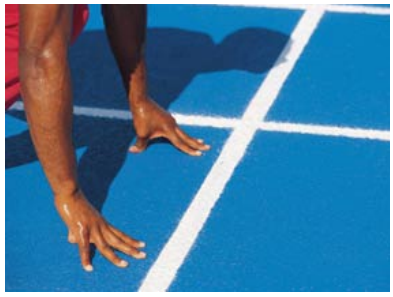
maXbox



maXbox Starter 52

Work with WMI

1.1 Management Instrumentation

Most Win computers these days have something on them called WMI or Windows Management Instrumentation. You can do a number of useful things on the local computer or even a remote computer using WMI like network management, security or real-time health monitoring[1].
For me the best is the query possibility called WQL for ex.:

```
isQuery:=
      'SELECT * FROM Win32_Service WHERE State = "Running"';
iset:= WMIExecQuery(isser, isQuery); //Wbem ObjectSet;
```

Then you get a list of your services running! Even more so, this is the recommended standard Microsoft way to do operational things and it is ever more so when it comes to Win 7 and 10.
WMI (most cases I've seen) is arranged in the form of object tables, whose data (fields or properties) are accessed in a subset of SQL that Microsoft calls WQL. So you don't need a DB or a server to execute those queries.

To grasp the power and use of WMI, consider how you managed and monitored Win workstations and servers yesterday, and perhaps even today. You probably used, or use, numerous graphical administrative tools to manage resources, such as disks, event logs, files, folders, file systems, networking components, OS settings, performances, printers, processes, registry settings, security services, shares, users, groups, and so on.
So how do you start? with a locator and a connector to pass:

```
var isloc: ISWBemLocator;
    isser: ISWBemServices; Enum: IEnumVariant;

    isloc:= WMIStart;
    isser:= WMIConnect(isloc, 'maxbox10','max',''); //WMIServices;
```

[1] A hint to the web of things

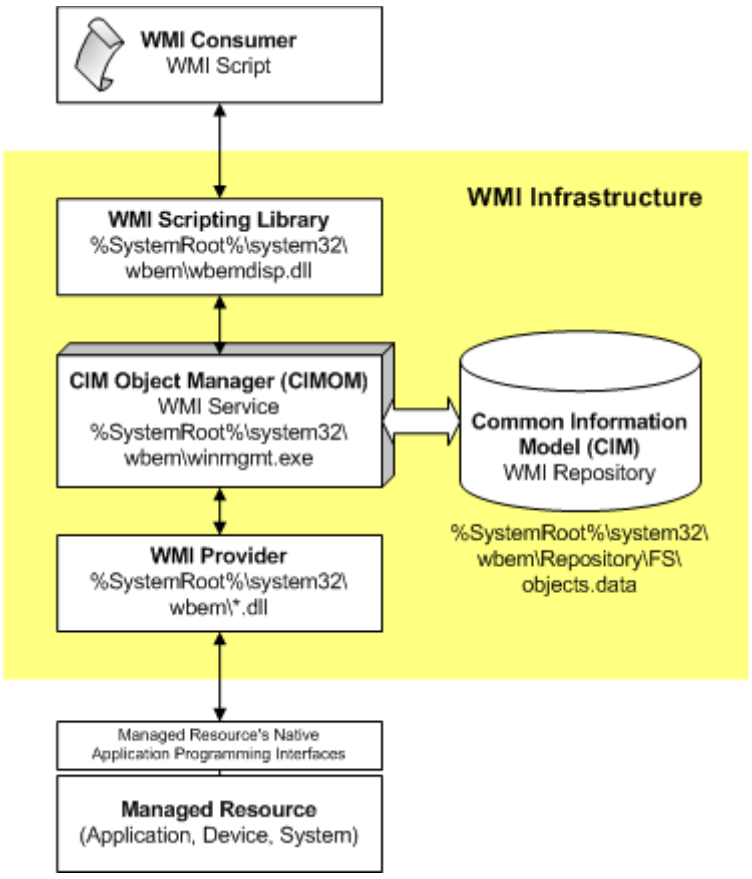
Those are the interfaces which are mapped to a type library:

```
function WMIStart: ISWBemLocator;
function WMIConnect(WBemLocator:ISWBemLocator;Server,account,password:string):
                                                                    ISWBemServices;
function WMIExecQuery(WBemServices:ISWBemServices; query:string): ISWbemObjectSet;
function WMIRowFindFirst(ObjectSet:ISWbemObjectSet; var Enum:IEnumVariant;
                                          var tempobj: OleVariant):boolean;
function WMIRowFindNext(ENum: IENumVariant; var tempobj: OleVariant): boolean;
function WMIColFindFirst(var propENum:IENumVariant;var tempObj:OleVariant):boolean;
function WMIColFindNext(propENum: IENumVariant; var tempobj: OleVariant):boolean;
function WMIGetValue(wbemservices:ISWBemServices;tablename,fieldname:string):string;
function WMIConvValue(tempobj: OleVariant; var keyname: string): string;
```

