Getting to know the Windows 10 MDM WMI Bridge provider

Peter van der Woude

V-Platinun Sponsor

Patch My PC
PATCH MANAGEMENT MADE EASY

RECAST SOFTWARE

V-Gold Sponsor

liquidware™

baseVISION
SECURE & MODERN WORKPLACE

2Pint Software

PolicyPak™
SECURING YOUR STANDARDS.

Event Sponsors

SmartIT

make it noble

RealStuff
Informatik AG

baggenstos.

About Peter van der Woude

Focus

Modern Workplace

From

Groningen, Netherlands

My Blog

https://petervanderwoude.nl

Microsoft MVP

Enterprise Mobility

Certifications

Microsoft 365 Certified: Enterprise Administrator Expert

Microsoft 365 Certified: Modern Desktop Administrator Associate

Hobbies

Family

Basketball

Gaming

Contact

pvanderwoude@hotmail.com

@pvanderwoude

/peterwoude

Agenda

www.wpninjas.eu

Introduction to Windows 10 MDM

The basics of Windows 10 MDM

Windows 10 MDM policies

What are Windows 10 MDM policies

Windows 10 MDM troubleshooting

How can we troubleshoot Windows 10 MDM

Windows 10 MDM WMI Bridge provider

How can we locally interact with Windows 10 MDM

Windows 10 MDM scripting

How can we script against Windows 10 MDM

Key takeaways:

- **Getting familiar with Windows 10 MDM**
- **Getting to know the Windows 10 MDM WMI Bridge provider**
- **Starting to use the Windows 10 MDM WMI Bridge provider**

Introduction to Windows 10 MDM

The basics of Windows 10 MDM

Short introduction to Windows 10 MDM

- Windows 10 MDM is based on **Open Mobile Association** (OMA) **Device Management** (DM)
- Windows 10 MDM uses the **Synchronization Markup Language** (SyncML) representation protocol to exchange data between client and server
- Windows 10 MDM uses **Configuration Service Providers** (CSP) to expose device configuration settings
  - Path to a CSP and setting is the **Open Mobile Alliance Uniform Resource Identifier** (OMA-URI)

Windows 10 MDM policies

What are Windows 10 MDM policies

Windows 10 MDM policies

- The starting point for **Windows 10 MDM policies** is the configuration service provider reference
- Policies in the Policy CSP can be configured on **User** scope and **Device** scope
  - ./[User]/Vendor/MSFT/Policy/[AreaName]
  - ./[Device]/Vendor/MSFT/Policy/[AreaName]
  - ./Vendor/MSFT/Policy/[AreaName]
- Policies in the Policy CSP have a different **Config** and **Result** path
- Policies in the Policy CSP can support the **Add**, **Get**, **Delete** and **Update** operations
  - Other CSPs can also support operations like **Execute**
- Some policies in the Policies CSP are **backed** by existing **ADMX**-settings
- Third-party and/or custom **ADMX**-files can be **ingested**

- Constructing an OMA-URI by using the configuration service provider reference

Windows 10 MDM policy refresh

- Starting with Windows 10, version 1903, policies of the Policy CSP are refreshed during check-in
- **A notification** – A check-in can be triggered by a notification from Microsoft Intune
- **A manual check-in** – A check-in can be triggered manually by the user
  - Settings panel
  - Company Portal app
- **A scheduled check-in** – A check-in can be triggered by a scheduled task

Schedule	Frequency
Schedule #1 created by the enrollment client	After triggered, repeat every 3 minutes for a duration of 15 minutes
Schedule #2 created by the enrollment client	After triggered, repeat every 15 minutes for a duration of 2 hours
Schedule #3 created by the enrollment client	After triggered, repeat every 8 hours indefinitely

- See also - https://www.petervanderwoude.nl/post/windows-10-mdm-policy-refresh/

- Looking at the policy refresh and the check-ins

Windows 10 MDM troubleshooting

How can we troubleshoot Windows 10 MDM

Windows 10 MDM troubleshooting

www.wpninjas.eu

- Event Viewer (Microsoft-Windows-DeviceManagement-Enterprise-Diagnostics-Provider/Admin)
- Advanced Diagnostics Report (Settings)
- MDM Diagnostics Tool (Settings and manual)
  - Different standard areas
    - Autopilot
    - DeviceEnrollment
    - DeviceProvisioning
    - TPM
  - Output
    - Event (trace) logs
    - Registry keys
    - HTML report
  - Example usage: MdmDiagnosticsTool.exe -out <output folder path>
- MDM Diagnostics Tool can be used remotely with the DiagnosticLog CSP to save the output in the cloud
- See also - https://www.petervanderwoude.nl/post/windows-10-mdm-troubleshooting/
- And - https://www.petervanderwoude.nl/post/triggering-devices-to-upload-diagnostic-files-to-cloud-storage/

