = rkRoot.GetSubKeyNames();
    Hashtable iconsInfo = new Hashtable();
    //Find the file icon.
    foreach (string keyName in keyNames)
    {
        if (String.IsNullOrEmpty(keyName))
            continue;
        int indexOfPoint = keyName.IndexOf(".");

             //If this key is not a file extension, .zip), skip it.
        if (indexOfPoint != 0)
            continue;
        RegistryKey rkFileType = rkRoot.OpenSubKey(keyName);
        if (rkFileType == null)
            continue;
        //Gets the default value of this key that
        //contains the information of file type.
        object defaultValue = rkFileType.GetValue("");
        if (defaultValue == null)
            continue;
          //Go to the key that specifies the default icon
             //associates with this file type.
        string defaultIcon = defaultValue.ToString() + "\\DefaultIcon";
        RegistryKey rkFileIcon = rkRoot.OpenSubKey(defaultIcon);
        if (rkFileIcon != null)
        {
            //Get the file contains the icon and the index of the icon in that file.
            object value = rkFileIcon.GetValue("");
            if (value != null)
            {
                //Clear all unnecessary " sign in the string to avoid error.
                string fileParam = value.ToString().Replace("\"", "");
                iconsInfo.Add(keyName, fileParam);
                 }
                 rkFileIcon.Close();
             }
             rkFileType.Close();
        }
        rkRoot.Close();
        return iconsInfo;
    }
    catch (Exception exc)
    {
        throw exc;
```

```
    }
}
```

Note that this function returns a `Hashtable`, you can store information in an array of `string`, but by using `Hashtable` we will more conveniently get the icon file name from its file extension. Remember to use `Microsoft.Win32` namespace which contains class `RegistryKey`.

First of all, we need to get `HKEY_CLASSES_ROOT`:

Hide Copy Code

```
RegistryKey rkRoot = Registry.ClassesRoot;
```

Next, we get all sub keys of it to get all file types:

Hide Copy Code

```
string[] keyNames = rkRoot.GetSubKeyNames();
```

To prevent the sub key that is not a file type (eg: `ACLFile`), we detect the dot sign:

Hide Copy Code

```
int indexOfPoint = keyName.IndexOf(".");

string defaultIcon = defaultValue.ToString() + "\\DefaultIcon";
RegistryKey rkFileIcon = rkRoot.OpenSubKey(defaultIcon);
```

This will lead us to the key that contains the icon file, the last one is adding this file path and the index of the icon to the hash table as a value and its associating filetype as the key:

Hide Copy Code

```
iconsInfo.Add(keyName, fileParam);
```

The EmbeddedIconInfo Struct

Note that the information got by the above method is just draw data. We need a structure to store the information associated with the icon file. Therefore, we create the `EmbeddedIconInfo` to pack the necessary information which are the name of file containing the icon and the index of the icon:

Hide Copy Code

```
/// <summary>
/// Structure that encapsulates basic information of icon embedded in a file.
/// </summary>
public struct EmbeddedIconInfo
{
    public string FileName;
    public int IconIndex;
}
```

The getEmbeddedIconInfo Method

This method does some logical operations to obtain an instance of `EmbeddedIconInfo` structure:

Hide Shrink ▲ Copy Code

```
/// <summary>
/// Parses the parameters string to the structure of EmbeddedIconInfo.
/// </summary>
/// <param name="fileAndParam">The params string, such as ex:
///     "C:\\Program Files\\NetMeeting\\conf.exe,1".</param>
protected static EmbeddedIconInfo getEmbeddedIconInfo(string fileAndParam)
{
    EmbeddedIconInfo embeddedIcon = new EmbeddedIconInfo();

    if (String.IsNullOrEmpty(fileAndParam))
        return embeddedIcon;
```

```
    //Use to store the file contains icon.
    string fileName = String.Empty;

    //The index of the icon in the file.
    int iconIndex = 0;
    string iconIndexString = String.Empty;

    int commaIndex = fileAndParam.IndexOf(",");
    //if fileAndParam is some thing likes this:
         //"C:\\Program Files\\NetMeeting\\conf.exe,1".
    if (commaIndex > 0)
    {
        fileName = fileAndParam.Substring(0, commaIndex);
        iconIndexString = fileAndParam.Substring(commaIndex + 1);
    }
    else
        fileName = fileAndParam;

    if (!String.IsNullOrEmpty(iconIndexString))
    {
        //Get the index of icon.
        iconIndex = int.Parse(iconIndexString);
        if (iconIndex < 0)
            iconIndex = 0;  //To avoid the invalid index.
    }

    embeddedIcon.FileName = fileName;
    embeddedIcon.IconIndex = iconIndex;

    return embeddedIcon;
}
```

The ExtractIconFromFile Method

Actually, the method `RegisteredFileType.ExtractIconFromFile()` calls the Windows API `ExtractIcon` in the background:

Hide Copy Code

