1: ///////////////////////////////////////////////////////
2: MaxBox Starter 52 Second Part
3: ///////////////////////////////////////////////////////
4: 
5: 6: Work with WMI System Management V2
7: //////////////////////////////////////////////////////////////////
8: Max Kleiner
9: 
10: 11: In the first available part
12: http://www.softwareschule.ch/download/maxbox_starter52.pdf
13: we got an impression about the force concerning WMI. In this second part we go
deeper with data analysis and filtering.
14: 15: Suppose you need to retrieve records from the Windows event logs for maybe
security reason. With the following script snippet you can easily read event logs
but before you run the listing, we should point out that the example script can
take a long time to run if your event logs contain thousands of records. So we
have to filter it with a clever query:
16: 17: isQuery:= Select * from Win32_NTLogEvent Where Logfile = "System" AND EventType
= "2" AND User = "NT AUTHORITY\NETWORK SERVICE";
18: 19: The filter is based on Logfile, EventType and User.
20: The eventtype is divided into:
21: 22: 1 Error
23: 24: 2 Warning()
25: 26: 3 Information()
27: 28: 4 Security audit success
29: 30: 5 Security audit failure
31: 32: For the logfile you can also choose. Events for the System and Application logs
fall into one of three categories: error, warning, and information. Error events
are the most serious and cause a red stop sign to appear on the left side of the
screen. Warning events identify a possible problem, but not as crucial a problem
as in an error event. Information events are basic notifications, such as
services starting and stopping, browser elections, and print jobs.
33: 34: The Security log uses two event types: success and failure. These events signal
whether a user was able to log on or access a resource. You want the system to
prevent unauthorized users from logging on, so a success event for an
unauthorized user is a problem.
35: Back to code. To get the query running we need a service and a locator:
36: 37: var isloc: ISWbemLocator;
38: isser: ISWbemServices;
39: iset: ISWbemObjectSet;
40: ENum: IEnumVariant;
41: tObj: OleVariant;
42: isQuery: string;
43:
44: isloc:= WMIStart;
45: isser:= WMIConnect(isloc, 'localhost','max',''); // ISWbemServices;
After passing the locator to a connector, we get a service to pass it again to our query to get an object set:

```cpp
isQuery := 'Select * from Win32_NTLogEvent Where Logfile = "System" and EventType= "2" AND User = "NT AUTHORITY\NETWORK SERVICE"';
```

The query with the user notification has to be escaped with \\ otherwise you get an error! This is because the backslash character is an escape character and if you need to put it in a select statement then you need to put it twice i.e.

```cpp
"BUILTIN\Administrators";
```

So the same query work also if the Operating System is Windows Server 2003 Itanium Edition or other win server systems. Next we call the query in a loop to get the object set as a result set:

```cpp
it := 0;
iset := WMIExecQuery(isser, isQuery); //WbemObjectSet;
```

As we set the select query to the wildcard * we get in a tempobj <tobj> as a variant the fields we are interested in:

```cpp
"File: " & wbemObjectLogFile & vbCrLf & _
"Record Number: " & wbemObjectRecordNumber
"Type: " & wbemObjectType
"Time Generated: " & wbemObjectTimeGenerated
"Source: " & wbemObjectSourceName
"Category: " & wbemObjectCategory
"Category String: " & wbemObjectCategoryString
"Event: " & wbemObjectEventCode
"User: " & wbemObjectUser
"Computer: " & wbemObjectComputerName
"Message: " & wbemObjectMessage
```

In our case are 4 fields of interest: TimeGenerated, SourceName, User and Message. And the result looks like (extract):

```cpp
-systime : 20160914194051.572325-000 - Name Microsoft-Windows-DNS-Client User NT AUTHORITY\NETWORK SERVICE
```

Name resolution for the name v10.vortex-win.data.microsoft.com timed out after none of the configured DNS servers responded.

```cpp
-systime : 20160817071325.358494-000 - Name Microsoft-Windows-DNS-Client User NT AUTHORITY\NETWORK SERVICE
```

