מבוא לאוטומציה באמצעות POWERSHELL

חלק א- הכרות בסיסית עם PS

כמנהלי רשת מבוססת שרתי מייקרוסופט אנו נתקלים במשימות החוזרות על עצמן שוב ושוב , בקשות משתמשים שיכולות היו בקלות להתבצע בעצמן . מטלות אילו גוזלות מאיתנו זמן עבודה רב ופוגעות ביעילות העבודה שלנו .

בעולם שפות הסקריפטים של מייקרוסופט עבדנו עם CMD ו-BATCH ויש כאלו שהתנסו עם VBSCRIPT שתי הטכנולוגיות הללו נחשבות מיושנות כעת היות ומבחינת מייקרוסופט כולנו עוברים ל POWERSHELL.

למנהלי הרשת היותר וותיקים זכורה בוודאי מע' ההפעלה DOS שהביאה לנו את COMMAND.COM , הממשק היחידי שהיה לנו היה מבוסס טקסט וכשהיינו רוצים לבצע פקודות ברצף היינו כותבים סקריפט באמצעות קבצי BATCH

ווינדוס הייתה בשורה מרעננת שכן היא הביאה עמה ממשק גרפי מתקדם אבל מייקרוסופט לא זנחה את ממשק הפקודה והביאה לנו את cmd.exe ומבחר כלים שימושיים כמו netsh ועוד . למתקדמים היו גם כלי sysinternals המעולים שעליהם נכתוב במאמר אחר .

בהמשך שחררה מיקרוסופט את VBscript על מנת להחליף את קבצי BATCH , השפה הזו ידעה להתממשק לרכיבי COM של מע' ההפעלה ולאפשר גמישות ופונקציונליות רבה יותר. השפה הייתה שימושית אך לא מתועדת מספיק ולא נתמכת בצורה אחידה במוצרי מייקרוסופט.את החלל שלא הצליחה למלא VBScript הצליחה POWERSHELL למלא , ביוני 2005 שוחררה גירסת הבטא של מה שבאפריל 2006 נקרא Windows Powershell , מווינדוס 7 היא מובנת במע' .

היתרון המשמעותי של PS הוא בכך שהשפה מבוססת על NET. שהיא השפה שבה כתובה מע' ההפעלה.

כמו כן חשוב להבין שהשפה של PS היא Object Oriented – כלומר מונחת עצמים

הסבר קצר : תכנות מונחה עצמים

השיטה של תכנות מונחה עצמים (אובייקטים) הינה למעשה, חיקוי של חשיבת האדם, כלומר, היא אינטואיטיבית וקרובה יותר לדרך החשיבה של רוב האנשים המטפלים בבעיה העומדת בפניהם.
אנו באופן טבעי ואוטומטי מסווגים את כל העצמים על פי קטגוריות שונות. לדוגמא, העולם הפיזי מחולק על ידנו לחי, צומח ודומם. קיימת גם חלוקה פנימית יותר ובה את החי אנו מחלקים ליונקים, עופות, דגים וכך הלאה לחלוקות פנימיות יותר. כל עצם או מושג המוכר לנו בעולם הזה משתייך למספר רב של קטגוריות בסדר היררכי כלשהו.
קל ונח יותר לשלוט בכמות האדירה של האינפורמציה בה אנו מוצפים וכך יעיל ונח יותר לסדר אינפורמציה זו במוחנו, בכדי שנוכל לשלפה ולזהותה בקלות בכל עת שנחפוץ.

נראה דוגמא – בבואנו לספר לחבר כי רכשנו רכב חדש אין צורך לפרט לו כי רכשנו מנוע, קרבורטור(מאייד), מצמד, דוושת דלק, רדיאטור, הגה וכו'.... אלא בעצם הגדרת המושג רכב יידע החבר כי כל אלו נכללים, ומכאן שאין צורך בפירוט מצדנו וזה יתרון עצום! אחרת עבור כל רכב היינו צריכים לציין את כל מרכיביו, בעוד שבזכות העובדה כי אנו מסווגים ומקטלגים את הדברים כל שנצטרך לציין הן התכונות המייחדות את הרכב שלנו ולא את המשותפות לכלל הרכבים.