```
/// <summary>
/// Extract the icon from file.
/// <param name="fileAndParam">The params string, such as ex:
///    "C:\\Program Files\\NetMeeting\\conf.exe,1".</param>
/// <returns>This method always returns the large size of the icon
///    (may be 32x32 px).</returns>
public static Icon ExtractIconFromFile (string fileAndParam)
{
    try
    {
        EmbeddedIconInfo embeddedIcon = getEmbeddedIconInfo(fileAndParam);

        //Gets the handle of the icon.
        IntPtr lIcon = ExtractIcon(0, embeddedIcon.FileName,
                    embeddedIcon.IconIndex);

        //Gets the real icon.
        return Icon.FromHandle(lIcon);
    }
    catch (Exception exc)
    {
        throw exc;
    }
}
```

Retrieving To Form

This is the most exciting work to see our achievement!

I created the form `IconForm` with the important member, `icons Hashtable`:

Hide Copy Code

```
/// <summary>
/// Used for containing file types and their icons information.
/// </summary>
private Hashtable iconsInfo;
```

The next step, we simply get all file types and their icons:

Hide Copy Code

```
this.iconsInfo = RegisteredFileType.GetFileTypeAndIcon();
```

It's time to show the icons:

Hide Shrink Copy Code

```
/// <summary>
/// Shows the icon associates with a specific file type.
/// </summary>
/// <param name="fileType">The type of file (or file extension).</param>
private void showIcon(string fileType)
{
    try
    {
    string fileAndParam = (this.icons[fileType]).ToString();

    if (String.IsNullOrEmpty(fileAndParam))
        return;

                Icon icon = null;

                icon = RegisteredFileType.ExtractIconFromFile(fileAndParam);

                //The icon cannot be zero.
                if (icon != null)
                {
                    //Draw the icon to the picture box.
                    this.pbIconView.Image = icon.ToBitmap();
                }
                else //if the icon is invalid, show an error image.
                    this.pbIconView.Image = this.pbIconView.ErrorImage;
    }
    catch (Exception exc)
    {
        throw exc;
    }
}
```

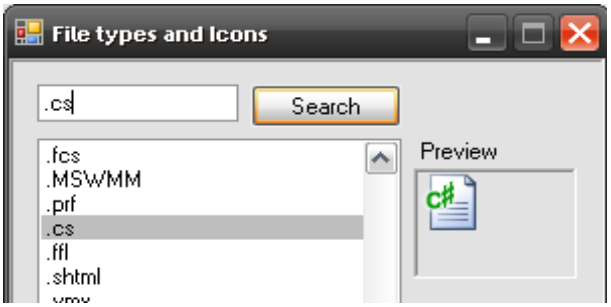
To make it more logical, I add another function to do business operations:

Hide Copy Code

```
/// <summary>
/// Validates the input data and renders the image.
/// </summary>
private void renderImage()
{
    try
    {
        if (this.lbxFileType.Items.Count <= 0
                || this.lbxFileType.SelectedItem == null)
            return;
        string fileType = this.lbxFileType.SelectedItem.ToString();

        this.showIcon(fileType);
    }
    catch (Exception exc)
    {
        throw exc;
    }
}
```

Here is the result:



A New Problem...

Recently, I have received an interesting question: How can we get the 16x16 px icons from these icons? If we use the `ExtractIcon` function in this way, it will always return the large size of the icon.

There are many solutions for this. One of the easiest ways is use an `ImageList` object to manage the icons and change the image's size by setting up the property `ImageSize`.

However, the quality of the images is not very good. This trouble was posted in this site: Get full quality 16 x 16 icon using Icon.ExtractAssociatedIcon and ImageList [^]. If you go to this site, you also find out a suggested solution to get full quality small icon. Yet, it seems to me that the suggestion is rather complicated.

In this section, I give you another solution to solve the problem. Instead of using the function `ExtractIcon`, we can use the Windows API `ExtractIconEx`. This method is more powerful than `ExtractIcon` because it creates an array of handles to large or small icons extracted from the specified executable file, DLL, or icon file.

This is the `ExtractIconEx` redefined in C#:

Hide Copy Code

```
[DllImport("shell32.dll", CharSet = CharSet.Auto)]
private static extern uint ExtractIconEx
    (string szFileName, int nIconIndex,
    IntPtr[] phiconLarge, IntPtr[] phiconSmall, uint nIcons);
```

We must destroy all icons extracted by `ExtractIconEx` by using the `DestroyIcon` function:

Hide Copy Code

