

Creating Managed / Management Applications with WMI and .NET

Dominick Baier (dbaier@ernw.de)

.NET Security Consultant / BS 7799 Lead Auditor

www.leastprivilege.com



Outline

- **What is WMI?**
- **WMI Architecture**
- **WMI Schema**

- **Accessing WMI with .NET**
- **Instrumenting Applications with WMI**

- **WMI Explorer for Visual Studio**
- **Microsoft Operations Manager 2005**



What is WMI?

- **Functional**

- A single point of access for application / system monitoring and control
- A rich set of event- and query-management services
- Remote enabled
- Microsoft's implementation of the WBEM standard

- **Technical**

- A set of system services, (Distributed) COM objects and interfaces
- Surfaces in the BCL in
 - `System.Management`
 - `System.Management.Instrumentation`



Why should I use WMI?

- **Management**
 - control and monitor nearly every aspect of your Windows landscape
 - Get notifications for system / application Events
 - Many WMI enabled apps are out there
- **Get Managed**
 - Make your application "monitorable"
 - through standard system management applications
 - Notify staff about important events in your App
 - think detection and reaction
 - Easy to use



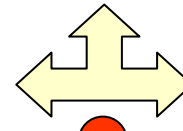
WMI Architecture

Any application that understands automation

i.e. C/C++, VB, VBScript, Jscript, VBA, Perl

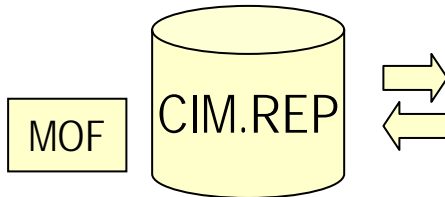
Consumer

Application



System.Management

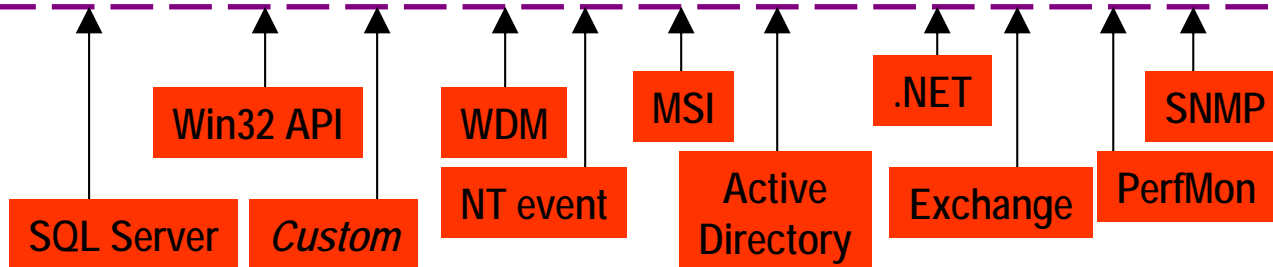
CIM



Windows Management Service
%SystemRoot%\System32\WBEM\WinMgmt.exe

COM

Provider



WMI Schema

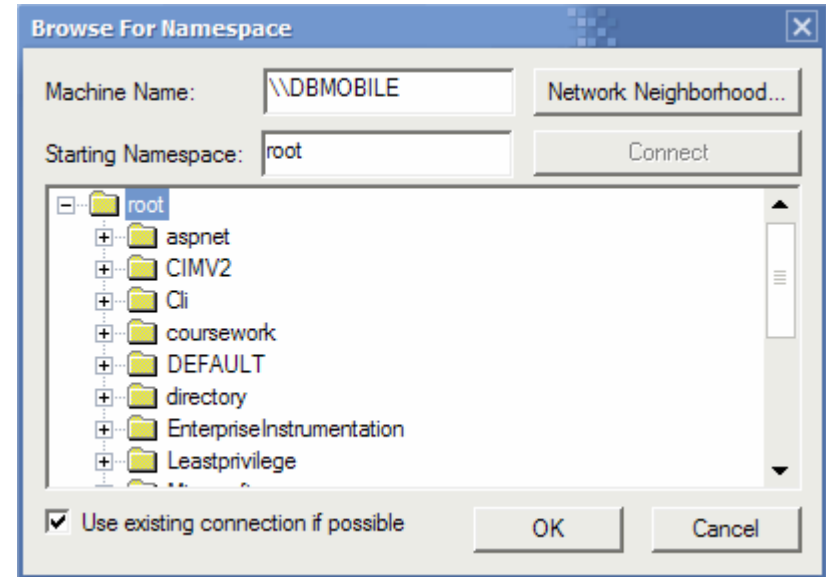
- **The WMI Schema describes the provided management information**
- **Schema is defined through a MOF file**
- **Schema is stored in the CIM**
- **Schema consists of**
 - Namespaces
 - Classes
 - Events