תכונה נוספת המאפיינת את תהליך החשיבה של האדם הינה – שיוך של פעולה לעצם כלשהו. למשל, נשווה בין שתי הפעולות הבאות: הדלקת נר והדלקת המחשב – לכאורה, מבחינה לוגית משמעותן של שתי הפעולות זהה – שתיהן פעולות "הדלקה", אך הנגזרת של כל פעולה היא מימוש פיזי אחר – כלומר, סידרת הצעדים אשר נבצע שונה לחלוטין בין פעולה אחת לאחרת.

ומכאן כשנתבקש לבצע פעולת "הדלקה" אנו נדע איזו פעילות עלינו לבצע על פי שיוך של הפעולה לעצם אותו אנו מתבקשים להדליק. כלומר, קיים קשר חשוב בין הפעולה לבין העצם עליו או איתו היא מתבצעת.

שני התהליכים החשיבתיים שהגדרנו אצל בני האדם מיושמים בצורה דומה בגישה מונחית העצמים ע"י יצירת טיפוס חדש המכונה Class (מחלקה) המאפשר ריכוז של מאפיינים (משתנים) ומטודות (פונקציות) תחת הגדרה אחת.

לאחר הגדרת המחלקה ניתן לייצר מופעים שלה, כל מופע של מחלקה קרוי Object (אובייקט). כאשר המחלקה הינה תבנית ליצירת אובייקטים מסוגה, כלומר, המחלקה מהווה את ההגדרה של המשתנים והשיטות שיכילו אח"כ כל מופעיה (האובייקטים מסוגה).

דוגמא מעשית ליכולות של PS בנושא עבודה עם אובייקטים :

בתרגיל הבא נפנה למחלקה האחראית על יצירת הדיבור במחשב ונסקור את הפרמטרים ואת הפעולות שהיא יודעת לבצע

ובאמצעות פקודות PS נגרום למחשב לדבר. הדוגמא הזו תמחיש את היכולת של PS לקרוא לכל מחלקה שקיימת במע' ההפעלה ולבצע פקודות ישירות עליה. כלומר PS משוחררת מכל מגבלה ומסוגלת לבצע כל פעולה אפשרית על מע' ההפעלה. אם בעולם הישן היינו מוגבלים בפקודות CMD לקוד שנכתב עבור הפקודה כעת יש לנו חופש מוחלט לבצע כל מה שנרצה במע' ההפעלה אם רק נכיר את המחלקות והאובייקטים הנכונים.

נפתח ממשק PS ונקליד את הפקודה הראשונה :

```
Add-Type -AssemblyName System.speech
```

```
PS C:\Users\Administrator> Add-Type -AssemblyName System.speech
PS C:\Users\Administrator>
```

למעשה אנו טוענים רכיב DLL הנמצא במע' ההפעלה ומכיל את המחלקה

https://docs.microsoft.com/en-us/dotnet/api/system.speech.synthesis.speechsynthesizer?view=netframework-4.8

SpeechSynthesizer Class

Namespace: System.Speech.Synthesis

Assembly: System.Speech.dll

Provides access to the functionality of an installed speech synthesis engine.

בשלב הבא נרצה ליצור משתנה שיכיל את האובייקט אותו טענו לזיכרון :

```
$speak = New-Object System.Speech.Synthesis.SpeechSynthesizer
```

```
$speak = New-Object System.Speech.Synthesis.SpeechSynthesizer
```

על מנת לבחור את האובייקט שיצרנו נרשום את הפקודה הבאה :

```
$speak | Get-Member
```