```
[DllImport("user32.dll", EntryPoint = "DestroyIcon", SetLastError = true)]
private static unsafe extern int DestroyIcon (IntPtr hIcon);
```

When I find the definition of ExtractIconEx [^] in *www.pinvoke.net*, I also see a good example in the site. That's why I use it in this application. Based on the example, I add an overloaded version of the method `ExtractIconFromFile` in class `RegisteredFileType`:

Hide Shrink Copy Code

```
/// <summary>
/// Extract the icon from file.
/// </summary>
/// <param name="fileAndParam">The params string, such as ex:
///    "C:\\Program Files\\NetMeeting\\conf.exe,1".</param>
/// <param name="isLarge">Determines the returned icon is a large
///    (may be 32x32 px) or small icon (16x16 px).</param>
public static Icon ExtractIconFromFile(string fileAndParam, bool isLarge)
{
    unsafe
    {
        uint readIconCount = 0;
        IntPtr[] hDummy = new IntPtr[1] { IntPtr.Zero };
        IntPtr[] hIconEx = new IntPtr[1] { IntPtr.Zero };

        try
        {
            EmbeddedIconInfo embeddedIcon =
                getEmbeddedIconInfo(fileAndParam);
```

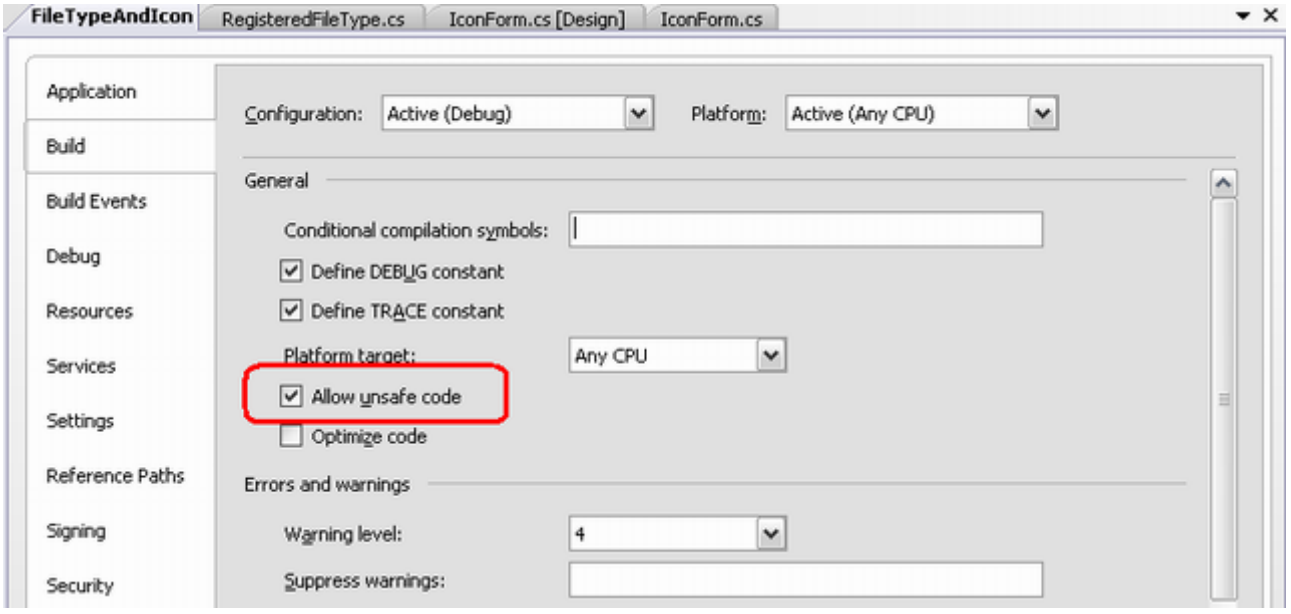
```
            if (isLarge)
                readIconCount = ExtractIconEx
                (embeddedIcon.FileName, 0, hIconEx, hDummy, 1);
            else
                readIconCount = ExtractIconEx
                (embeddedIcon.FileName, 0, hDummy, hIconEx, 1);

            if (readIconCount > 0 && hIconEx[0] != IntPtr.Zero)
            {
                //Get first icon.
                Icon extractedIcon =
                (Icon)Icon.FromHandle(hIconEx[0]).Clone();

                return extractedIcon;
            }
            else //No icon read.
                return null;
        catch (Exception exc)
        {
            //Extract icon error.
            throw new ApplicationException
                ("Could not extract icon", exc);
        }
        finally
        {
            //Release resources.
            foreach (IntPtr ptr in hIconEx)
                if (ptr != IntPtr.Zero)
                    DestroyIcon(ptr);

            foreach (IntPtr ptr in hDummy)
                if (ptr != IntPtr.Zero)
                    DestroyIcon(ptr);
        }
    }
}
```

The code is easy to understand. The `unsafe` keyword helps the encapsulated code to run faster. To compile the application with `unsafe` code, we must configure the application that allows unmanaged code. In Visual Studio, right click on the project and go to tab "Build":



Now, we make some changes in `IconForm` to test the new version of the method `ExtractIconFromFile`.

Hide Shrink ▲ Copy Code