 - Relationships between classes



WMI Namespaces

- Partition the WMI Schema
- Hold classes
- Are extensible
- Start with **root**
- Most of the system-provided classes are in **root\cimv2**



WMI Classes

- **Describe the provided information**
- **Support inheritance**
- **Consist of**
 - Properties
 - read, write, read/write
 - Methods
 - in / out parameters
 - Meta Data
 - data types, relationships
- **E.g. Win32_Process, Win32_NetworkAdapterConfiguration**

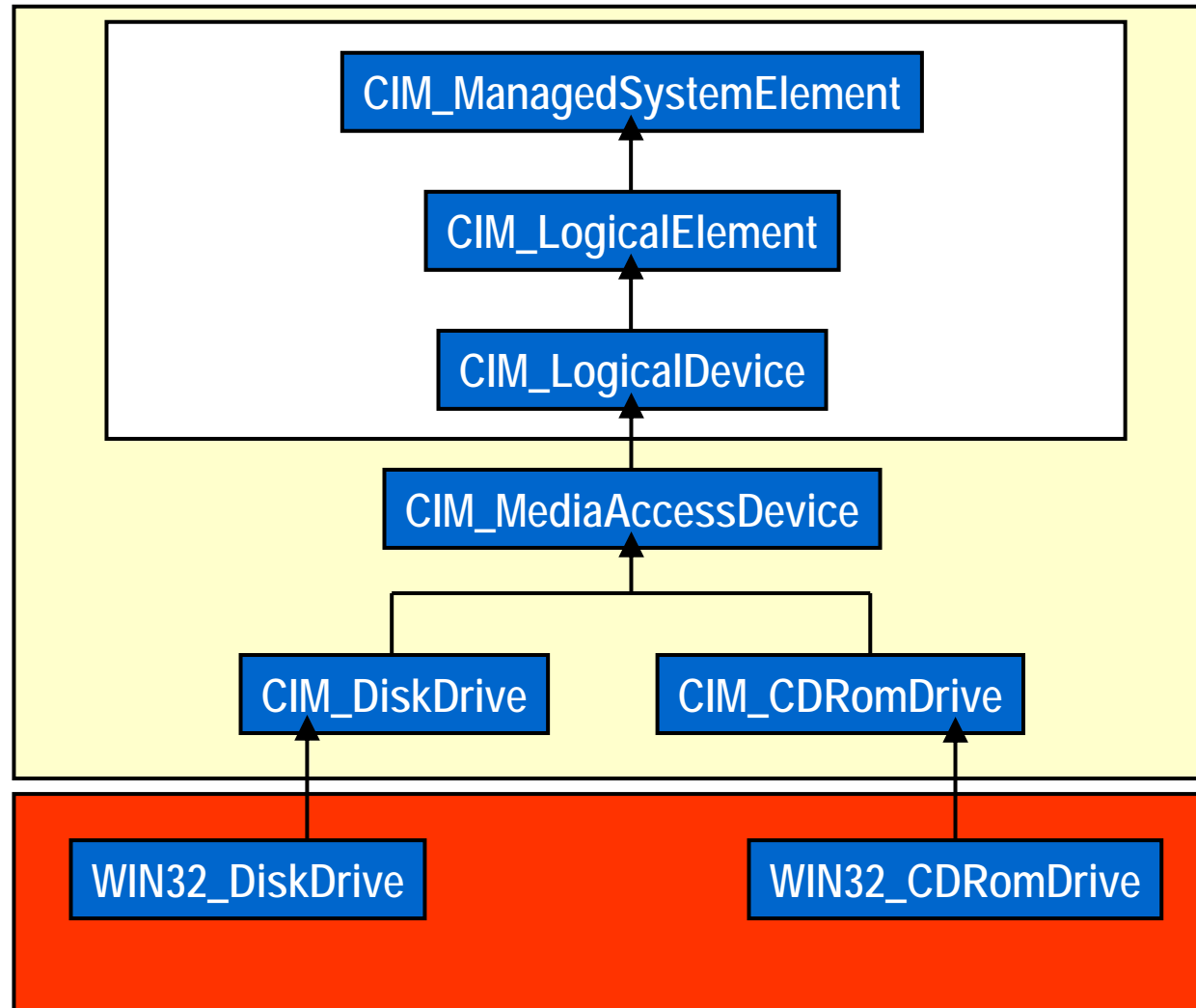


WMI Classes and Inheritance

**Core
Schema**

**Common
Schema**

**Extensible
Schema**



Schema Definition

- Classes are described in MOF files

```
[Dynamic, Provider("CIMWin32")]
class Win32_NetworkAdapterConfiguration : CIM_Setting
{
    [Read : key] uint32 Index;

    [Read] string MACAddress;
    [Read] string IPAddress[];
    [Read] string IPSubnet[];
    [Read] string DefaultIPGateway[];

    [Implemented] uint32 EnabledDHCP();
    [Implemented] uint32 EnableStatic([In] string IPAddress[],
                                     [In] string SubnetMask[]);
    [Implemented] uint32 SetGateways([In] string DefaultIPGateway[],
                                     [In] uint16 GatewayCostMetric[]);

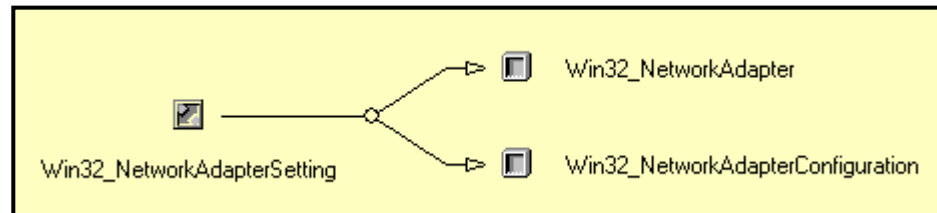
    ...
}
```



Schema Definition

- **Classes can have associations to other classes**
- **Associations can be traversed programmatically**

```
[Dynamic, Provider("CIMWin32")]  
class Win32_NetworkAdapterSetting : Win32_DeviceSettings  
{  
    [Read] Win32_NetworkAdapter Ref Element;  
    [Read] Win32_NetworkAdapterConfiguration Ref Setting;  
};
```



Browsing the Repository

- **There are several tools to inspect the WMI repository**
- **WMI Tools**
 - <http://www.microsoft.com/downloads/details.aspx?FamilyID=6430f853-1120-48db-8cc5-f2abdc3ed314&DisplayLang=en>
- **WMIC**
 - included in XP / W2K3
- **WMI Browser written in C#**
 - <http://www.developerland.com/DotNet/Enterprise/145.aspx>



Connecting to WMI

- **Locally**

```
// relying on the standard namespace (which is root\cimv2 by default)