רשמנו את המשתנה ובאמצעות PIPE הסימן : | כתבנו את הפקודה Get-MEMBER או GM בקיצור על מנת לראות את כל המאפיינים והפעולות של האובייקט

```
PS C:\Windows\System32\WindowsPowerShell\v1.0\en-US> $speak | Get-Member

   TypeName: System.Speech.Synthesis.SpeechSynthesizer

Name                          MemberType Definition
----                          ---------- ----------
BookmarkReached               Event      System.EventHandler`1[System.Speech.Synthesis.BookmarkReachedEventArgs]
PhonemeReached                Event      System.EventHandler`1[System.Speech.Synthesis.PhonemeReachedEventArgs] P
SpeakCompleted                Event      System.EventHandler`1[System.Speech.Synthesis.SpeakCompletedEventArgs] S
SpeakProgress                 Event      System.EventHandler`1[System.Speech.Synthesis.SpeakProgressEventArgs] Sp
SpeakStarted                  Event      System.EventHandler`1[System.Speech.Synthesis.SpeakStartedEventArgs] Spe
StateChanged                  Event      System.EventHandler`1[System.Speech.Synthesis.StateChangedEventArgs] Sta
VisemeReached                 Event      System.EventHandler`1[System.Speech.Synthesis.VisemeReachedEventArgs] Vi
VoiceChange                   Event      System.EventHandler`1[System.Speech.Synthesis.VoiceChangeEventArgs] Voic
AddLexicon                    Method     void AddLexicon(uri uri, string mediaType)
Dispose                       Method     void Dispose(), void IDisposable.Dispose()
Equals                        Method     bool Equals(System.Object obj)
GetCurrentlySpokenPrompt      Method     System.Speech.Synthesis.Prompt GetCurrentlySpokenPrompt()
GetHashCode                   Method     int GetHashCode()
GetInstalledVoices            Method     System.Collections.ObjectModel.ReadOnlyCollection[System.Speech.Synthesi
GetType                       Method     type GetType()
Pause                         Method     void Pause()
RemoveLexicon                 Method     void RemoveLexicon(uri uri)
Resume                        Method     void Resume()
SelectVoice                   Method     void SelectVoice(string name)
SelectVoiceByHints            Method     void SelectVoiceByHints(System.Speech.Synthesis.VoiceGender gender), voi
SetOutputToAudioStream        Method     void SetOutputToAudioStream(System.IO.Stream audioDestination, System.Sp
SetOutputToDefaultAudioDevice Method     void SetOutputToDefaultAudioDevice()
SetOutputToNull               Method     void SetOutputToNull()
SetOutputToWaveFile           Method     void SetOutputToWaveFile(string path), void SetOutputToWaveFile(string p
SetOutputToWaveStream         Method     void SetOutputToWaveStream(System.IO.Stream audioDestination)
Speak                         Method     void Speak(string textToSpeak), void Speak(System.Speech.Synthesis.Promp
SpeakAsync                    Method     System.Speech.Synthesis.Prompt SpeakAsync(string textToSpeak), void Spea
SpeakAsyncCancel              Method     void SpeakAsyncCancel(System.Speech.Synthesis.Prompt prompt)
SpeakAsyncCancelAll           Method     void SpeakAsyncCancelAll()
SpeakSsml                     Method     void SpeakSsml(string textToSpeak)
SpeakSsmlAsync                Method     System.Speech.Synthesis.Prompt SpeakSsmlAsync(string textToSpeak)
ToString                      Method     string ToString()
Rate                          Property   int Rate {get;set;}
State                         Property   System.Speech.Synthesis.SynthesizerState State {get;}
Voice                         Property   System.Speech.Synthesis.VoiceInfo Voice {get;}
Volume                        Property   int Volume {get;set;}
```

אנו יכולים לראות שהאובייקט מחולק למאפיינים , Method (פעולות) ו-Evnet (פעולות שמבצעות רישום או תגובה על מנת להפעיל טריגר)

נוכל להבחין שיש פה שיטה (פעולה) שנקראת Speak שלוקחת מחרוזת ומוציאה ממנו קול :

```
Speak                         Method     void Speak(string textToSpeak), void Speak(System.Speec...
```

```
$speak.Speak("This is PowerShell says Hello")
```