Name resolution for the name win10.ipv6.microsoft.com timed out after none of the configured DNS servers responded.

```cpp
-systime : 20160814150919.620692-000 - Name Microsoft-Windows-DNS-Client User NT AUTHORITY\NETWORK SERVICE
```
93: Name resolution for the name fs.microsoft.com timed out after none of the
configured DNS servers responded.
94:
95: For each event, the logs show the date and time when the event occurred, as well
as the event source and the user. The source is the service, device driver, or
application that wrote the event to the log. A source can subdivide the events it
writes into multiple categories to let you easily find messages. Each event has an
event ID, which helps MSProduct Support troubleshoot problems.
96:
97: In the end we free the olevariant with:
98:
99: tObj:= unassigned;
100:
101: The script can be found:
102: http://www.softwareschule.ch/examples/766_wmi_management2.txt
103:
104: Now a few hints to late binding with WMI. A Variant variable is always
initialized to Unassigned. You can assign almost any kind of value to the
variable, and it will keep track of the type and value. To learn the type of a
Variant, call the VarType function.
105: The VarType function returns an integer representing the current data type for a
Variant variable <VariantVariable>.
106:
107: When you use variant in an expression, the script automatically converts the
other value in the expression to a Variant and returns a Variant result. You can
assign that result to a statically typed variable, provided the Variant’s type is
compatible with the destination variable.
108:
109: The most common use for Variants is to write an OLE automation client. You can
assign an IDispatch interface to a Variant variable, and use that variable to
call functions the interface declares. The compiler does not know about these
functions, so the function calls are not checked for correctness until runtime.
For example, you can create an OLE client to also print the log events of WMI
installed on your system, as shown in the following code.
110:
111: var
112: wmiLocator: OLEVariant;
113: wmiServices: OLEVariant;
114: wmiObject: OLEVariant;
115: wmiProp: OLEVariant;
116: wmiLocator:= CreateOleObject('WbemScripting.SWbemLocator');
117: wMIServices:= WmiLocator.ConnectServer('localhost', 'root\CIMV2', '', '');
118:
119: Note: if you pass a user name to a localhost you get the exception:
120: {Exception: SWbemLocator: User credentials cannot be used for Local
connections.}
121:
122: Next step is to call the query:
123:
124: WMIProperty:= 'SerialNumber'; WMIClass:= 'Win32_BIOS';
125: WmiObjectSet:= wMIService.ExecQuery(Format('Select %s from %s',
126: [WMIProperty, WMIClass]), 'WQL', wMISFlagForwardOnly);
127: but thats not right way, remember, we drive with variants so pass just strings to
eval!
128:
129: writeln('Set returns: ' + vartostr(WmiObjectSet.count))
134: However I am getting a generic failure on the "WmiObjectSet.Item(0)" line. Is
135: this method of doing things supported, or must I always use the "For Each" method
136: with the collection that is returned from WMI queries? Yes you must or change to
137: early binding before!
138: The Item is separated from the class by a . character. The instance is identified
139: by specifying a property value separated from the property name by a = character.
140: The entire path then looks like \\SYSTEM\NAMESPACE:CLASS.PROPNAME="PROPVALUE".
141: Again, the system and namespace may be defaulted but the class name must be
142: specified.
143: tObj := WmiObjectSet.item('Win32_NetworkAdapterConfiguration.Index=1');
144: The script doesn't know anything about the Version property or any other method
145: or property of the WMI OLE client. Instead, it compiles your property and method
146: references into calls to the IDispatch interface. You lose the benefit of compile-
147: time checks, but you gain the flexibility of runtime binding.
148: If you want to keep the benefits of type safety, you will need a type library
149: from the vendor of the OLE automation server as we did in our example.
150: Note: Authenticated users cannot, by default, log events to the Application log
151: on a remote computer. As a result, the example described in this topic will fail
152: if you use the UNCServerName property of the NETEventLogEventConsumer class and
153: specify a remote computer as its value.
154: At last you can check this by the Event Viewer Tool:
155: The Event Viewer is a tool you use to examine the three NT event logs: System, Security, and Application. The event logs are in the directory
156: \winntroot\system32\config, where winntroot is the directory that houses NT. The three log files are sysevent.evt, secevent.evt, and appevent.evt. You cannot use a regular text editor to view these files.
157: Example of SQL Stored Procedure Code with WMI:
158: EXEC @rc = master.dbo.sp_OAMethod @mwiServices, 'InstancesOf', @wmiObjectSet
159: OUTPUT, 'Win32_NetworkAdapterConfiguration'
160: SELECT @loopIdx = 0
161: WHILE @loopIdx < @wmiObjectCount - 1 BEGIN
162: EXEC @rc= master.dbo.sp_OAMethod @wmiObjectSet,'Item', @wmiObject OUTPUT, @loopIdx
163: IF @rc <> 0 BEGIN
164: EXEC @rc= master.dbo.sp_OAMethod @wmiObjectSet,'Caption',
165: END ELSE BEGIN
166: END
EXEC @rc= master.dbo.sp_OAGetProperty @wmiObject, 'IPAddress', @wmiNetAdapterIP OUTPUT
END
SELECT @loopIdx = @loopIdx + 1
END