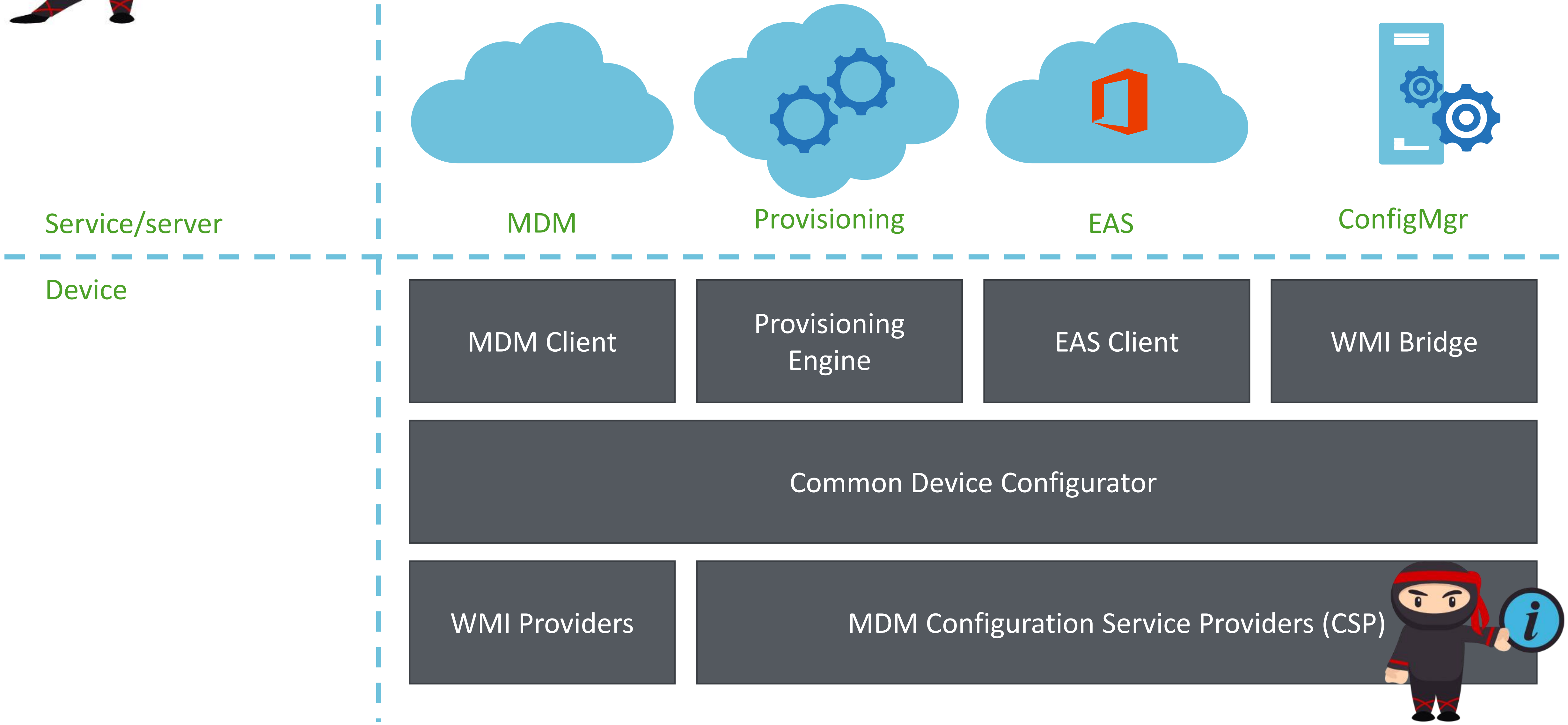
- Looking at the different troubleshooting areas