נכתוב את הפקודה ונוכל לשמוע דרישת שלום מהמחשב

```
PS C:\Users\Administrator> $speak.Speak("This is PowerShell says Hello")
PS C:\Users\Administrator>
```

כמה משחקים קטנים :

```
$speak.voice
```

יראה לנו את הקול הנוכחי באמצעותו מופעל הדיבור

```
$speak.GetInstalledVoices().VoiceInfo
```

יראה לנו את הקולות המותקנים במחשב

```
$speak.SelectVoice('Microsoft Zira Desktop')
```

יאפשר לנו לבחור קול אחר מהרשימה

מה למדנו מהדוגמא הזו ?

למדנו שPS יודעת לבצע קריאה לכל אובייקט שנמצא במחשב לא משנה אם זה אאוטלוק או AD

ראינו שכל אובייקט מורכב ממחלקות וכל מחלקה מכילה פרמטרים שונים ופעולות , נוכל לגשת למידע הזה באמצעות הפקודה Get-Member או GM בקיצור.

PS מכילה מראש המון פקודות אשר חוסכות מאיתנו את הצורך לגשת לאובייקט ולבצע עליו פעולה בצורה ישירה , לדוגמא הפקודה SET-ADuser מבצעת שינויים הקשורים למשתמש על האובייקט המנהל את בסיס הנתונים של AD .

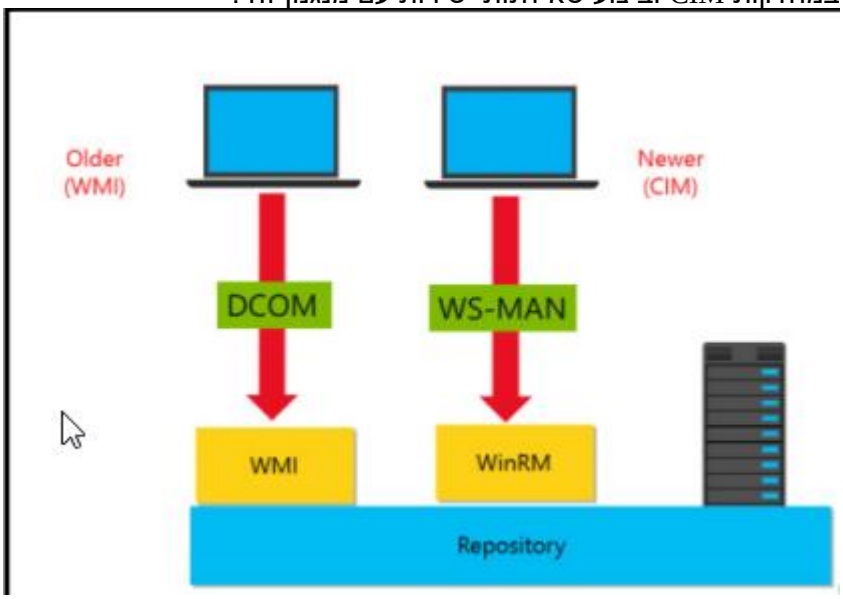
יתרון נוסף שיש ל-PS הוא השימוש ב-CIM ו-WMI .

הסבר קצר על WMI ו-CIM

החל מווינדוס NT מייקרוסופט החלה ליישם מנגנון שנקרא WMI Windows Management Instrumentation , המטרה של מנגנון זה הינה להקל על מנהלי רשת לבצע שאילתות וניהול של מאפייני החומרה של מחשב מבוסס ווינדוס ,כתחליף לפרוטוקול SNMP. WMI מאפשר ייצוג מבוסס אובייקטים של מערכת המחשב ומאפשר ביצוע שאילתות בכל שפת תכנות מודרנית כולל PS , מעבר לכך WMI מאפשר ביצוע שאילתה ממחשב מרוחק .

