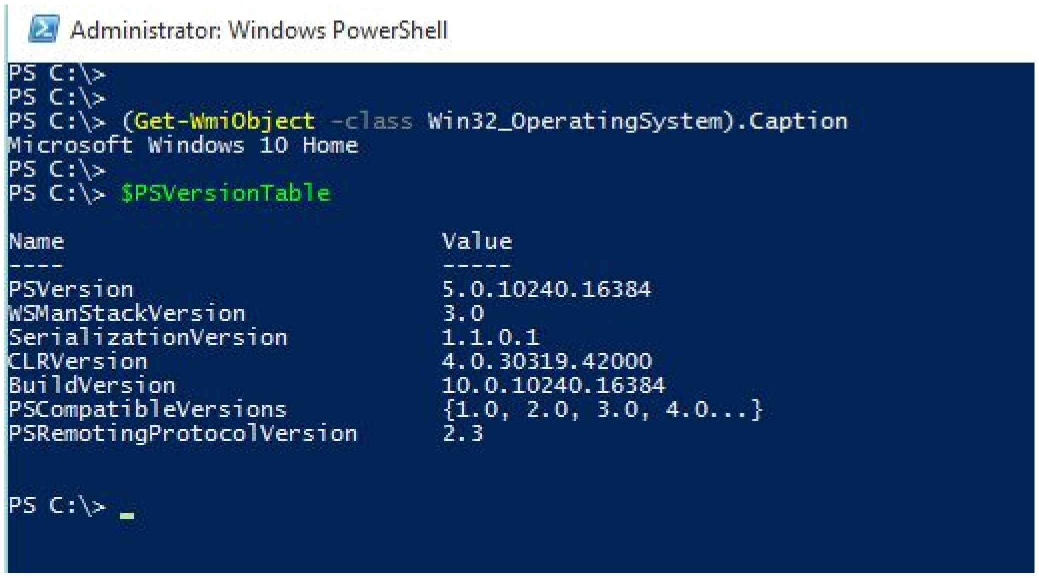
Windows Storage
Powershell and DISKPART

This presentation will mostly focus on combining Powershell & DISKPART, either in scripting, or simply by using commands.

The testing has been performed on a Windows 10 Home OS, Powershell version 5.



Administrator: Windows PowerShell

```
PS C:\>
PS C:\>
PS C:\> (Get-WmiObject -class Win32_OperatingSystem).Caption
Microsoft Windows 10 Home
PS C:\>
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.0.10240.16384
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.42000
BuildVersion                   10.0.10240.16384
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3


PS C:\> _
```

I have tried to stay away from importing any kind of modules that might have made the presentation easier, since the main purpose is to manage storage with the resources that can already be found on the computer.

For now,I will focus on simple commands, that will not involve any alteration of storage (such as format or deletion of partitions).