```
/// <summary>
/// Shows the icon associates with a specific file type.
/// </summary>
```

```
/// <param name="fileType">The type of file (or file extension).</param>
private void showIcon(string fileType)
{
    try
    {
    string fileAndParam = (this.icons[fileType]).ToString();

    if (String.IsNullOrEmpty(fileAndParam))
        return;

    Icon icon = null;

    bool isLarge = true;

    if (currentSize == ImageSize.Small)
        isLarge = false;

    icon = RegisteredFileType.ExtractIconFromFile(fileAndParam, isLarge);


    //The icon cannot be zero.
    if (icon != null)
    {
        //Draw the icon to the picture box.
        this.pbIconView.Image = icon.ToBitmap();
    }
    else //if the icon is invalid, show an error image.
        this.pbIconView.Image = this.pbIconView.ErrorImage;
    }
    catch (Exception exc)
    {
        throw exc;
    }
}
```

The `ImageSize` is a simple `enum`:

Hide Copy Code

```
public enum ImageSize
{
     /// <summary>
     /// View image in 16x16 px.
     /// </summary>
    Small,

     /// <summary>
     /// View image in 32x32 px.
     /// </summary>
    Large
}
```

You can use a boolean variable instead!

Points of Interest

I had added a simple search feature. It is absolutely easy because all the keys and their values were stored in `Hashtable` icons, so we can quickly access the searched information like accessing an array.

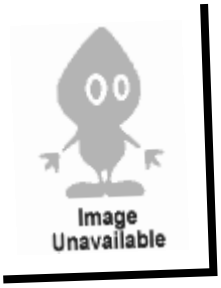
History

- 7th September, 2008: Initial post
- 18th October, 2009: Add new feature of viewing small size icons

License

Share

About the Author



Man Vuong
Software Developer
Vietnam

I am a student of UIT - University of Information Technology, a member of Vietnam National University - Ho Chi Minh City (VNU-HCM). I like programming and computer very much.

You may also be interested in...

System File Association

Generate and add keyword variations using AdWords API

A class to load and use file associated icons

Window Tabs (WndTabs) Add-In for DevStudio

SAPrefs - Netscape-like Preferences Dialog

OLE DB - First steps

Comments and Discussions

Negative icon index is valid
Member 11082711 29-Jan-16 7:08

Bug in ExtractIconFromFile
Member 11082711 29-Jan-16 6:55

Negative index
stormbit 26-Dec-14 22:21

My vote of 5
Aravind Kumar K 26-Oct-12 17:12

My vote of 5
fredatcodeproject 28-Jun-12 1:33

How to follow this tutorial?
RichardBla 21-Jan-12 18:18

Code in vb.net
nsk_saravana 10-Dec-11 19:07

My vote of 5
Willem de Vries 4-Oct-11 18:24

My vote of 4
Ambrus Weisz 16-May-11 3:04

My vote of 5
Member 2754044 21-Oct-10 17:42

My vote of 2
Thesisus 14-Aug-10 23:33

txt files
friendsterjoel 25-Jan-10 20:58

Use "using" with registry keys
mvonballmo 21-Oct-09 14:50

16x16 px icon
nesquik87 17-Oct-09 2:55

Re: 16x16 px icon
Man Vuong 19-Oct-09 23:52

Re: 16x16 px icon
nesquik87 20-Oct-09 5:41

problems with two file types
mirko86 13-Dec-08 2:12

Re: problems with two file types
type5demonlord 16-Feb-09 16:37

Re: problems with two file types
ferocks 18-Sep-09 7:24

Re: problems with two file types
Sameers (theAngrycodeR) 11-Jan-10 17:34

A very simple alternative
Thesisus 14-Aug-10 23:32

Re: A very simple alternative
Mihai Drebot 8-Mar-11 17:50

Re: problems with two file types [solved]
Xilmiki 20-Aug-11 15:25

A few comments
jmertus **29-Sep-08 4:34**

Re: A few comments
kidvn 16-Dec-08 23:00

Refresh

1 2 Next »

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.