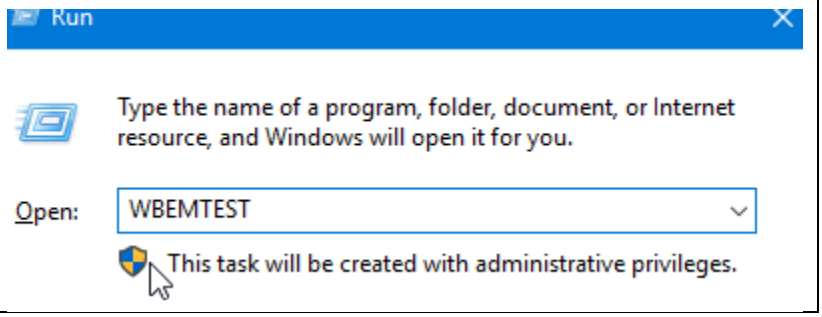
WMI הינו היישום של מיקרוסופט ל-(Web-Based Enterprise Management (WBEM , שזוהי יוזמה לפיתוח תקן ניהולי אחיד לגישה בסביבה ארגונית. WMI משתמש במנגנון של Common Information Model (CIM) שהינו תקן בתעשייה ופותח על ידי ארגון DMTF שמייקרוסופט חברה בו. (https://www.dmtf.org/)

במילים פשוטות WMI הינו יישום של תקן ניהולי פתוח שעובד עם מע' הפעלה ווינדוס בלבד , היות שכך המנגנון לא תועד ונכתב כמו שצריך והיה קושי ליישם אותו . מייקרוסופט בגירסאות המתקדמות של PS (3 צפונה) מעודדת את השימוש במחלקות CIM וביצוע שאילתות ישירות עם מנגנון זה .

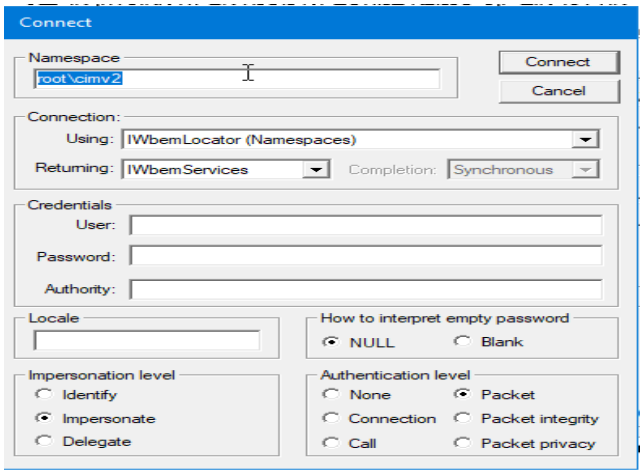


בתמונה נוכל לראות שהן WMI והן CIM ניגשים לאותו מאגר מידע ההבדל נעוץ בדרך שבה פונים למחשב מרוחק , DCOM מיושן ולא מאובטח בהשוואה ל WS-MAN

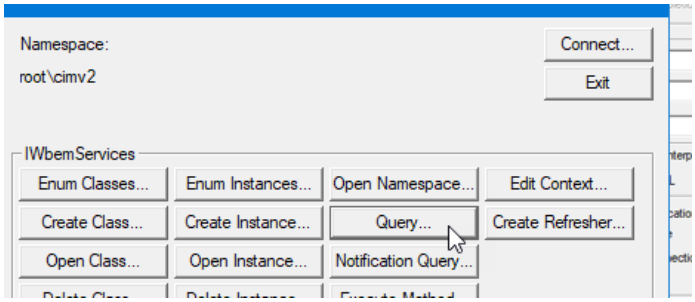
כדי לבצע שאילתה ראשונית נוכל להריץ בווינדוס שלנו את התוכנה WBEMTEST שתאפשר לנו להתחבר למחלקות ולבצע שאילתה