Windows 10 MDM WMI Bridge provider

How can we locally interact with Windows 10 MDM

Configuration options

www.wpninjas.eu



Windows 10 MDM WMI Bridge provider

- WMI Providers are the backbone of WMI
- MDM WMI Bridge Provider enables mobile device management
- MDM WMI Bridge Provider is defined in **DMWmiBridgeProv.mof** and implemented in **DMWmiBridgeProv.dll**
- MDM WMI Bridge Provider uses the namespace **\root\cimv2\mdm\dmmap**
- MDM WMI Bridge Provider creates classes for the different Windows 10 CSPs
- My minimal toolkit for WMI: **WMI Explorer** and/or **Wbemtest**

- Be familiar with the MDM Bridge WMI provider doc
- Exploring WMI and looking at the different instances and methods

Windows 10 MDM scripting

How can we script against Windows 10 MDM

Windows 10 MDM scripting

- Nothing more, nothing less than scripting against a WMI provider
- *-Wmi* versus *-Cim*
  - WMI is based on the **Common Information Model** (CIM) standard
  - Introduction of **Windows Remote Management** (WinRM) also known as PSRemoting
  - WinRM uses the **Web Services for Management** (WS-Man) protocol for data transfer
  - New cmdlets take advantage of WinRM and are based on the CIM standard
  - Even for local use it’s old versus new
- My minimal toolkit for scripting: Visual Studio Code, GitHub Desktop, Windows Terminal and PSExec

- See also - https://www.petervanderwoude.nl/post/windows-10-mdm-powershell-scripting/

Old versus new

- Get-WmiObject
- Set-WmiInstance
- Invoke-WmiMethod
- Register-WmiEvent
- Remove-WmiObject

- Get-CimAssociatedInstance
- Get-CimClass
- **Get-CimInstance**
- Get-CimSession
- **Set-CimInstance**
- **Invoke-CimMethod**
- **New-CimInstance**
- New-CimSession
- New-CimSessionOption
- Register-CimIndicationEvent
- **Remove-CimInstance**
- Remove-CimSession

Windows 10 MDM scripting

- Nothing more, nothing less than scripting against a WMI provider
- *-Wmi* versus *-Cim*
  - WMI is based on the **Common Information Model** (CIM) standard
  - Introduction of **Windows Remote Management** (WinRM) also known as PSRemoting
  - WinRM uses the **Web Services for Management** (WS-Man) protocol for data transfer
  - New cmdlets take advantage of WinRM and are based on the CIM standard
  - Even for local use it’s old versus new
- My minimal toolkit for scripting: Visual Studio Code, GitHub Desktop, Windows Terminal and PSExec

- See also - https://www.petervanderwoude.nl/post/windows-10-mdm-powershell-scripting/

- Scripting with the MDM WMI Bridge provider

Quick summary

- To properly use the Windows 10 MDM WMI Bridge provider …
  - … be familiar with the CSPs
  - … understand the WMI provider
  - … and connect the dots
- It’s “just” WMI and PowerShell
- Remember: SYSTEM context

Adjusting Start

```
#Declare variables
$namespaceName = 'root\cimv2\mdm\dmmap'
$className = 'MDM_Policy_Config01_Start02'
$parentID = './Vendor/MSFT/Policy/Config' #ParentID must not contain the scope
to create a new instance
$instanceID = 'Start'


#Declare additional variables
$configureProperty = 'HideSleep'
$valueProperty = [int]1
$startTime = Get-Date


#Create a new instance
New-CimInstance -Namespace $namespaceName -ClassName $className -Property @{
InstanceID=$instanceID; ParentID=$parentID; $configureProperty=$valueProperty }
```

Get status events

```
#Verify status of the new instance

$eventParentID = $parentID.Replace('.','./Device') #ParentID must contain the
scope to find relevant event

Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-DeviceManagement-
Enterprise-Diagnostics-Provider/Admin'; StartTime=$startTime;
Message1=$configureProperty }

Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-DeviceManagement-
Enterprise-Diagnostics-Provider/Admin'; StartTime=$startTime;
Message1="$eventParentID/$instanceID/$configureProperty" }
```