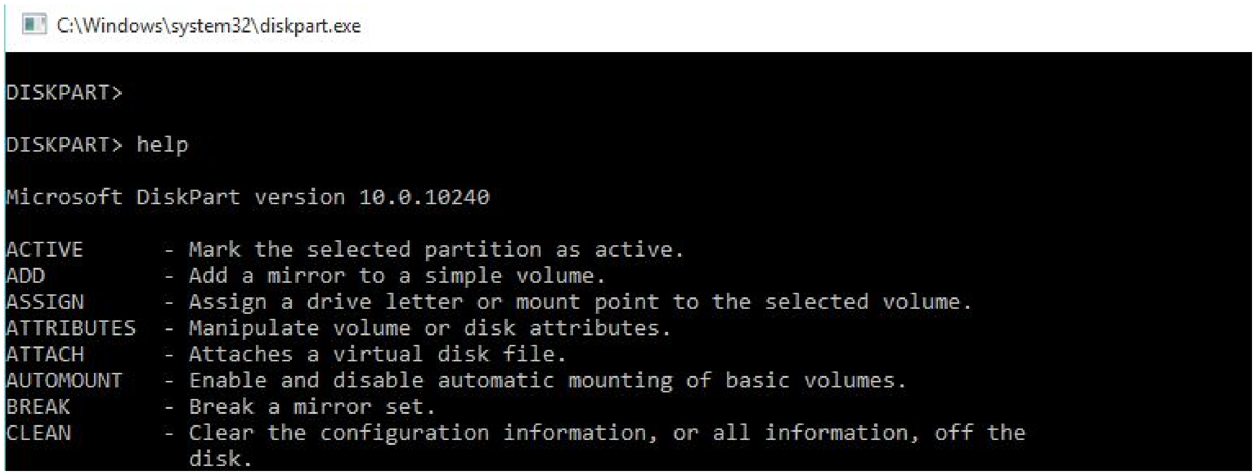
Brief introduction on DISKPART

As you well know, DISKPART allows you to manage disk, partitions and volumes, either through commands or scripting, since Windows 2000 appearance.. You might want to consider it as the follower of **fdisk** utility.

For starting the utility, type **diskpart** either in command prompt or in powershell environment.

Use **help**, to display the list of commands:

(few commands example)



```
C:\Windows\system32\diskpart.exe

DISKPART>

DISKPART> help

Microsoft DiskPart version 10.0.10240

ACTIVE      - Mark the selected partition as active.
ADD         - Add a mirror to a simple volume.
ASSIGN      - Assign a drive letter or mount point to the selected volume.
ATTRIBUTES  - Manipulate volume or disk attributes.
ATTACH      - Attaches a virtual disk file.
AUTOMOUNT   - Enable and disable automatic mounting of basic volumes.
BREAK       - Break a mirror set.
CLEAN       - Clear the configuration information, or all information, off the
              disk.
```

Brief introduction on Powershell

Powershell is a command line shell and also supports scripting, and it can be combined with different programming/scripting languages (such as C#, Perl, Python) or even relational databases.

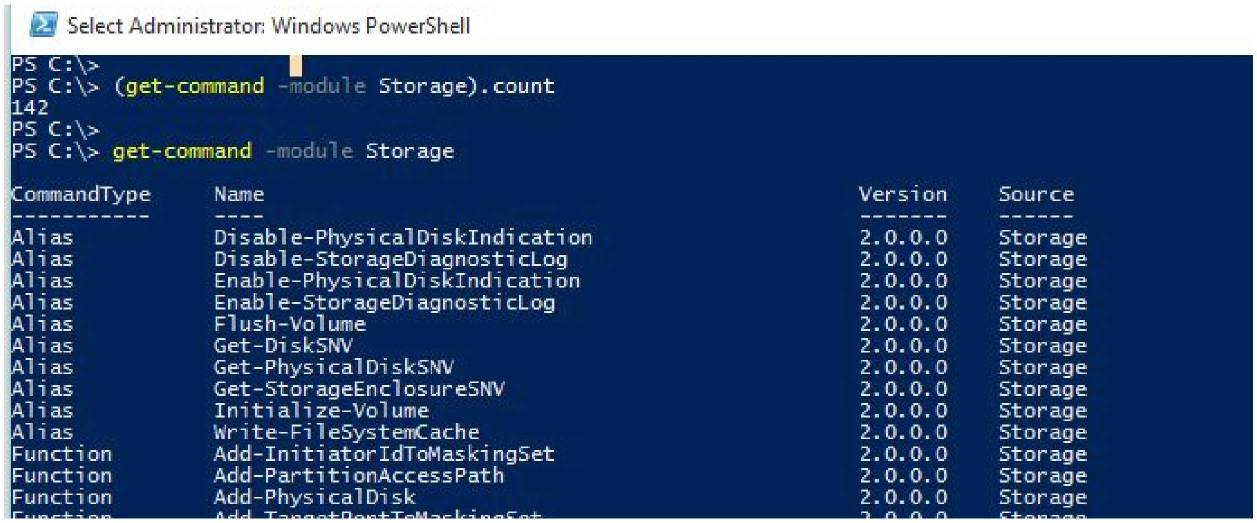
It also supports commands from cmd (with few exceptions: like **path**, for instance) , and can behave like a Unix shell - you can run known commands such **ls**, **more**, **pwd**.