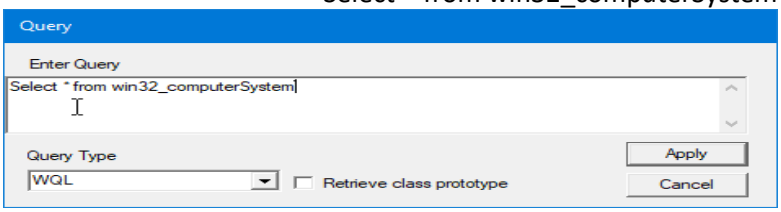
נתחבר :



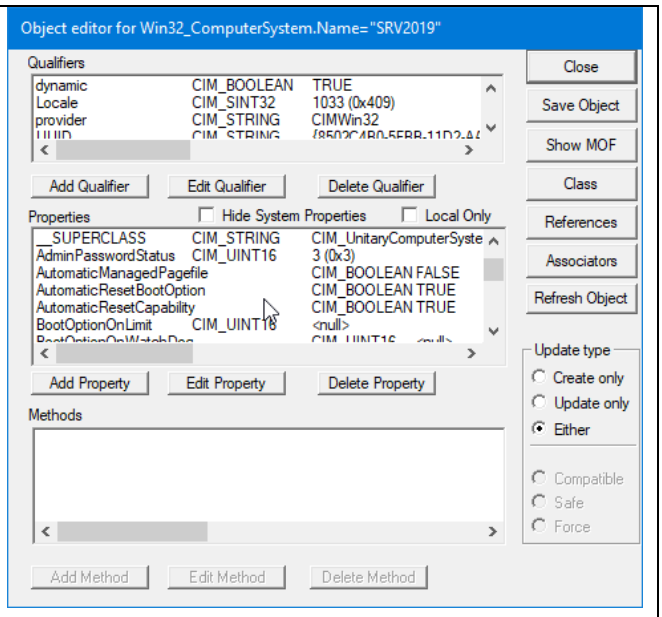
במסך הבא נזין שאילתה :



Select * from win32_computerSystem



ונקבל בתשובה את שם המחשב שלנו
לחיצה נוספת על התשובה תראה לנו את כל המאפיינים של המחלקה הזו וכל הפעולות אותן ניתן לבצע



