

Re: Ridiculous!

Source: <http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.general/2004-04/0810.html>

From: Willy Denoyette [MVP] (willy.denoyette_at_pandora.be)

Date: 04/08/04

Date: Thu, 8 Apr 2004 23:12:49 +0200

I'm the "author" of this example, but be warned, you should never ever use this to check NTFS file permissions in user applications (other than FS management applications), in order to determine whether you have read, write or whatever access to a file.

Just open the file in the mode you need and let the filesystem do it's job, all you need to do is handle the exception thrown.

Willy.

"Michael Bray" <mbrayXXnoSPAMpleaseXX@SkPiAlMl.ctiusa.com> wrote in message news:Xns94C56B5AF6655mbrayxyzctiusaabccom@207.46.248.16...

> "Ayende Rahien" <Ayende@no.spam> wrote in
> news:O3#3iEXHEHA.3968@TK2MSFTNGP12.phx.gbl:
>
>> <rant warning="I'm pissed off and have to vent" request="need help">
>> For the last couple of hours I'm struggling with a very *annoying*
>> problem. How to check if a user has a write access to a file?
>> Considering that .Net is supposed to be a system for writing
>> applications for servers & clients, which in both cases has may have
>> multi-users, I'm amazed that this is not possible in the framework.
>> Initially I looked for something like:
>>
>> System.IO.File.Access(string Filename, PermissionAccess);
>>
>> Nothing!
>
>
> I happened to run across some code in this very same newsgroup asking
> basically the same question – I'll repost it for you assuming that you
> didn't see it. This isn't my code and I haven't tested it, so use at your
> own risk... I've included a few of the headers as well.
>
> -----
>
> Subject: Re: Get ACL
> From: "Willy Denoyette [MVP]" <willy.denoyette@pandora.be>
> Newsgroups: microsoft.public.dotnet.languages.csharp
>

microsoft.public.dotnet.general: Re: Ridiculous!

```
> You shouldn't use "unsupported" stuff like Win32Security.dll, use the
> System.DirectoryServices (XP and higher) or System.Management namespace
> instead.
> Next is a complete example illustrating how to dump the ACE's from a File
> object DACL using System.Management classes.
>
> using System;
> using System.Management;
> using System.Collections;
> // Access mask (see AccessMask property)
> [Flags]
> enum Mask : uint
> {
> FileReadData = 0x00000001,
> FileWriteData = 0x00000002,
> FileAppendData = 0x00000004,
> FileReadEA = 0x00000008,
> FileWriteEA = 0x00000010,
> FileExecute = 0x00000020,
> FileDeleteChild = 0x00000040,
> FileReadAttributes = 0x00000080,
> FileWriteAttributes = 0x00000100,
>
> Delete = 0x00010000,
> ReadControl = 0x00020000,
> WriteDac = 0x00040000,
> WriteOwner = 0x00080000,
> Synchronize = 0x00100000,
>
> AccessSystemSecurity = 0x01000000,
> MaximumAllowed = 0x02000000,
>
> GenericAll = 0x10000000,
> GenericExecute = 0x20000000,
> GenericWrite = 0x40000000,
> GenericRead = 0x80000000
> }
> [Flags]
> enum AceFlags : int
> {
> ObjectInheritAce = 1,
> ContainerInheritAce = 2,
> NoPropagateInheritAce = 4,
> InheritOnlyAce = 8,
> InheritedAce = 16
> }
>
> [Flags]
> enum AceType : int
> {
> AccessAllowed = 0,
```

Re: Ridiculous!

```

> AccessDenied = 1,
> Audit = 2
> }
> class Tester {
> public static void Main() {
> string fileObject = @"c:\\pipot.txt"; // Watch the double Backslashes
> using(ManagementObject lfs = new
> ManagementObject(@"Win32_LogicalFileSecuritySetting.Path=" + "" +
> fileObject + ""))
> {
> // Get the security descriptor for this object
> // Dump all trustees (this includes owner)
> ManagementBaseObject outParams =
> lfs.InvokeMethod("GetSecurityDescriptor", null, null);
> if(((uint)(outParams.Properties["ReturnValue"].Value)) == 0) // if
> success
> {
> ManagementBaseObject secDescriptor =
> ((ManagementBaseObject)(outParams.Properties["Descriptor"].Value));
> //The DACL is an array of Win32_ACE objects.
> ManagementBaseObject[] dacl =
> ((ManagementBaseObject[])(secDescriptor.Properties["Dacl"].Value));
> DumpACEs(dacl);
>
> }
> }
> }
>
> static void DumpACEs(ManagementBaseObject[] dacl)
> {
> foreach(ManagementBaseObject mbo in dacl){
> Console.WriteLine("\n-----\nMask: {0:X} - Flags: {1} - Type: {2}",
> mbo["AccessMask"], mbo["AceFlags"], mbo["AceType"]);
> // Access allowed/denied ACE
> if(Convert.ToInt32(mbo["AceType"]) == (int)AceType.AccessDenied)
> Console.WriteLine("DENIED ACE TYPE");
> else
> Console.WriteLine("ALLOWED ACE TYPE");
> // Dump trustees
> ManagementBaseObject Trustee = ((ManagementBaseObject)(mbo["Trustee"]));
> Console.WriteLine("Name: {0} - Domain: {1} - SID {2}\n",
> Trustee.Properties["Name"].Value,
> Trustee.Properties["Domain"].Value,
> Trustee.Properties["SIDString"].Value);
> // Dump ACE mask in readable form
> UInt32 mask = (UInt32)mbo["AccessMask"];
> Console.WriteLine(Enum.Format(typeof(Mask), mask, "g"));
> }
> }
> }
>

```