*If it's easier on you for the Unix shell comparison, Powershell behaves a bit like **Cygwin**.*

OK, now that we know Powershell can coexist with pretty much everything out-there, time to see how it can be useful on storage level, and how to call DISKPART commands from it, and of course, crafting a small application prototype (source code included)
Yes, Powershell does support **graphic user interface(GUI)**, as well.

Powershell and Storage Cmdlets

As per below output, there are 142 commands to be used on the Storage side.

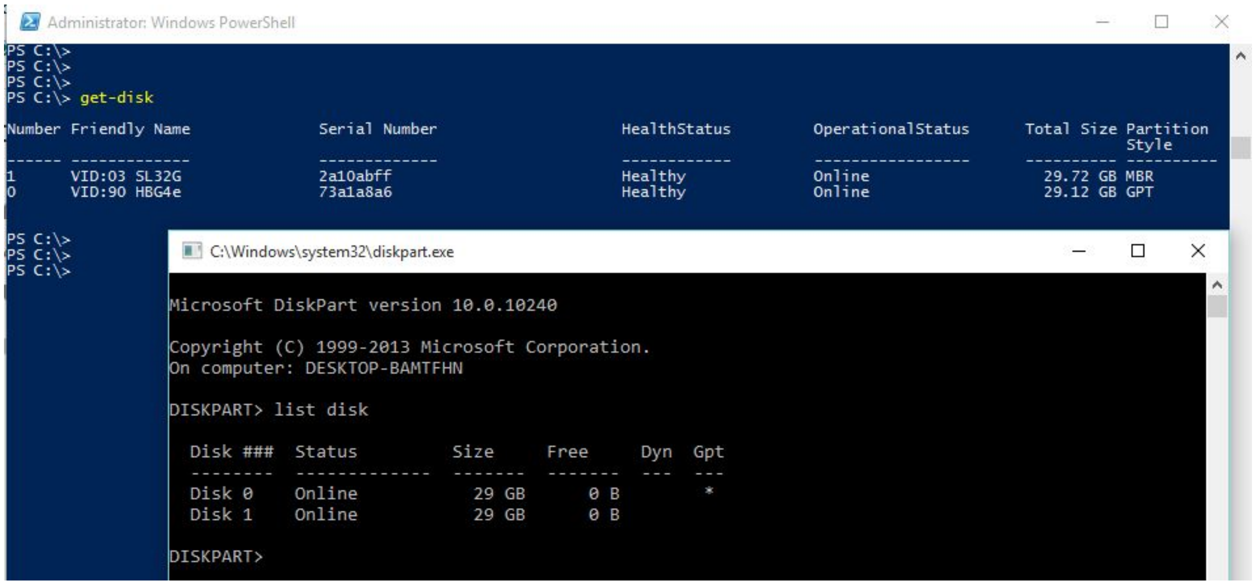


Select Administrator: Windows PowerShell

```
PS C:\>
PS C:\> (get-command -module Storage).count
142
PS C:\>
PS C:\> get-command -module Storage

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Disable-PhysicalDiskIndication                     2.0.0.0    Storage
Alias           Disable-StorageDiagnosticLog                       2.0.0.0    Storage
Alias           Enable-PhysicalDiskIndication                      2.0.0.0    Storage
Alias           Enable-StorageDiagnosticLog                        2.0.0.0    Storage
Alias           Flush-Volume                                       2.0.0.0    Storage
Alias           Get-DiskSNV                                        2.0.0.0    Storage
Alias           Get-PhysicalDiskSNV                                2.0.0.0    Storage
Alias           Get-StorageEnclosureSNV                            2.0.0.0    Storage
Alias           Initialize-Volume                                  2.0.0.0    Storage
Alias           Write-FileSystemCache                              2.0.0.0    Storage
Function        Add-InitiatorIdToMaskingSet                        2.0.0.0    Storage
Function        Add-PartitionAccessPath                            2.0.0.0    Storage
Function        Add-PhysicalDisk                                   2.0.0.0    Storage
Function        Add-TargetPortToMaskingSet                         2.0.0.0    Storage
```