ManagementClass mc = new ManagementClass("Win32_NetworkAdapterConfiguration");

// using a fully qualified namespace
ManagementClass mc = new ManagementClass
    ("\\root\\cimv2:Win32_NetworkAdapterConfiguration");
```



Connecting to WMI

- Remote

```
// using default settings and integrated security

ManagementClass mc = new ManagementClass
    ("\\\\server\\root\\cimv2:Win32_NetworkAdapterConfiguration");

// controlling connection options

ConnectionOptions co = new ConnectionOptions();
co.Username = "server\\administrator";
co.Password = "password";
co.Impersonation = ImpersonationLevel.Impersonate;
co.Authentication = AuthenticationLevel.PacketPrivacy;

ManagementScope scope = new ManagementScope("\\\\server\\root\\cimv2", co);
scope.Connect();
```



Getting Instances

- `.GetInstances()` returns all instances of a class

```
public void ShowInterfaces()
{
    ManagementClass mc = new ManagementClass("Win32_NetworkAdapterConfiguration");
    ManagementObjectCollection moc = mc.GetInstances();

    foreach (ManagementObject mo in moc)
    {
        if (! (bool) mo["IPEnabled"])
            continue;

        Console.WriteLine("Interface Number : " + mo["Index"]);
        Console.WriteLine("{0}\n SVC: '{1}' MAC: [{2}]\n",
            (string) mo["Caption"],
            (string) mo["ServiceName"],
            (string) mo["MACAddress"]);
    }
}
```



Getting Instances

- You can also access a specific instance by the key

```
private ManagementObject getInterface(int index)
{
    ManagementObject mo = new ManagementObject
        ("Win32_NetworkAdapterConfiguration.Index=" + index.ToString());

    mo.Get();
    return mo;
}
```

- ...or by using a query

```
private ManagementObjectCollection getAllIPEnabledInterfaces()
{
    ManagementObjectSearcher query = new ManagementObjectSearcher
        ("Select Index, Caption, ServiceName, MACAddress from
        Win32_NetworkAdapterConfiguration where IPEnabled=true");

    ManagementObjectCollection moc = query.Get(); return moc;
}
```



