**Chapter II**

Understanding CIM and WMI

\_CIM là một chuẩn mở được xác định bởi các Management Distributed Task Force (DMTF) như là một phần của sáng kiến Web-Based Enterprise Management (WBEM). CIM được sử dụng để định nghĩa mô hình dữ liệu mở rộng , quy trình, và có được những thông tin đặc trưng của các nguồn tài nguyên được quản lý như linh kiện phần cứng và phần mềm. CIM là một mô hình lập trình đó là hướng đối tượng và sản xuất độc lập, có nghĩa là bạn có thể quản lý các nguồn tài nguyên khác nhau từ các nhà cung cấp khác nhau bằng cách chỉ sử dụng các tiêu chuẩn CIM.

\_WMI là một Microsoft implementation (thực thi) of CIM được giới thiệu trong Windows 2000 để cho phép quản lý của tất cả các phần mềm và phần cứng Windows các thành phần

**CIM and WMI in Windows PowerShell**

get-command -name \*WMI\* -type cmdlet

* Lệnh lấy các lệnh của WMI có kiểu là cmdlet

get-command \*CIM\* -type cmdlet

* Lệnh lấy các lệnh của CIM có kiểu là cmdlet (command-lets)

**\*WMI là một thực thi của CIM**

**\_** Các lệnh CIM liên quan có nhiều thông số và thậm chí có những cmdlet nhiều hơn trong WMI để cung cấp cho bạn thêm thông tin

\_ CIM và WMI đại diện cho các thông tin được cung cấp trong dưới dạng các namespace và các class

Get-wmiobject –list và get-cimclass => lấy danh sách các class có sẵn trong hệ thống.

(get-wmiobject -list).count –eq (get-cimclass).count => so sánh kết quả của 2 lệnh

get-ciminstance -classname win32-bios và get-ciminstance -classname Win32\_BIOS

* Xác định các tham số của class

\_Sử dụng -query như một tham số thay vì một tên lớp để thực hiện một truy vấn WMI được xác định trước bằng văn bản trong WMI Query Language (WQL).

$Query = "Select \* From Win32\_NetworkAdapter Where Name like '%Intel%'"

Get-WmiObject -Query $Query | Select DeviceID, Name

Hoặc Get-CimInstance -Query $Query | Select DeviceID, Name

Remove-WmiObject và Remove-CimInstance => xóa 1 thể hiện

Set-WmiInstance và Set-CimInstance => update instance

VD: $folder = Get-WmiObject -Class Win32\_Directory -Filter "Name='D:\\HocKy\_5'"

$folder | Remove-WmiObject

**More reasons to adopt CIM**

\_Nó là tiêu chuẩn mở, có nghĩa là nó không bị khóa vào trong Windows hệ thống chỉ hoạt động, do đó bạn có thể sử dụng CIM quản lý nhà cung cấp khác và nhà sản xuất.

\_CIM sử dụng giao thức (WS-MAN) WS-Management để quản lý từ xa để bạn có thể sử dụng nó với bất kỳ máy chủ từ xa hoặc các thiết bị thực hiện điều này giao thức. Tuy nhiên, WMI được sử dụng để quản lý Windows chỉ qua DCOM giao thức.

\_Nó có thể được sử dụng với cơ sở hạ tầng mở Management (KMI) phù hợp thiết bị.

\_Nó có thể được sử dụng để quản lý bất kỳ máy tính hoặc thiết bị với một hệ điều hành có đối tượng quản lý CIM (CIMOM) tuân thủ thực hiện không phân biệt vào nhà cung cấp. Vì vậy, bạn có thể sử dụng để quản lý CIM Windows cũng như không phải hệ điều hành Windows.

Working with XML

\_XML Phân tích và định dạng là một trong những chức năng thường được sử dụng nhất trong phát triển ứng dụng. PowerShell cung cấp được xây dựng hỗ trợ cho XML trong một thông minh cách mà cho phép bạn làm việc với các tập tin XML dễ dàng với dòng tối thiểu của code. Đây là đủ để làm cho PowerShell sự lựa chọn hoàn hảo cho tác vụ XML hàng ngày và hoạt động.

**Loading XML files**

\_Có hai cách để tải một fle XML trong PowerShell-một trong hai cách sử dụng lệnh Get-Content hoặc sử dụng lệnh với các truy vấn XPath Select-Xml.