Let's start making a comparison between Powershell and DISKPART now

Check disks visible to the operating system



Administrator: Windows PowerShell

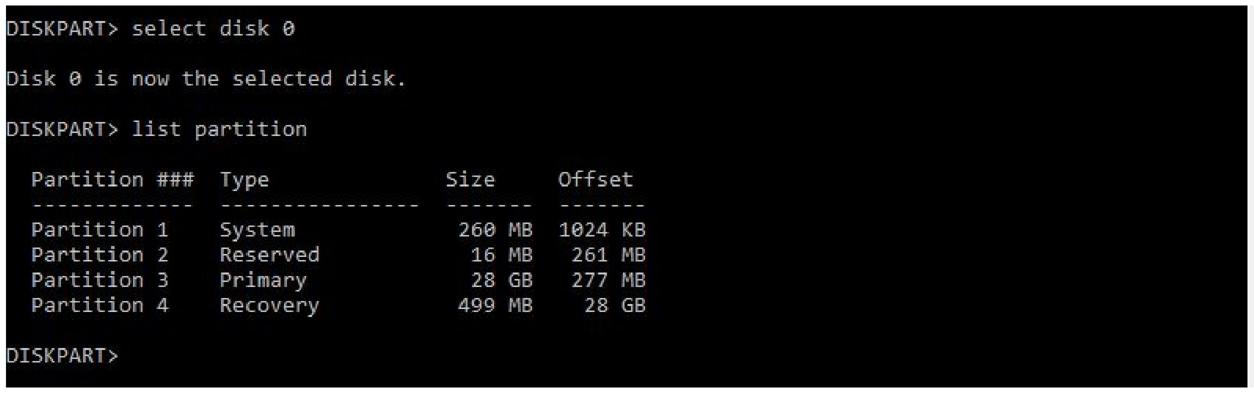
```
PS C:\>
PS C:\>
PS C:\>
PS C:\> get-disk

Number Friendly Name            Serial Number                    HealthStatus         OperationalStatus      Total Size Partition
                                                                                                                        Style
------ -------------            -------------                    ------------         -----------------      ---------- ---------
1      VID:03 SL32G             2a10abff                         Healthy              Online                   29.72 GB MBR
0      VID:90 HBG4e             73a1a8a6                         Healthy              Online                   29.12 GB GPT

PS C:\>
PS C:\>
PS C:\>
```

C:\Windows\system32\diskpart.exe

```
Microsoft DiskPart version 10.0.10240

Copyright (C) 1999-2013 Microsoft Corporation.
On computer: DESKTOP-BAMTFHN

DISKPART> list disk

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
  Disk 0    Online           29 GB      0 B        *
  Disk 1    Online           29 GB      0 B

DISKPART>
```

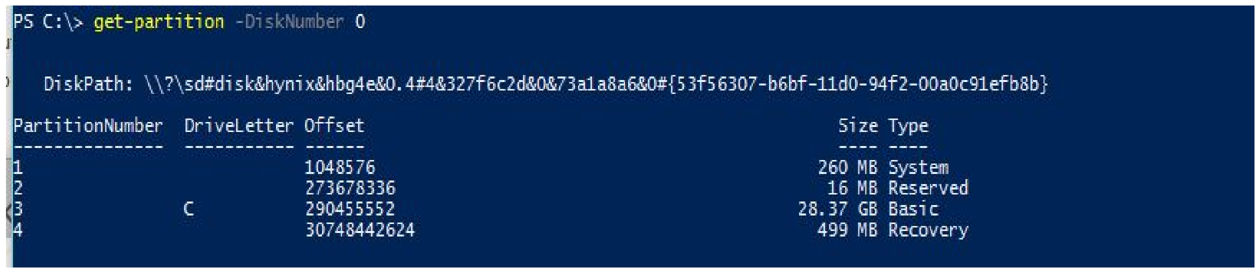
2) Partition selection

2.1) Select partitions on DISKPART



```
DISKPART> select disk 0

Disk 0 is now the selected disk.

DISKPART> list partition

  Partition ###  Type              Size     Offset
  -------------  ----------------  -------  -------
  Partition 1    System             260 MB  1024 KB
  Partition 2    Reserved            16 MB   261 MB
  Partition 3    Primary             28 GB   277 MB
  Partition 4    Recovery           499 MB    28 GB

DISKPART>
```

