Printable Version

WMI Made Easy For C#

By Kevin Matthew Goss

Windows Management is somewhat mystified to most developers, and admittedly I myself was taken aback by it for a while. Recently I attempted to dig in to the .Net Management class and learn WMI a little better. To my surprise, it was quite easy with the accumulated knowledge I have gained in C# along with the WMI reference on MSDN.

WMI is a very powerfull tool, and once you know how to get what you need, it can be invaluable as a time saver. When developing Windows applications, developers often need information a system, either local or remote, that although commonplace, can be very tough to get. There is using the remote registry, but I myself do not allow remote registry access as do many network admins. WMI is usually wide open on networks, assuming you have the privelidges necessary to query it, just as it is with remote registry querying/editing. And although remote registry querying is very simple, WMI appeals to developers for yet another reason: WQL.

WQL allows us to query WMI providers using a SQL-like query language. If you know the provider classes and the fields available, then you can get the info very easily. For instance, if you wanted to get a list of logical drives from a system you would use the following query:

Select * from Win32_LogicalDisk

You can, however, refine the search by using where clauses and getting specific "fields" in the query. The following query gets the amount of freespace, the size, and the name of all fixed disk drives:

Select FreeSpace,Size,Name from Win32_LogicalDisk where DriveType=3

As you can see, constructing a simple WMI query is quite easy. To get results, you need and interface, and in .Net it is provided by the System.Management namespace. To make it work all you need is a query, and a little bit of code, just as if you were querying a database.

You need a few different objects to perform WMI queries in .Net. They include the following:

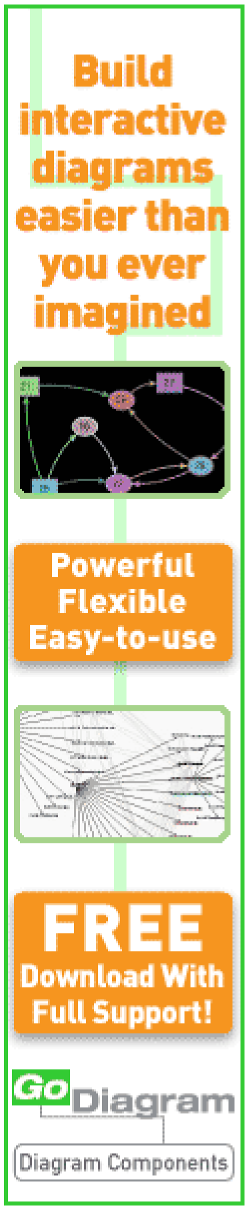
Search Forum
(47199 Postings)
GO

3 Months FREE! click here for info

C# Help Board

Archived Articles
680 Articles

C# Books
C# Consultants
Search Site/Articles
What Is C#?
Download Compiler
Code Archive
Archived Articles
Advertise
Contribute
C# Jobs
Beginners Tutorial
C# Contractors
C# Consulting
Links
C# Manual
Contact Us
Legal


Build interactive diagrams easier than you ever imagined
Powerful Flexible Easy-to-use
FREE Download With Full Support!
GoDiagram
Diagram Components

Best sites

(all within System.Management)

- ConnectionOptions
- ManagementScope
- ObjectQuery
- ManagementObjectSearcher
- ManagementObjectCollection
- ManagementObject

Though this may seem like a lot of objects to perform a simple query, it is quite easy in practice. I will not go into great detail on the objects (you can review each object in the .Net documentation as they are documented very thoroughly). I will attempt to show a very easy way of utilizing these objects to query WMI providers as well as perform intristic methods available on some of the objects.

The following code shows use the query above on a remote system (MachineX) using user JohnDoe and password JohnsPass:

```
//Connection credentials to the remote computer - not needed if the
logged in account has access
ConnectionOptions oConn = new ConnectionOptions();
oConn.Username = "JohnDoe";
oConn.Password = "JohnsPass";

System.Management.ManagementScope oMs = new
System.Management.ManagementScope("\\MachineX", oConn);

//get Fixed disk stats
System.Management.ObjectQuery oQuery = new
System.Management.ObjectQuery("select FreeSpace,Size,Name
from Win32_LogicalDisk where DriveType=3");

//Execute the query
ManagementObjectSearcher oSearcher = new
ManagementObjectSearcher(oMs,oQuery);

//Get the results
ManagementObjectCollection oReturnCollection = oSearcher.Get();

//loop through found drives and write out info
foreach( ManagementObject oReturn in oReturnCollection )
{
   // Disk name
   Console.WriteLine("Name : " + oReturn["Name"].ToString());
   // Free Space in bytes
   Console.WriteLine("FreeSpace: " + oReturn["FreeSpace"].ToString
());
   // Size in bytes
   Console.WriteLine("Size: " + oReturn["Size"].ToString());
}
```