Second you call the query and iterate through the object set:

```
isQuery:= 'SELECT * FROM Win32_Service WHERE State = "Running"';
      iset:= WMIExecQuery(isser, isQuery); //WbemObjectSet;
      WMIRowFindFirst(iset, ENum, tempObj);
      repeat
        writeln('case: '+tempobj.name+' -- '+(tempobj.state))
      until not WMIRowFindNext(ENum, tempobj); //}
      tempObj:= unassigned;
```

Code and script can be found at:
http://www.softwareschule.ch/examples/766_wmi_management.txt



What is allowed and not allowed, along with these table definitions are in the MSDN reference for WMI:

http://msdn.microsoft.com/en-us/library/aa394582(VS.85).aspx

Of course there's a great number of them, and you can find out just about anything regarding your local system or any system on the network that you can connect to like the running hardware statistic, along with memory usages, processes, and threads or a physical element like temp:

```
FWbemObjectSet:= FWMIService.Get('Win32_TemperatureProbe');
```

Beware, that selecting all the data in some of the tables will take a very large time and a huge amount of resources, so be careful to limit your statements as you would in dealing with a regular database.
You do also have more than one root name space available, even the query language in the object set is chose able:

```
const
  WbemComputer      ='localhost'; //wbemFlagForwardOnly= $00000020;
  WbemRootNameSpace ='root\CIMV2';

FWbemObjectSet:= FWMIService.ExecQuery(Format('Select %s from %s',
                [WMIProperty, WMIClass]),'WQL',wbemFlagForwardOnly);
```

Each item in a `SWbemObjectSet` is a `SWbemObject` (yet another object (Interface) in the WMI scripting library) that represents a single instance of the requested resource. You use `SWbemObject` and `WMIRowFindNext` to access the methods and properties defined in the managed resource's class definition `WMIClass`.

The WMI architecture consists of three primary layers as shown in the picture above from MS (`ms974579.aspx`):

- Managed resources
- WMI infrastructure
- Consumers

Let's do a second example in the same script, we want to list all running processes like a task manager does:

```
isQuery:= 'SELECT * FROM Win32_Process';
iset:= WMIExecQuery(isser, isQuery); //WbemObjectSet;
writeln(botoStr(WMIRowFindFirst(iset, ENum, tempObj)));
it:= 0;
  repeat
    PrintF('Processes run: %s - PID %s',
                [tempobj.name,vartoStr(tempobj.processid)])
    inc(it)
  until not WMIRowFindNext(ENum, tempObj);
writeln('running Processes : '+itoa(it))
```

So, if the scripting steps to retrieve information from WMI are identical, what are the `Win32_Process`, `Win32_Service`, and `Win32_NTLogEvent` classes?
Furthermore, where do they come from, what other classes are available, and how do you use them? The answers to these questions lie in the components that make up the WMI architecture.
As I said we mapped the type library to functions to simplify the handling in a dynamic script, lets have a look at the interface:

```
// ************************************************************************//
// DispIntf:  ISWbemPropertyDisp
// Flags:     (4432) Hidden Dual OleAutomation Dispatchable
// GUID:      {1A388F98-D4BA-11D1-8B09-00600806D9B6}
// ************************************************************************//
  ISWbemPropertyDisp = dispinterface
    ['{1A388F98-D4BA-11D1-8B09-00600806D9B6}']
    function Value: OleVariant; dispid 0;
    property Name: WideString readonly dispid 1;
    property IsLocal: WordBool readonly dispid 2;
    property Origin: WideString readonly dispid 3;
    property CIMType: WbemCimtypeEnum readonly dispid 4;
    property Qualifiers_: ISWbemQualifierSet readonly dispid 5;
    property IsArray: WordBool readonly dispid 6;
  end;
```

The `dispinterface` statement defines a set of properties and methods on which you can call. This function `value` will add a value of the type `OleVariant` that are represented by their properties (digits, strings and more) so we use in most cases strings as the representation of the value (`VarToStr()`).

You can use WMI in maXbox by early binding (pre compiled with `WbemScripting_TLB;`) or late binding with `CreateOleObject()`.

```
wmiLocator:= CreateOleObject('WbemScripting.SWbemLocator');
```

As you begin to write scripts to interact with WMI managed resources, you'll often see the term instance used to refer to a virtual representation of the managed resource in the running script.
So we check the availability of the WMI late binding:

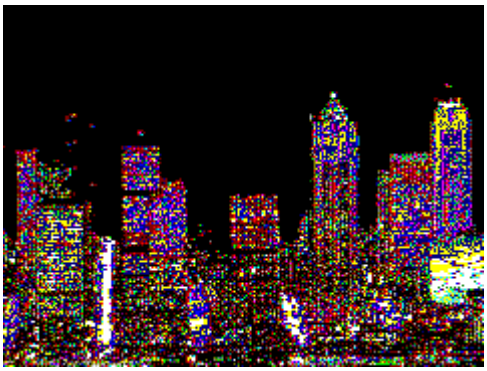
```
 (ClassIDToString(StringToClassID('WbemScripting.SWbemLocator'));
  writeln('WMI Installed:
      '+botoStr(ProgIDInstalled('WbemScripting.SWbemLocator')))

>>> {76A64158-CB41-11D1-8B02-00600806D9B6}
WMI Installed: TRUE
```

an interesting service is always BITS : (you guess why?)
`case: BITS - Background Intelligent Transfer Service --Running`

Also within WMI, there are numerous growing provider methods which will do different things.
At least there are two ways to install and configure your box tool into a directory you want. The first way is to use the unzip command-line tool or IDE, which is discussed above.
That means no installation needed. Another way is to copy all the files to navigate to a folder you like, and then simply drag and drop another scripts into the /examples directory.

https://maxbox.codeplex.com/

The only thing you need to backup is the ini file **`maxboxdef.ini`** with your history or another root files with settings that have changed.