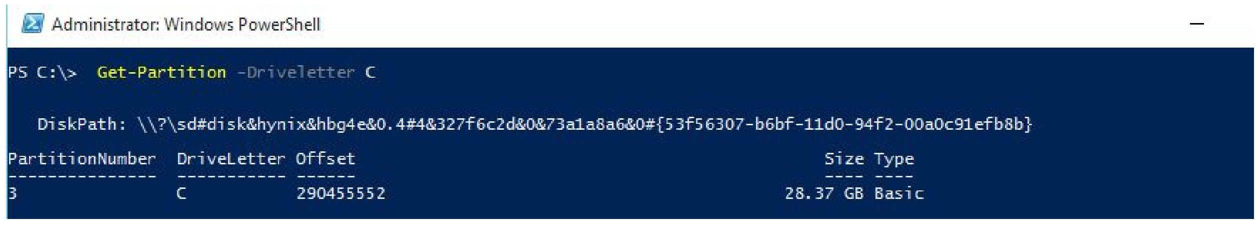
2.2) Select partitions on Powershell



```
PS C:\> get-partition -DiskNumber 0

   DiskPath: \\?\sd#disk&hynix&hbg4e&0.4#4&327f6c2d&0&73a1a8a6&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset                                        Size Type
---------------  ----------- ------                                        ---- ----
1                            1048576                                     260 MB System
2                            273678336                                    16 MB Reserved
3                C           290455552                                 28.37 GB Basic
4                            30748442624                                 499 MB Recovery
```

Let’s see what else we can find over-here.

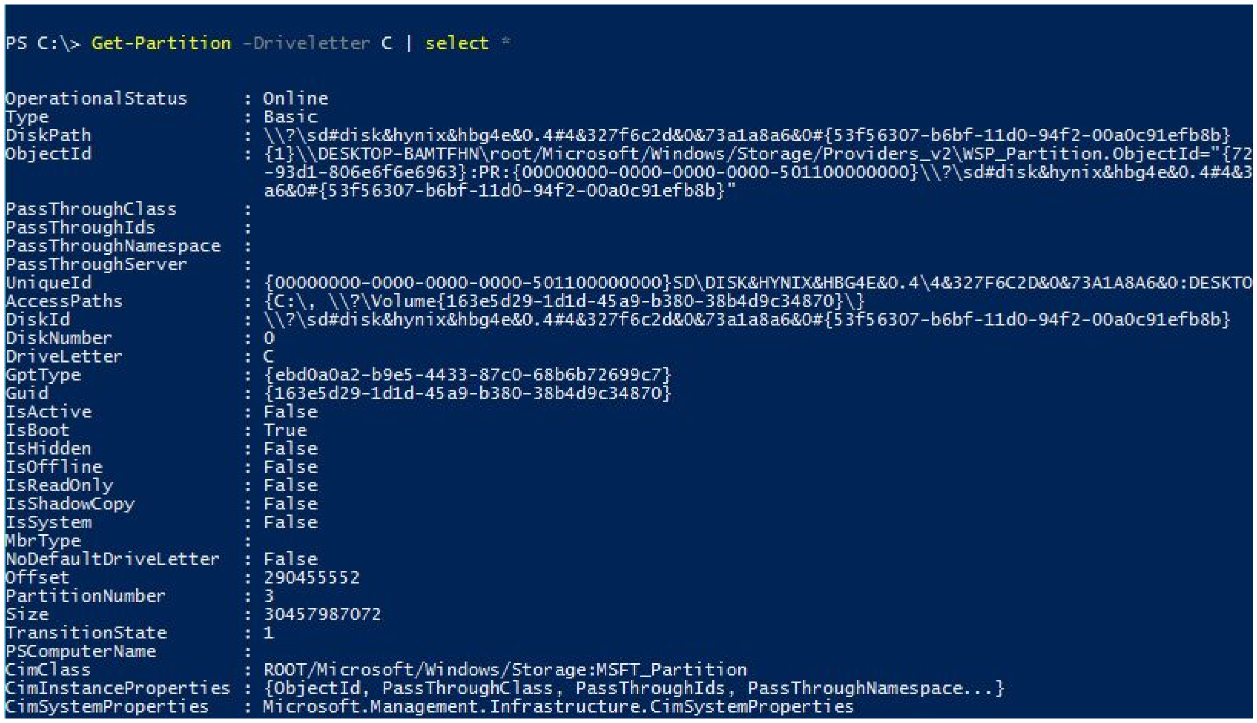
2.2.a) Let’s check partitions associated with drive C:



```
PS C:\>  Get-Partition -Driveletter C

   DiskPath: \\?\sd#disk&hynix&hbg4e&0.4#4&327f6c2d&0&73a1a8a6&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

PartitionNumber  DriveLetter Offset                                        Size Type
---------------  ----------- ------                                        ---- ----
3                C           290455552                                 28.37 GB Basic
```