**Using the Get-Content cmdlet**

Get-Content => Lệnh này sử dụng để tải nội dung từ các văn bản cũng như fles XML, đó là tập tin văn bản được viết trong một định dạng cấu trúc

get-content D:\service.xml => đọc file xml

\_Tải nội dung của các fle XML như văn bản bình thường

$sv = [xml] (get-content D:\service.xml)

Hoặc [xml] $sv = get-content D:\service.xml

\_Các tập tin XML được tải có thông tin về các thành viên trong đội ngũ nhân viên khác nhau ngành có chuyên ngành khác nhau. Bây giờ, chúng tôi có các nội dung của các fle lưu trữ trong một biến gọi là $employees có thể truy cập bình thường như bất kỳ đối tượng khác cùng với các khả năng XML

$employees.staff.ChildNodes => truy cập vào node con của file xml

$employee.staff.branch.Get\_Attributes() => lấy thông tin về thuộc tính của node

$employee.staff.branch. location => lấy giá trị của tên thuộc tính

$employees.staff.branch[0]. Location => thay đổi giá trị của thuộc tính

\_Thay đổi và sửa đổi node đơn

$employee.staff.branch.employee

$emp = $employees.staff.branch.employee

$emp.Role = "PowerShell Guru"

$employee.SelectNodes("//employee[Name='Sherif Talaat']")

\_Thêm node mới

$newemployee = $employee.CreateElement("employee")

$newemployee.set\_InnerXML("<Name>Ahmad Mofeed</Name> <Role>Security Consultant</Role>")

$employee.staff.branch[0].AppendChild($newemployee)

$employee.staff.branch.AppendChild($newemployee)

$employee.staff.branch.employee

**Using the Select-Xml cmdlet**

\_ Lấy dữ liệu từ file XML sử dụng truy vấn XPath

select-Xml -Path D:\employee.xml -XPath "staff/branch/employee"

\_Để mở rộng các node và liệt kê các giá trị của chúng, chúng ta phải sử dụng Select-Object lệnh với tham số -ExpandProperty

select-Xml -Path D:\employee.xml -Xpath "staff/branch/employee" | select-object –expandProperty node

**Importing and exporting XML files**

\_Lệnh Export-CliXml lệnh để xuất khẩu các đối tượng (s) vào một tập tin XML và Import- CliXml nhập và tải các tập tin mà được xuất trước đó của PowerShell

\_Xuất đối tượng từ tập tin XML

Get-Service | Export-Clixml D:\Services.xml

\_Nhập đối tượng vào tập tin XML

Import-Clixml D:\Services.xml -First 5

\_Lệnh ConvertTo- Xml tương tự như lệnh ExportCliXml, nơi mà cả các lệnh tạo ra một đại diện XML của một hoặc nhiều các đối tượng. Sự khác biệt duy nhất là các lệnh **Export- CliXml** lưu trữ các mã XML trong một file khi lệnh **ConvertTo- Xml** trả về một đối tượng XML có thể được sử dụng như một đầu vào cho lệnh khác.

**Creating an instance of a COM object**

\_Để tạo ra một thể hiện của một đối tượng COM, chúng ta sử dụng lệnh **New-Object** với tham số **-ComObject** và **ProgID** như một tham số, nơi ProgID là tên thân thiện của class COM sử dụng trong quá trình đăng ký class

$com = New-Object –ComObject <ProgID>

VD: $com = New-Object –Comobject Word.Application

**Automating Internet Explorer with COM and PowerShell**

\_Tạo đối tượng mới của lớp IE COM

$ie = New-Object -ComObject InternetExplorer.Application

$ie.navigate("about:blank")  
$ie.height = 800  
$ie.width = 1200  
$ie.visible = $true

\_Nhập ID và Password

$EmailAddress = Read-Host -Prompt "Enter your Microsoft Account.."

$Password = Read-Host -AsSecureString -Prompt "Enter your Password..."

**Automating Microsoft Excel with COM and PowerShell**

\_Tạo ra một thể hiện của Excel.Application

$Excel = New-Object -ComObject Excel.Application

\_Khởi động Excel

$Excel.visible = $True

\_Tạo ra Excel workbook mới, và thêm một worksheet cho nó.

$ExcelWB = $Excel.Workbooks.Add()  
$ExcelWS = $ExcelWB.Worksheets.Item(1)

\_Tạo một tiêu đề cho báo cáo trong hai ô đầu tiên trong hàng đầu tiên

$ExcelWS.Cells.Item(1,1) = "Services Status Report"  
$ExcelWS.Range("A1","B1").Cells.Merge()

\_ Chúng tôi sẽ tạo ra một tiêu đề cho bảng với hai cột: **Services Name** và **Service Status** ở hàng thứ hai của bảng tính.

$ExcelWS.Cells.Item(2,1) = "Services Name"  
$ExcelWS.Cells.Item(2,2) = "Service Status"

\_ Liệt kê tất cả các dịch vụ Windows bằng cách sử dụng lệnh Get-Service và lặp trên danh sách sử dụng vòng lặp ForEach để tạo ra một hàng mới cho mỗi dịch vụ trong danh sách.

$row = 3

ForEach($Service in Get-Service)

{

$ExcelWS.Cells.Item($row, 1) = $Service.DisplayName

$ExcelWS.Cells.Item($row, 2) = $Service.Status.ToString()

if($service.Status -eq "Running")

{

$ExcelWS.Cells.Item($row,1).Font.ColorIndex = 10

$ExcelWS.Cells.Item($row,2).Font.ColorIndex = 10

}

Elseif($Service.Status -eq "Stopped")

{

$ExcelWS.Cells.Item($row,1).Font.ColorIndex = 3

$ExcelWS.Cells.Item($row,2).Font.ColorIndex = 3

}

$row++

}

\_Lưu báo cáo và thoát khỏi dụ Excel.