Calling Methods

```
public void SetIPAddress(string InterfaceNumber, string IPAddress,
                        string SubnetMask)
{
    ManagementBaseObject inPar = null;
    ManagementBaseObject outPar = null;
    ManagementObject mo = null;

    mo = getInterface(InterfaceNumber);
    if (mo != null)
    {
        inPar = mo.GetMethodParameters("EnableStatic");
        inPar["IPAddress"] = new string[] { IPAddress };
        inPar["SubnetMask"] = new string[] { SubnetMask };

        outPar = mo.InvokeMethod("EnableStatic", inPar, null);
    }
}
```



WMI Events

- **Intrinsic Events**

- Occur in response to changes in the CIM
- Instance[Creation|Modification|Deletion]Event
- Class[Creation|Modification|Deletion]Event
- Namespace[Creation|Modification|Deletion]Event

- **Extrinsic Events**

- Originate outside the CIM
- Are raised by Event Providers



Events

- **Getting notifications if instances in WMI change**
 - e.g. mimicking Whidbey's `System.Net.NetworkInformation`

```
public void StartWatching()
{
    WqlEventQuery query = new WqlEventQuery("__InstanceModificationEvent",
        new TimeSpan(0,0,5),
        "TargetInstance isa \"Win32_NetworkAdapterConfiguration\"");

    ManagementEventWatcher eventWatcher = new ManagementEventWatcher(query);
    eventWatcher.EventArrived += new EventArrivedEventHandler(EventArrived);

    eventWatcher.Start();
}

private void EventArrived(object sender, EventArrivedEventArgs e)
{
    uint index = System.Convert.ToUInt32(
        ((ManagementBaseObject)e.NewEvent["TargetInstance"])["Index"]);

    Console.WriteLine("NIC #{0} has changed its state", index);
}
```



WMI Explorer for Visual Studio

- **Addon for VS 2003**
- **Included in VS 2005**

- **Features**
 - Browsing the repository
 - Calling methods on instances

 - Creating managed wrappers for classes / instances
 - Subscribe to Events



Creating WMI Classes

- **System.Management.Instrumentation provides a number of attributes that make it easy to instrument your application**
 - `[assembly: Instrumented("root/Leastprivilege")]`
- **Classes that should be published to the WMI repository**
 - are decorated with `[InstrumentationClass]`
 - or inherit from `Instance / BaseEvent`
- **Public fields and properties become WMI properties**
 - you can also apply the `[IgnoreMember]` Attribute
- **You have to register the class in the CIM**
 - `DefaultManagementInstaller` takes care of that
- **Support is limited to read properties**



Creating WMI Classes

```
public class LogonFailureSummary : Instance
{
    public string Area;
    public int Count;
    public string LastFailedLogon;
}

[RunInstaller(true)]
public class MyProviderInstaller : DefaultManagementProjectInstaller
{
}

public class Program
{
    public static void Main()
    {
        LogonFailureSummary l = new LogonFailureSummary();
        l.Area = "Main App Logon";
        l.Count = 5; l.LastFailedLogon = DateTime.Now.ToString();

        l.Published = true; // publish the instance to WMI
    }
}
```



Querying our WMI Classes

```
LogonFailureSummary.LogonFailureSummaryCollection logons =  
    LogonFailureSummary.GetInstances();  
  
IEnumerator lenum = logons.GetEnumerator();  
  
while (lenum.MoveNext())  
{  
    LogonFailureSummary l = (LogonFailureSummary) lenum.Current;  
    Console.WriteLine("{0}: , {1}", l.Area, l.Count.ToString());  
}
```



Creating WMI Events

```
public class SecurityEvent : BaseEvent
{
    public string Message;

    public SecurityEvent(string message)
    {
        this.Message = message;
        base.Fire();
    }
}

public class Program
{
    static void Main()
    {
        // logon procedure

        if (3 == numInvalidLogons)
            new SecurityEvent("Three invalid logons for user " + user);
    }
}
```



Subscribing to WMI Events

```
WqlEventQuery eventQuery = new WqlEventQuery("SecurityEvent");
ManagementScope scope = new
    ManagementScope(@"\\.\root\LeastPrivilege\Demo");

eventWatcher = new ManagementEventWatcher(scope, eventQuery);
eventWatcher.EventArrived += new
    EventArrivedEventHandler(Delegate_EventArrived);

eventWatcher.Start();

private void Delegate_EventArrived(object sender, EventArrivedEventArgs e)
{
    updateEvents(Convert.ToString(e.NewEvent.Properties["Message"].Value));
}
```



Summary

- **WMI enables you to write powerful Windows applications**
- **WMI exposes rich functionality and management data**
- **Instrument your own applications to integrate with standard systems management tools**
- **.NET support is limited**