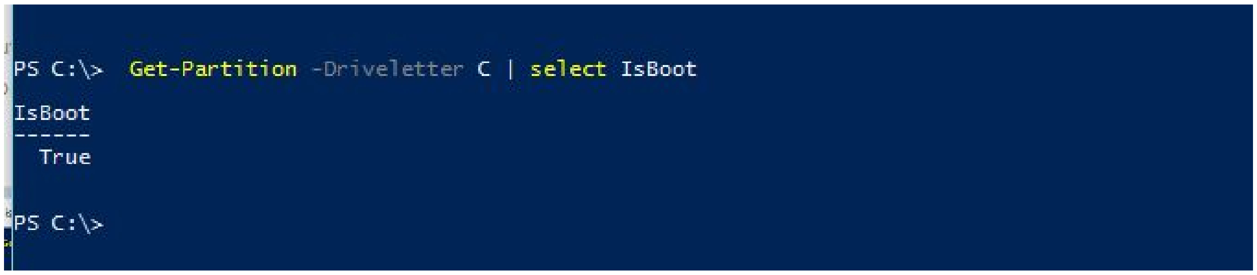
2.2.b) Powershell commands can be used in similar ways as SQL (as I have mentioned in the introductive part). In case you need to list a certain field, just use **select** option:

Select all (**select ***) for partition at drive C. You can also notice the usage of pipes (yey! More Unix similarities)



```
PS C:\> Get-Partition -Driveletter C | select *


OperationalStatus     : Online
Type                  : Basic
DiskPath              : \\?\sd#disk&hynix&hbg4e&0.4#4&327f6c2d&0&73a1a8a6&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
ObjectId              : {1}\\DESKTOP-BAMTFHN\root/Microsoft/Windows/Storage/Providers_v2\WSP_Partition.ObjectId="{72
                        -93d1-806e6f6e6963}:PR:{00000000-0000-0000-0000-501100000000}\\?\sd#disk&hynix&hbg4e&0.4#4&3
                        a6&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}"
PassThroughClass      :
PassThroughIds        :
PassThroughNamespace  :
PassThroughServer     :
UniqueId              : {00000000-0000-0000-0000-501100000000}SD\DISK&HYNIX&HBG4E&0.4\4&327F6C2D&0&73A1A8A6&0:DESKTO
AccessPaths           : {C:\, \\?\Volume{163e5d29-1d1d-45a9-b380-38b4d9c34870}\}
DiskId                : \\?\sd#disk&hynix&hbg4e&0.4#4&327f6c2d&0&73a1a8a6&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
DiskNumber            : 0
DriveLetter           : C
GptType               : {ebd0a0a2-b9e5-4433-87c0-68b6b72699c7}
Guid                  : {163e5d29-1d1d-45a9-b380-38b4d9c34870}
IsActive              : False
IsBoot                : True
IsHidden              : False
IsOffline             : False
IsReadOnly            : False
IsShadowCopy          : False
IsSystem              : False
MbrType               :
NoDefaultDriveLetter  : False
Offset                : 290455552
PartitionNumber       : 3
Size                  : 30457987072
TransitionState       : 1
PSComputerName        :
CimClass              : ROOT/Microsoft/Windows/Storage:MSFT_Partition
CimInstanceProperties : {ObjectId, PassThroughClass, PassThroughIds, PassThroughNamespace...}
CimSystemProperties   : Microsoft.Management.Infrastructure.CimSystemProperties
```