-- CLEANUP
EXEC master.dbo.sp_OADestroy @wmiServices
EXEC master.dbo.sp_OADestroy @wmiLocator

---------Variant Test Function---------
var objIE_WMI: variant;
// Show type of a variant
procedure ShowBasicVariantType(varVar: Variant);
var
typStr: string;
basicType: Integer;
begin
// Get the Variant basic type :
// this means excluding array or indirection modifiers
basicType := VarType(varVar) and VarTypeMask;
// Set a string to match the type
begin case basicType of
varEmpty: typStr := 'varEmpty';
varNull: typStr := 'varNull';
varSmallInt: typStr := 'varSmallInt';
varInteger: typStr := 'varInteger';
varSingle: typStr := 'varSingle';
varDouble: typStr := 'varDouble';
varCurrency: typStr := 'varCurrency';
varDate: typStr := 'varDate';
varOleStr: typStr := 'varOleStr';
varDispatch: typStr := 'varDispatch';
varError: typStr := 'varError';
varBoolean: typStr := 'varBoolean';
varVariant: typStr := 'varVariant';
varUnknown: typStr := 'varUnknown';
varByte: typStr := 'varByte';
varWord: typStr := 'varWord';
varLongWord: typStr := 'varLongWord';
varInt64: typStr := 'varInt64';
varStrArg: typStr := 'varStrArg';
varString: typStr := 'varString';
varAny: typStr := 'varAny';
varTypeMask: typStr := 'varTypeMask';
end;
// Show the Variant type
ShowMessage('Variant type is ' + typeString);
end;

http://www.softwareschule.ch/maxbox.htm
Procedure TestVariant3;
  var myVar : Variant;
begin
  // Assign various values to a Variant
  // and then show the resulting Variant type
  ShowMessage('Variant value = not yet set');
  ShowBasicVariantType(myVar);
  // Simple value
  myVar := 123;
  ShowMessage('Variant value = 123');
  ShowBasicVariantType(myVar);
  // Calculated value using a Variant and a constant
  myVar := myVar + 456;
  ShowMessage('Variant value = 123 + 456');
  ShowBasicVariantType(myVar);
  myVar := 'String ' + IntToStr(myVar);
  ShowMessage('Variant value = String 579');
  ShowBasicVariantType(myVar);
end;

Physically, the CIM resides in the %SystemRoot%\system32\wbem\Repository\FS\directory and consists of the following four files:

index.btr. Binary-tree (btree) index file.
index.map. Transaction control file.
objects.data. CIM repository where managed resource definitions are stored.
objects.map. Transaction control file.