"Wise men speak: Better late than never.

Feedback @ max@kleiner.com
Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

https://github.com/maxkleiner/maXbox4/releases

https://www.academia.edu/31112544/Work_with_microservice_maXbox_starter48.pdf

1.2 References

Examples and Version of this Tutorial 52:

2017-05 Cumulative Update for Windows 10 Version 1607 for x64-based Systems (KB4019472) – maXbox 4.2.5.10

http://www.softwareschule.ch/examples/766_wmi_management.txt

www.softwareschule.ch/examples/210_public_private_cryptosystem5_ibz_herdt2.txt

Examples of WMI Routines:

WMI Scripting Primer:

https://msdn.microsoft.com/en-us/library/ms974579.aspx

http://www.softwareschule.ch/examples/750_RSA_Toolproof4.txt

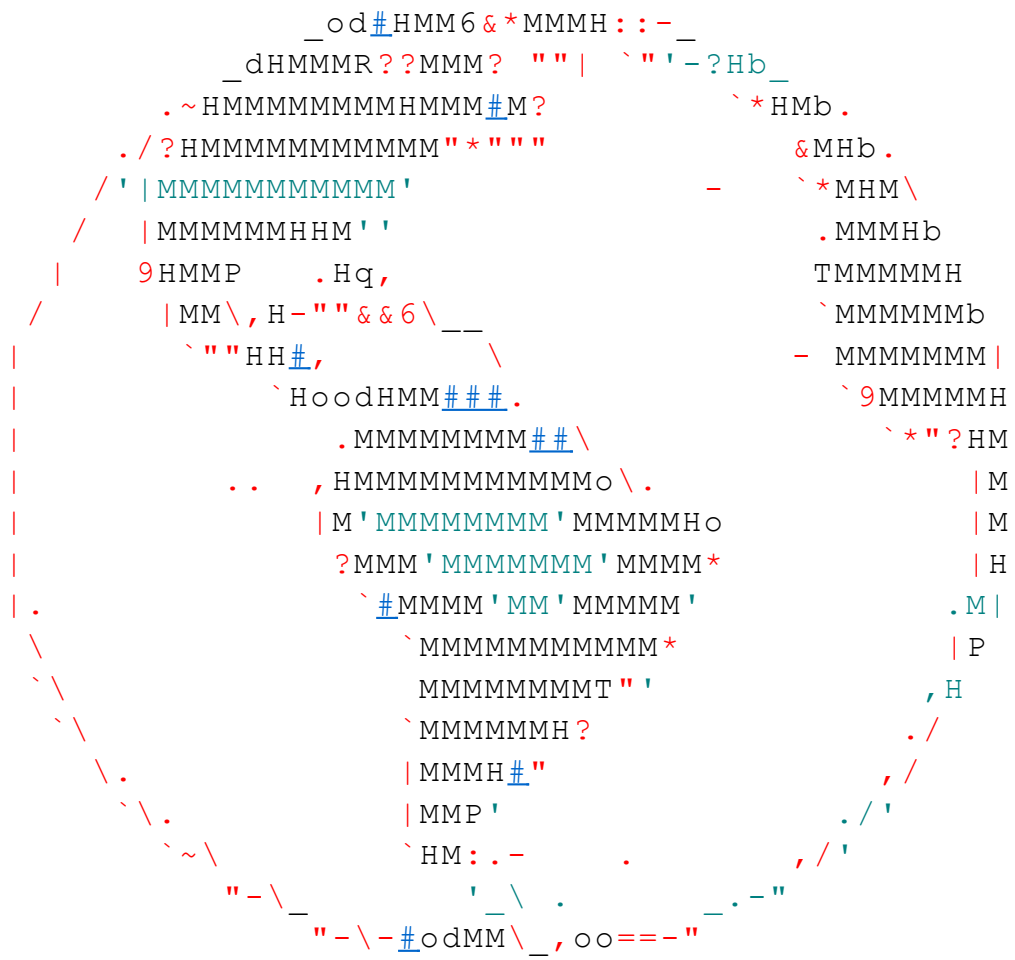
WMI and Database SQL Programming Starter 12 Tutorial and Tutorial 52

http://www.softwareschule.ch/download/maxbox_starter12.pdf

DispInterface explain:

https://msdn.microsoft.com/en-us/library/ebhsdt9x.aspx

http://www.delphidabbler.com/tips/216



Zwei Worte werden Dir im Leben viele Türen öffnen - "ziehen" und "stossen".