Let's take a closer look at those exposed fields, and reduce our select to a single field to be listed as output:

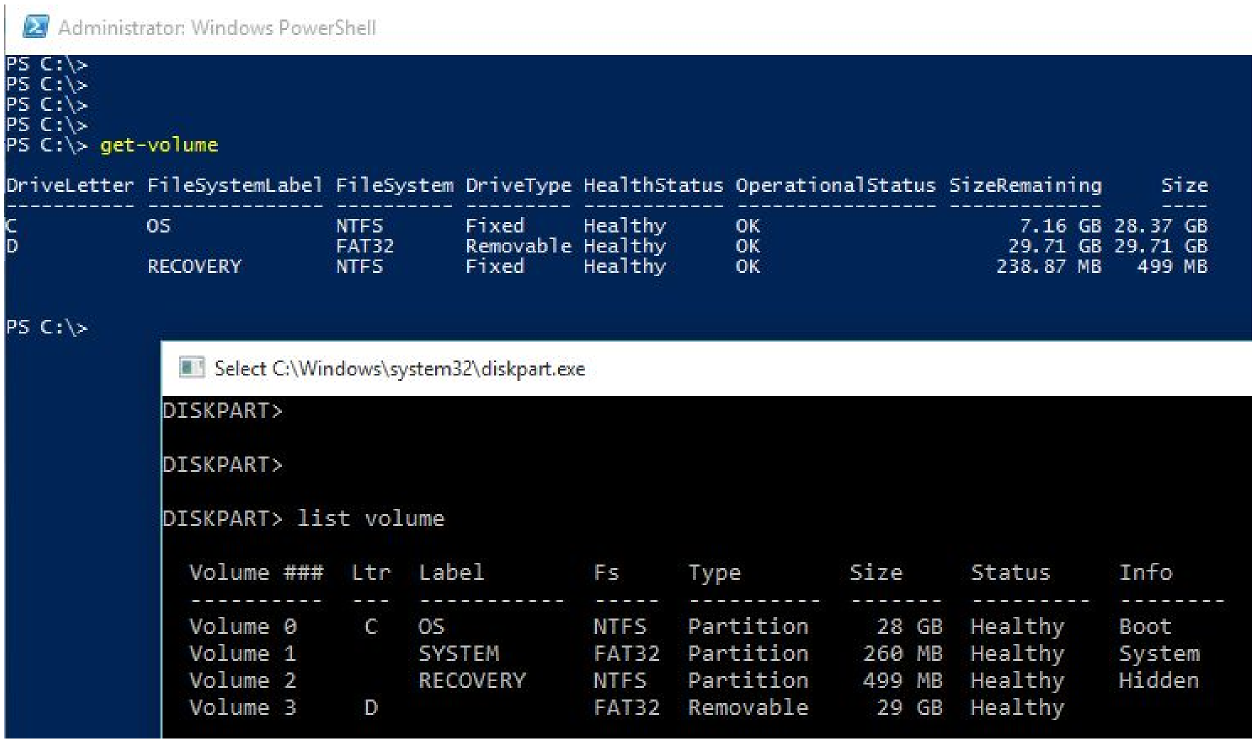


```
PS C:\>  Get-Partition -Driveletter C | select IsBoot

IsBoot
------
  True


PS C:\>
```

...and we have found our bootable partition!

Volumes

As you can notice, Powershell does not expose the hidden Volume 2, when listing all volumes.