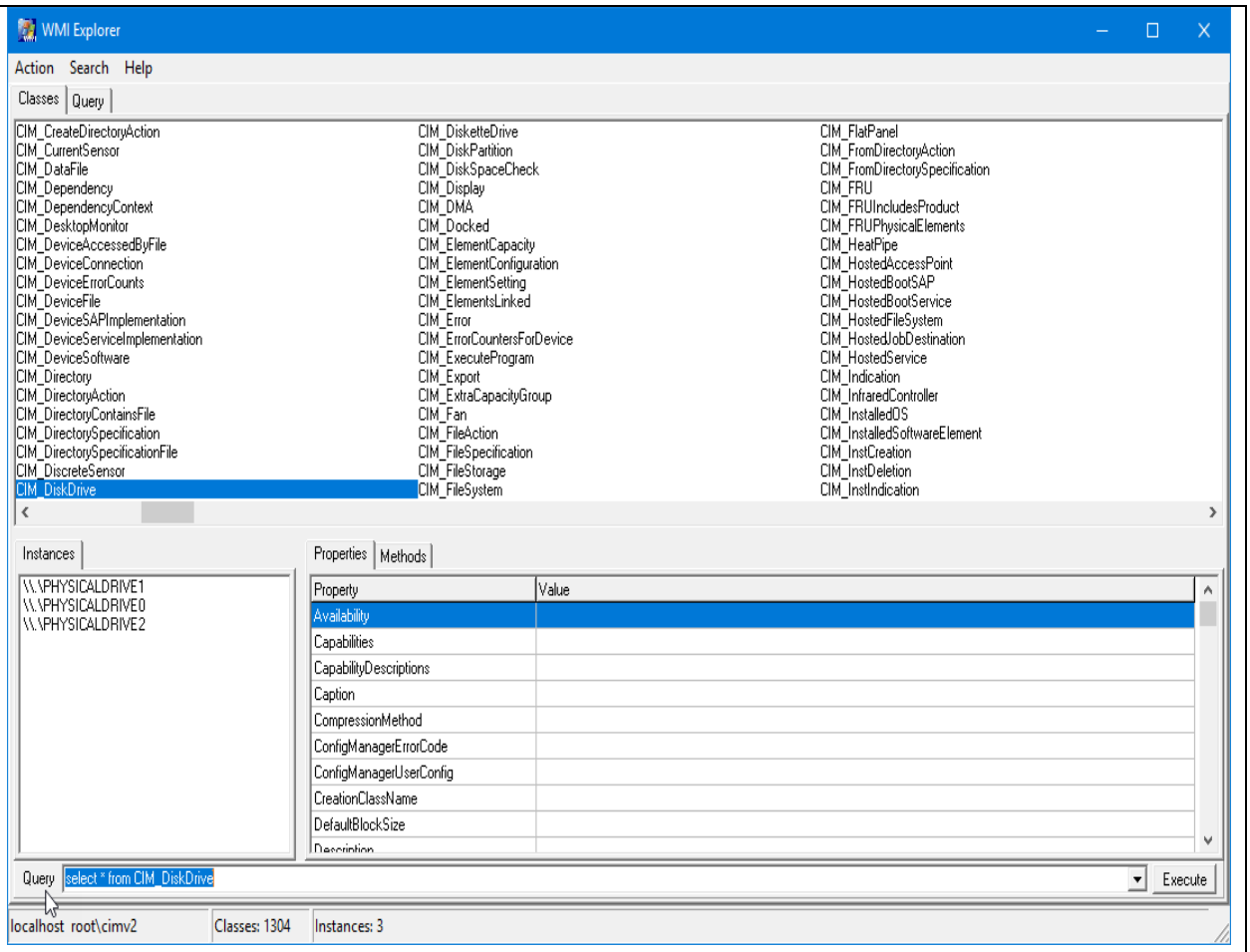
הצלחנו להבחין שניתן לבצע שאילתות WMI באמצעות שפה שנקראת WQL
https://docs.microsoft.com/he-il/windows/win32/wmisdk/wql-sql-for-wmi?redirectedfrom=MSDN

ישנו גם מנגנון מבוסס שורת פקודה wmic המאפשר לכתוב שאילתות דרך ממשק CMD

wmic bios get smbiosbiosversion הפקודה הזו תתן לנו בפלט שלה את גירסת הביוס של המחשב

```
PS C:\Users\Administrator> wmic bios get smbiosbiosversion
SMBIOSBIOSVersion
K01 v02.05
```

על מנת לבצע סקירה של כל המחלקות והמאפיינים שלהם תוכלו להוריד WMI EXPLORER (חיפוש פשוט בגוגל ימצא לכם המון כלים חינמיים) אחד פשוט נמצא בקישור הבא :
http://www.hostmonitor.biz/download/wmiexplorer.zip



בדוגמה בחרתי במחלקה של כונן דיסקים , ואני יכול לראות את האינסטנסים (כל הכוננים) את המאפיינים ואת הפעולות שאני יכול לבצע , וכמו כן את השאילתה הרלוונטית בשפת WQL

באמצעות הפקודה הבאה נוכל לראות את כל המחלקות שאליהן אוכל לגשת על מנת לקבל מידע :
Get-CimClass –Namespace root\CIMv2

נבצע שאילתה פשוטה על כונן הדיסקים :
Get-CimInstance –ClassName Win32_LogicalDisk
לאחר מכן נבקש רק כוננים קשיחים :
Get-CimInstance –ClassName Win32_LogicalDisk –Filter "DriveType=3”
אותה שאילתה באמצעות WQL :
Get-CimInstance –Query "SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3”

יכולת מעניינת היא ליצור CIM SESSION כלומר ליצור חיבור למחשב אחד או יותר המרוחקים :
$s2 = New-CimSession –ComputerName LON-CL1,LON-DC1
Get-CimInstance –CimSession $s2 –ClassName Win32_OperatingSystem

