

Forum:[Hack Forums](#)

Thread: [WMIGatherer - WMI wrapper and system information gathering library](#)

chrizator - January 31, 2018, 12:52 pm

In a lot of my projects I work with WMI, usually to get informations about something. For me it was always time consuming and nerve wrecking to add a proper WMI implementation. That's basically why I made WMIGatherer and thought sharing it might be a good idea. It's a library that makes working with WMI in C# easier. I don't like to call it WMI wrapper, but I could not find a proper word to describe it. It also contains a "Gathering"-namespace that provides methods for information gathering purposes through WMI.

Just to give you a basic understanding what you can do with it:
(How things work and how to use it exactly is explained in the README file on GitHub.)

Querying Part

Query a single property:

Spoiler [\(Click to View\)](#)

```
Code string windowsCaption = Wmi.PropertyQuery<string>("Win32_OperatingSystem", "Caption");
```

Query multiple properties / multiple classes:

Spoiler [\(Click to View\)](#)

```
Code string[] properties = new string[] { "Capacity", "ClockSpeed" };
string condition = "Capacity > 5000";
WmiClassCollection collection = Wmi.Query("Win32_PhysicalMemory", properties, condition);
(Detailed usage is explained on GitHub [README.md])
```

Information Gathering Part

Get installed antivirus, antispware, firewall:

Spoiler [\(Click to View\)](#)

```
Code var antiviruses = SecurityGatherer.GetSecurityProduct(SecurityProductType.AntiVirus);
var firewalls = SecurityGatherer.GetSecurityProduct(SecurityProductType.Firewall);
var antispware = SecurityGatherer.GetSecurityProduct(SecurityProductType.AntiSpyware);
```

```
foreach (var av in antiviruses)
{
    string name = av.Name;
    string path = av.PathToExe;
    // Do your shit with it...
}
```

Create a HWID according to your desires:

Spoiler [\(Click to View\)](#)

```
Code // Only CPU ID used
string hwidLight = HardwareGatherer.GetHwid(HwidStrength.Light);

// CPU ID + HDD Signature
string hwidMedium = HardwareGatherer.GetHwid(HwidStrength.Medium);

// CPU ID + HDD Signature + BIOS Serial Number + MAC Address
string hwidStrong = HardwareGatherer.GetHwid(HwidStrength.Strong);
```

This is just a brief introduction, you can do a lot more. I would like this little project to become bigger and bigger. So, if somebody wants to have something added (any awesome WMI class or whatever), TELL ME!

Also, if unwanted exceptions or issues occur, please tell me. I tested it, but I'm sure that somewhere I forgot something, so let me know.

Before I forget, don't blame me for not using the 'var' keyword inside my code. I simply hate it, though some say it's wonderful.

GitHub: github.com/chrizator/WMIgatherer
(I can't post links :/)

HFDone - January 31, 2018, 1:03 pm

wmi is good to do hard things easy the only problem is speed

chrizator - January 31, 2018, 1:09 pm

(Jan 31, 2018 02:03 PM)HFDone Wrote: [\[CITE\]](#) wmi is good to do hard things easy the only problem is speed

That's true. WMI should be used when there is no way around it or speed doesn't matter.

Hoodie+ - January 31, 2018, 1:11 pm

Does this work for .NET 2.0?

I know that .NET 2.0 doesn't support querying WMI, or at least I've heard.

chrizator - January 31, 2018, 1:18 pm

(Jan 31, 2018 02:11 PM)Hoodie+ Wrote: [\[CITE\]](#) Does this work for .NET 2.0?

I know that .NET 2.0 doesn't support querying WMI, or at least I've heard.

Well, WMI (System.Management) works under .NET 2.0, but this project requires at least .NET 3.5.

Ninja - January 31, 2018, 8:49 pm

Its been a while since someone released code that I enjoyed reading through. This is some good code. Thanks for the release, buddy!

(Jan 31, 2018 02:11 PM)Hoodie+ Wrote: [\[CITE\]](#) Does this work for .NET 2.0?

I know that .NET 2.0 doesn't support querying WMI, or at least I've heard.

Pretty sure WMI itself works under 2.0, but not sure if OPs project works under 2.0.

chrizator - January 31, 2018, 9:27 pm

(Jan 31, 2018 09:49 PM)Ninja Wrote: [\[CITE\]](#) Its been a while since someone released code that I enjoyed reading through. This is some good code. Thanks for the release, buddy!

Pretty sure WMI itself works under 2.0, but not sure if OPs project works under 2.0.

Cheers, Ninja. ?