Schedule a reboot

```
#Declare variables
$namespaceName = 'root\cimv2\mdm\dmmap'
$className = 'MDM_Reboot_Schedule01'
$parentID = './Vendor/MSFT/Reboot'
$instanceID = 'Schedule'

#Declare additional variables
$configureProperty = "Single"
$valueProperty = "" #"2019-10-01T22:00:00Z"

#Get a specific instance
$instanceObject = Get-CimInstance -Namespace $namespaceName -ClassName $className -Filter
"ParentID='$parentID' and InstanceID='$instanceID'"

#Adjust a specific property
$instanceObject.$configureProperty = $valueProperty

#Modify an existing instance
Set-CimInstance -CimInstance $instanceObject
```

Trigger a reboot

www.wpninjas.eu

```
#Declare standard variables
$namespaceName = 'root\cimv2\mdm\dmmap'
$className = 'MDM_Reboot'
$parentID = './Vendor/MSFT'
$instanceID = 'Reboot'
$methodName = 'RebootNowMethod'

#Get a specific instance
$instanceObject = Get-CimInstance -Namespace $namespaceName -ClassName
$className -Filter "ParentID='$parentID' and InstanceID='$instanceID'"

#Trigger specific method
Invoke-CimMethod -InputObject $instanceObject -MethodName $methodName
```

Upgrade Windows edition with productkey

```
#Declare standard variables
$namespaceName = 'root\cimv2\mdm\dmmap'
$className = 'MDM_WindowsLicensing'
$parentID = './Vendor/MSFT'
$instanceID = 'WindowsLicensing'
$methodName = 'UpgradeEditionWithProductKeyMethod'

#Declare additional variables
$configureProperty = 'param'
$valueProperty = 'NPPR9-FWDCX-D2C8J-H872K-2YT43'  #Public KMS Client Key for Windows 10 
Enterprise

#Get a specific instance
$instanceObject = Get-CimInstance -Namespace $namespaceName -ClassName $className -Filter 
"ParentID='$parentID' and InstanceID='$instanceID'"

#Trigger specific method
Invoke-CimMethod -InputObject $instanceObject -MethodName $methodName -Arguments 
@{$configureProperty = $valueProperty}
```

Template for adjusting a setting (1)

```
function Update-PolicySetting {
    param (
        [Parameter(Mandatory=$true)]$className,
        [Parameter(Mandatory=$true)]$parentID,
        [Parameter(Mandatory=$true)]$instanceID,
        [Parameter(Mandatory=$false)]$configureProperty,
        [Parameter(Mandatory=$false)]$valueProperty,
        [Parameter(Mandatory=$false)][Switch]$removeInstance
    )
    try {
        #Get a specific instance
        $instanceObject = Get-CimInstance -Namespace 'root\cimv2\mdm\dmmap' -ClassName $className -Filter "ParentID='$parentID' and InstanceID='$instanceID'" -ErrorAction Stop
    }
    catch {
        Write-Host $_ | Out-String
    }

    if ($removeInstance -eq $false) {
        if ($PSBoundParameters.ContainsKey('configureProperty') -and ($PSBoundParameters.ContainsKey('valueProperty'))) {
            if ($null -eq $instanceObject) {
                try {
                    #Create a new instance
                    New-CimInstance -Namespace 'root\cimv2\mdm\dmmap' -ClassName $className -Property @{ InstanceID=$instanceID; ParentID=$parentID; $configureProperty=$valueProperty } -ErrorAction Stop
                    Write-Output "Successfully created the instance of '$instanceID'"
                }
                catch {
                    Write-Host $_ | Out-String
                }
            }
            else {
                try {
                    #Adjust a specific property
```

Template for adjusting a setting (2)

```
                   $instanceObject.$configureProperty = $valueProperty


                    #Modify an existing instance
                    Set-CimInstance -CimInstance $instanceObject -ErrorAction Stop
                    Write-Output "Successfully adjusted the instance of '$instanceID'"
                }
                catch {
                    Write-Host $_ | Out-String
                }
            }
        }
        else {
            Write-Output ">> Make sure to provide a value for configureProperty and valueProperty when creating or adjusting an instance <<"
        }
    }
    elseif ($removeInstance -eq $true) {
        if ($null -ne $instanceObject) {
            try {
                #Remove a specific instance
                Remove-CimInstance -InputObject $instanceObject -ErrorAction Stop
                Write-Output "Successfully removed the instance of '$instanceID'"
            }
            catch {
                Write-Host $_ | Out-String
            }
        }
        else {
            Write-Output "No instance available of '$instanceID'"
        }
    }
}
```

Thank You