

Forum:[Hack Forums](#)

Thread: [CodeDom Compiler | Add & Read Resources | Change Icon + Guide](#)

xQQ - June 17, 2013, 8:56 am

Hello again,

today I felt like sharing my custom CodeDom Compiler class that comes with few good features. It's fully Object Oriented, so it's easy to use and won't need a further edit.

Features:

- Compile C# Sourcecode
- Add an Icon to the compiled .exe
- Add resources and grab them in runtime (explained below)
- Enable Debug-Mode to figure out whats not working (properly)

Source:

```
Code using System.CodeDom.Compiler;
using Microsoft.CSharp;
using System.Windows.Forms;
```

```
/* QCompiler
 *
 * Creates .NET executable from sourcecode
 * Features: Add an Icon, Add Resources, Debug-Mode
 * Written by xQQ
 * Futher Information: https://hackforums.net/showthread.php?tid=3557696
 *
 */

class QCompiler
{
#region Constructors
public QCompiler() : this("", "", "") { }
public QCompiler(string Source) : this(Source, null, null) { }
public QCompiler(string Source, string Icon = "", string Name = "Drop.exe")
{
    Sourcecode = Source;
    OutputName = Name;
    Iconpath = Icon;
}

#endregion

#region Properties
private string _Sourcecode;
public string Sourcecode
{
    get { return _Sourcecode; }
    set { _Sourcecode = value; }
}
public string OutputName = "Drop.exe";
```

```

private string _Iconpath = "";
public string Iconpath
{
    get { return _Iconpath; }
    set { _Iconpath = "/win32icon:" + value; }
}
private bool _AddRes = false;
public bool AddRes
{
    get { return _AddRes; }
    set { _AddRes = value; }
}
private string _Resources;
public string Resources
{
    get { return Resources; }
    set { _Resources = value; }
}
public bool DebugMode = false;
#endregion

#region Methods
public int GeneratExecutable()
{
    if (Sourcecode == "")
    {
        MessageBox.Show("No Source");
        return 1;
    }
    else
    {
        CompilerParameters parameters = new CompilerParameters();
        parameters.GenerateExecutable = true;
        parameters.OutputAssembly = OutputName;
        parameters.ReferencedAssemblies.Add("System.dll");
        //Uncomment next line if you want to compile a form application
        //parameters.ReferencedAssemblies.Add("System.Windows.Forms.dll");
        parameters.CompilerOptions = "/target:winexe /optimize+ /debug- /target:winexe " + Iconpath;
        if (AddRes == true)
        {
            parameters.EmbeddedResources.Add(Resources);
        }
        if (DebugMode == true)
        {
            Clipboard.SetText(Sourcecode);
            parameters.WarningLevel = 3;
            parameters.TreatWarningsAsErrors = true;
        }
        CompilerResults result = new CSharpCodeProvider().CompileAssemblyFromSource(parameters, Sourcecode);
        foreach (CompilerError err in result.Errors)
        {
            MessageBox.Show(err.ToString());
        }
    }
}

```

```

    }
    return 0;
}
}
#endregion
}

```

Usage:

This is just a HowTo use this Compiler. I will not go into detail on how things actually work, if you are interested head to msdn.

The easiest way is just calling the constructor

```
Code QCompiler Compiler = new QCompiler(source);
```

QCompiler Compiler_Icon = new QCompiler(source, null, IconPath); We can also use the second parameter to give the output a name of choice, otherwise it will be "Drop.exe"

```
Code QCompiler Compiler = new QCompiler(source, "FileName.exe", IconPath)
```

Resource

Compiling with CodeDom has one big disadvantage: It has limit for the maximum chars per line. Most people use codedom for their binders and crypters, so if you need to read the bytes of a .exe, they will be stored as only one line. This is unhandy as you need to split this string of bytes first into many lines. Using resources can bypass this. So how we gonna add a resource?

In our Builder we need to create a .resources file first, which later will be included in our stub:

```
Code System.Resources.ResourceWriter ResWriter = new
```

```
System.Resources.ResourceWriter("PathWhereToSave.resources");
```

```
ResWriter.AddResource("ResourceString", "Hello World!");
```

ResWriter.Close(); Obvious you want to change the values. Always remember to close the stream, otherwise you'll get critical errors later.

The next thing we need to do is enabling the matching feature in the compiler by doing the following.

```
Code gen.AddRes = true;
```

gen.Resources = "PathWhereToSave.resources"; This will just include the resource to our generated .exe. But how to access them?

In our stub, we need to add

```
Code System.Resources.ResourceManager ResManager = new
```

```
System.Resources.ResourceManager("PathWhereToSave",
```

System.Reflection.Assembly.GetExecutingAssembly()); Now we can access the resources with our fresh ResManager. To get our "Hello World", we simply call the following:

```
Code string FromRes = (string)ResManager.GetObject("ResourceString");
```

You can show a messagebox to see if it worked.

And for all the coders working on their 1337 FUD Crypters and Binders, if you want to work with bytes, you need to use this for writing

```
Code System.Resources.ResourceWriter ResWriter = new
```

```
System.Resources.ResourceWriter("SomeName.resources");
```

```
ResWriter.AddResource("MyBytes", SomeByteArray);
```

```
ResWriter.Close();
```

And this for reading:

```
Code byte[] TheBytes = (byte[])ResManager.GetObject("MyBytes");
```

Debug-Mode

Well nothing much to explain about this. You can enable it with Code Compiler.DebugMode = true

This

- saves sourcecode to clipboard, so you can have a look over it if something went wrong

- will have the compiler treat warnings as errors

- will look for everything the compiler could complain about, even if it wouldn't affect the output at all.

credits: <http://msdn.microsoft.com/>

So I hope you like it. Always feel free to post questions, feedback and suggestions.

TheExDamage - *June 17, 2013, 10:37 pm*

Nice work keep it up ?

Psychotiic - *June 18, 2013, 6:21 pm*

this is nice work, i really only care about the icon change though, but very nice overall