As you can see, the code is not that difficult. This, although a simple query, would save a lot of time compared to other methods, especially when querying a remote machine. Please note that usually the ManagementScope would require a WMI namespace in addition to the machine name, but .Net kindly defaults to the root namespace. If you wish to use it anyway you would use the following scope:

> **\\MachineX\root\cimv2** (the double \ is required for string literals in C#)

One of the problems I had in using WMI with .Net was not knowing what "fields" were available for a given object. I found the class reference on MSDN and all the problems went away, at least most of them. Data type conversions can be a problem, especially with datetime structures. Datetime data types from WMI providers are *not* compatible with .Net DateTime variables. You must use a managed function that you can get from my sample code or by using the *mgmtclassgen* utility that comes with the .Net SDK (Thanks to Chetan Parmar for this info and the code). Also, some objects will return *null* in some fields, so make sure to check for it (see sample code).

WMI Method Invocation

Another interesting feature of WMI is the methods that are available with certain objects. For instance, with the Win32_Process object you can use the GetOwner method to return the name and domain of the user under whom the process is running. You must use the ***Invoke*** method of the ManagementObject object and send it the proper parameters. In this case you are only required to send the name of the method as a string ("GetUser") and a 2 element string array for the return. Don't be fooled. Even though the array would *seem* to be used as a *ref* variable, you do not have to declare that way when calling the ***Invoke*** method.

Below is a sample of getting all processes along with the name, user and domain, memory used, priority, and process id for each process. This information is similar to what you see in the task manager. If you want CPU usage you have to use the Win32_PerfFormattedData_PerfProc_Process class which is actually a WMI interface for the perfomance counters. I do not use this class because the GetOwner method is not available with it.

```
//Connection credentials to the remote computer - not needed if the
logged in account has access
ConnectionOptions oConn = new ConnectionOptions();
oConn.Username = "JohnDoe";
oConn.Password = "JohnsPass";

System.Management.ManagementScope oMs = new
System.Management.ManagementScope("\\MachineX", oConn);
```

```
//get Process objects
System.Management.ObjectQuery oQuery = new
System.Management.ObjectQuery("Select * from Win32_Process");
foreach( ManagementObject oReturn in oReturnCollection )
{
   //Name of process
   Console.WriteLine(oReturn["Name"].ToString().ToLower());
   //arg to send with method invoke to return user and domain -
below is link to SDK doc on it
   string[] o = new String[2];
   //Invoke the method and populate the o var with the user name
and domain
   oReturn.InvokeMethod("GetOwner",(object[])o);
   //write out user info that was returned
   Console.WriteLine("User: " + o[1]+ "\" + o[0]);
   Console.WriteLine("PID: " + oReturn["ProcessId"].ToString());
   //get priority
   if(oReturn["Priority"] != null)
      Console.WriteLine("Priority: " + oReturn["Priority"].ToString());

   //get creation date - need managed code function to convert date
-
   if(oReturn["CreationDate"] != null)
   {
      //get datetime string and convert
      string s = oReturn["CreationDate"].ToString();
         //see ToDateTime function in sample code
      DateTime dc = ToDateTime(s);
      //write out creation date
      Console.WriteLine("CreationDate: " + dc.AddTicks(-
TimeZone.CurrentTimeZone.GetUtcOffset
(DateTime.Now).Ticks).ToLocalTime().ToString());
   }

   //this is the amount of memory used
   if(oReturn["WorkingSetSize"] != null)
   {
      long mem = Convert.ToInt64(oReturn
["WorkingSetSize"].ToString()) / 1024;
      Console.WriteLine("Mem Usage: {0:#,###.##}Kb",mem);
   }
}
```

There is a wealth of information waiting to be gleaned from WMI and it is far easier than using several API calls or remote registry calls. WMI simplifies things by making all common information handy in one place. You can get all system info, partition info, processor stats, profile settings, and much more using WMI. WMI can replace performance counters as well. Once you get used to it, browse through the MSDN WMI class reference and you are sure to find what you are looking for in most cases.

I included all of the above code in a sample console project with the source code download, including the datetime conversion function. Happy coding!!!

Download WMITest.zip

References:

MSDN WMI Reference
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_reference.asp?frame=true

Article by John O'Donnell
http://www.dotnet247.com/247reference/a.aspx?u=http://www.c-harpcorner.com/Code/2002/Jan/InterrogatingSystemsWithWMIJO.asp