בדוגמא הזו אני אוכל לבדוק את סוג מע' ההפעלה לשני המחשבים הנמצאים במשתנה $s2

היות וכל דבר ב-PS הוא אובייקט בואו ננסה לכתוב את הפקודה הבאה :

```
S C:\Users\Administrator> Get-CimInstance -ClassName Win32_LogicalDisk |GM
```

הפקודה תציג לנו את כל המאפיינים של המחלקה כולל הפעולות אותן ניתן לבצע על הדיסק
באמצעות הפקודה Invoke-CimMethod נוכל לבצע פעולות שונות :

```
Get-CimInstance -ClassName Win32_Process -Filter "Name='notepad.exe'" –Computername LON-DC1 |
Invoke-CimMethod -MethodName Terminate
```

הפקודה הבאה תוודא ש NOTEPAD אכן רץ במחשב המרוחק ותסגור אותו

לסיכום : WMI ו-CIM הינם כלי ניהול עוצמתיים , הם מאפשרים לנו גישה למידע על החומרה ורכיבי מע' ההפעלה של המחשב המקומי או המרוחק ואף יותר ממחשב אחד במקביל. ניתן לבצע שאילתות בשפת WQL וניתן לבצע שאילתות ופעולות ישירות דרך פקודות PS .

חשוב להבין שהטכנולוגיה הינה OPEN SOURCE וישנן מע' הפעלה לינוקסיות המכילות פניה למחלקות הללו באמצעות רכיב שנקרא OMI היודע לתווך שאילתות CIM למע' .
השילוב של היכולת הזו והיכולת של PS להריץ פעולות ברקע וכמו כן העובדה שכיום היא OPEN SOURCE מביאה אותנו ליכולת בשם POWERSHELL DSC והיא היכולת לבצע אוטומציה מלאה למע' ווינדוס ומע' חיצוניות באמצעות שפה ייחודית . על כך אכתוב מאמר אחר לגמרי.

המלצה שלי לשחק עם WMI ולקרוא על המחלקות השונות בתיעוד של מייקרוסופט , כמו כן ברשת תוכלו למצוא המון סקריפטים ב-PS העושים שימוש בCIM או WMI על מנת ליצור סקירה של כלל החומרה בארגון

הסבר מעניין בקישור הבא :
https://4sysops.com/archives/how-to-build-a-powershell-inventory-script-for-windows-servers/

קישורים לקריאה :

https://devblogs.microsoft.com/scripting/comparing-cim-and-wmi-in-powershell/

https://devblogs.microsoft.com/scripting/should-i-use-cim-or-wmi-with-windows-powershell/

סיכום קצר של חלק א:

גילינו ש-PS הינה שפת סקריפטים מודרנית המבוססת על .NET המאפשרת לבצע כל פעולה ואינה מוגבלת לפקודות ספיציפיות

למדנו בצורה כללית מהו תכנות מונחה עצמים והיתרונות שלו בשימוש בסקריפטים (גישה מהירה לכלל האובייקטים במע' ההפעלה וביצוע פעולות)

הכרנו סט חדש של אובייקטים לניהול בשם CIM ראינו שיש לנו גישה ישירה מתוך PS וכמו כן שליטה על מע' שאינן מייקרוסופטיות.

בחלק השני נסקור את : מבנה השפה וגישה לעזרה, התקנה של מודולים נוספים , עריכת הממשק ופרופיל המשתמש, שימוש בעורך מתקדם VS CODE.

חלק ב :

מבנה השפה

התקנת מודולים

ממשק ופרופילים

קבלת עזרה

עריכה ISE VS CODE

חלק ג:

הדגמה של ניהול המחשב באמצעות CIM

הדגמה של ניהול אובייקטים באמצעות AD

מבוא לאוטומציה באמצעות POWERSHELL

חלק ב- עבודה עם פקודות