EZLib - February 2, 2018, 4:14 am

Thank you for releasing this. I'll be putting it to use.

0xDEAD10CC - February 3, 2018, 11:41 am

Just FYI you can do expression bodied properties:

```
Code public bool IsReadOnly => false;
```

Instead of this in C#:

```
Code public bool IsReadOnly
{
    get => false;
}
```

And your WmiClass could make use of some ref returns (C#7)

```
Code public WmiProperty[] Properties { get; }
```

```
public ref WmiProperty this[string propertyName]
{
    get
    {
        for (int i = 0; i < Properties.Length; i++)
        {
            if (Properties[i].Name.Equals(propertyName, StringComparison.Ordinal))
            {
                return ref Properties[i];
            }
        }

        string ex_msg = $"The specified property '{propertyName}' does not exist.";
        throw new IndexOutOfRangeException(ex_msg);
    }
}
```

Also much better to use Equals() for string comparison and define your comparison.

You've overcomplicated some of your util functions though such as GetHexString().

Not terrible code overall though. For something like this though you could probably make more use of abstract classes and interfaces for something which definitely has a hierarchy.

EdwardSn0wdent - February 3, 2018, 4:47 pm

Interested seeing your own approach to this tomorrow. Can I assume that this your own implementation of Microsoft's Scriptomatic tool, WMI Code Creator?

I enjoy working with WMI when speed is not of importance. It's rich in information although most examples out there are poorly written and need updating. Which is costly in time for me. In a small number of cases, it can be known to be extremely iffy.

chrizator - February 4, 2018, 4:08 am

(Feb 3, 2018 12:41 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Just FYI you can do expression bodied properties:

```
Code public bool IsReadOnly => false;
```

Instead of this in C#:

```
Code public bool IsReadOnly
```

```

{
    get => false;
}

```

And your WmiClass could make use of some ref returns (C#7)

```

Code public WmiProperty[] Properties { get; }

public ref WmiProperty this[string propertyName]
{
    get
    {
        for (int i = 0; i < Properties.Length; i++)
        {
            if (Properties[i].Name.Equals(propertyName, StringComparison.Ordinal))
            {
                return ref Properties[i];
            }
        }

        string ex_msg = $"The specified property '{propertyName}' does not exist.";
        throw new IndexOutOfRangeException(ex_msg);
    }
}

```

Also much better to use Equals() for string comparison and define your comparison.

You've overcomplicated some of your util functions though such as GetHexString().

Not terrible code overall though. For something like this though you could probably make more use of abstract classes and interfaces for something which definitely has a hierarchy.

Thanks for the advice. It's great to have people on hear that actually read code and share their experience. I'll try to adapt my code ASAP.

Btw, I'm wondering how much of a difference Equals() makes instead of comparing the object references.

0xDEAD10CC - February 4, 2018, 7:49 am

(Feb 4, 2018 05:08 AM)chrizator Wrote: [\[CITE\]](#) Thanks for the advice. It's great to have people on hear that actually read code and share their experience. I'll try to adapt my code ASAP.

Btw, I'm wondering how much of a difference Equals() makes instead of comparing the object references.

You're not comparing references when you use == on strings because it's overloaded.

Increased speed and recommended by Microsoft since 2005:

<https://msdn.microsoft.com/en-us/library/ms973919?f=255&MSPPError=-2147217396>

Avoid problems like this from time to time with other reference types:

<https://blogs.msdn.microsoft.com/csharpfaq/2004/03/29/when-should-i-use-and-when-should-i-use-equals/>

It's more of a common habit for developers as well and makes it explicit as to what kind of string comparison you're doing. Also, if you want to compare references there's already a ReferenceEquals function too so you don't have to worry about this overloading. It's okay to use == and others on value types though as they don't overload these operators by default.

chrizator - February 4, 2018, 8:23 am

(Feb 4, 2018 08:49 AM)0xDEAD10CC Wrote: [\[CITE\]](#) You're not comparing references when you use

== on strings because it's overloaded.

Increased speed and recommended by Microsoft since 2005:

<https://msdn.microsoft.com/en-us/library/ms973919?f=255&MSPPError=-2147217396>

Avoid problems like this from time to time with other reference types:

<https://blogs.msdn.microsoft.com/csharpfaq/2004/03/29/when-should-i-use-and-when-should-i-use-equals/>

It's more of a common habit for developers as well and makes it explicit as to what kind of string comparison you're doing. Also, if you want to compare references there's already a `ReferenceEquals` function too so you don't have to worry about this overloading. It's okay to use `==` and others on value types though as they don't overload these operators by default.

Oh, that's interesting to know. Thanks for the clarification!