$ExcelWS.SaveAs("D:\ServicesStatusReport.xlsx")  
$Excel.Quit()

**Working with .NET objects**

**Creating .NET objects**$date = New-object -typeName System.DateTime -ArgumentList 2013, 10, 24

\_ Bạn có thể xác định loại đối tượng trực tiếp mà không cần sử dụng –TypeName

$string = new-object System.String -ArgumentList "PowerShell Rocks!"

VD:

[datetime] $date = "2013/10/24"

[string] $string = "Powershell Rocks!"

**Extending .NET objects**

\_Load XML file  
[xml] $xml = Get-Content D:\Employees.xml   
\_Add new NoteProperty Member using Add-Member  
Add-Member -InputObject $xml -MemberType NoteProperty –Name Description -Value "Employees information database"   
\_Show the new added member  
$xml | Get-Member -MemberType NoteProperty | fl

VD: Thêm một phương thức tùy chỉnh của các loại ScriptMethod

\_Creating an array of websites URLs  
$websites = @("facebook.com","twitter.com","google.com","xxx.com")  
\_Add new ScriptMethod member to the array object  
Add-Member -InputObject $websites -MemberType ScriptMethod -Name Censored -Value {$this -replace "xxx","\*\*\*"}  
\_Execute the new added method  
$websites.Censored()

**Extending .NET framework types**

\_Windows PowerShell cho phép bạn định nghĩa các kiểu framework .NET (classes) để bạn có thể tạo các đối tượng của những class sau này bằng cách sử dụng lệnh New-Object. Những loại đó có thể được được xác định bởi tập tin mã nguồn, tập tin assembly hoặc thậm chí là một inline code của C #, VB, và JScript.  
**Defining the object type using an inline C# class**

\_Ví dụ sau đây sẽ cho thấy làm thế nào để tạo ra một loại đối tượng mới từ một inline class C #. Đầu tiên, chúng tôi sẽ tạo ra một định nghĩa lớp cho một máy tính cơ bản với bốn phương pháp đại diện cho bốn phép tính số học. Sau đó, chúng tôi sẽ sử dụng lệnh Add-Type lệnh để thêm lớp mới vào phiên PowerShell hiện hành. Cuối cùng, chúng ta sẽ tạo ra một đối tượng của lớp này bằng cách sử dụng lệnh New-Object.

$myCalc = @”  
public class PSCalc  
{  
public int Add(int x, int y) {return x + y;}  
public int Subtract(int x, int y) {return x – y;}  
public int Multiply(int x, int y) {return x \* y;}  
public int Divid(int x, int y) {return x / y;}  
}  
”@  
PS C:\> Add-Type –TypeDefinition $myCalc  
PS C:\> $op = New-Object PSCalc

$op.Multiply(4,8)

\_Bạn có thể thực hiện một phương thức static của một class trực tiếp trong PowerShell. Ví dụ, lớp System.Math có nhiều phương thức static; một trong chúng được gọi là Pow (). Để gọi phương pháp này từ PowerShell, bạn nên viết như sau:

[system.Math]::Pow(2, 4)

\*\*

Add-type

Adds a Microsoft .NET Framework class in a Windows PowerShell session.

\_Sử dụng Add-Type cmdlet để thêm các lớp học để phiên. Bởi vì nó được sử dụng mã nguồn inline các lệnh sử dụng các tham số TypeDefinition để xác định code trong biến $ Source.

\*\*

**Defining an object type using an assembly name or file**

\_Một cách khác để xác định một loại đối tượng mới được sử dụng assembly (namespace) hoặc các file assembly (DLL), và sau đó sử dụng lệnh New-Object để tạo ra các đối tượng. Ví dụ sau sẽ hiển thị như thế nào chúng ta có thể thêm một loại mới sử dụng assembly namespace

$form = New-Object System.Windows.Forms

**Understanding Windows PowerShell Modules**

\_Windows PowerShell Module là cách thức tổ chức và đóng gói kịch bản PowerShell và tập tin code thành các đơn vị được chia và tái sử dụng. Windows PowerShell đi kèm với một số lượng khá tốt của module được xây dựng trong đó cung cấp cho cmdlet hầu như tất cả các role Windows Server và các tính năng. Ví dụ, có một module cho Server Manager, Hyper-V, Active Directory, và IIS.

\_Để liệt kê tất cả các mô-đun có sẵn cài đặt trên các hệ điều hành, chúng tôi sử dụng  
cmdlet Get-Module với các thông số -ListAvailable:

Get-module -listAvailable | select name, version, moduletype

\_ Để bắt đầu sử dụng bất kỳ module hoặc một built-in của bên thứ ba mô-đun-bạn phải sử dụng

Import-Module <MODULE\_NAME>

Import-Module -Name SQLPS