Administrator: Windows PowerShell

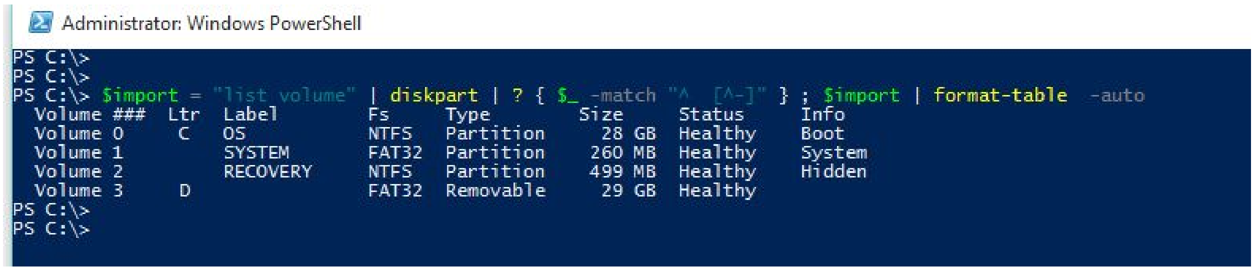
```
PS C:\>
PS C:\>
PS C:\>
PS C:\>
PS C:\> get-volume

DriveLetter FileSystemLabel FileSystem DriveType HealthStatus OperationalStatus SizeRemaining     Size
----------- --------------- ---------- --------- ------------ ----------------- -------------     ----
C           OS              NTFS       Fixed     Healthy      OK                      7.16 GB 28.37 GB
D                           FAT32      Removable Healthy      OK                     29.71 GB 29.71 GB
            RECOVERY        NTFS       Fixed     Healthy      OK                    238.87 MB   499 MB

PS C:\>
```

Select C:\Windows\system32\diskpart.exe

```
DISKPART>

DISKPART>

DISKPART> list volume

  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  ----------  ---  -----------  -----  ----------  -------  ---------  --------
  Volume 0     C   OS           NTFS   Partition     28 GB  Healthy    Boot
  Volume 1         SYSTEM       FAT32  Partition    260 MB  Healthy    System
  Volume 2         RECOVERY     NTFS   Partition    499 MB  Healthy    Hidden
  Volume 3     D                FAT32  Removable     29 GB  Healthy
```

> Now, it's time a good time to show Powershell importing information from DISKPART

So, in case we want all volumes to be listed, we could try to use a very small script:

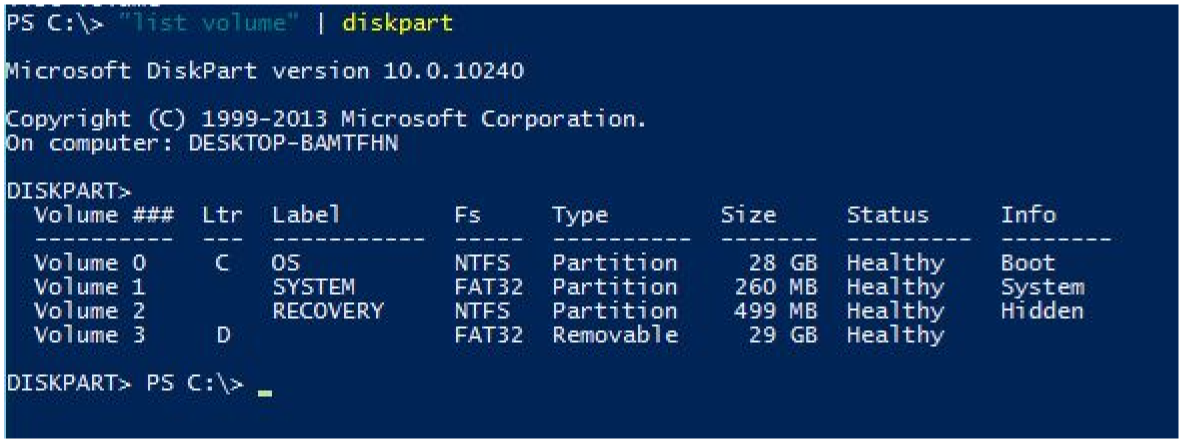


Administrator: Windows PowerShell

```
PS C:\>
PS C:\>
PS C:\> $import = "list volume" | diskpart | ? { $_ -match "^  [^-]" } ; $import | format-table  -auto
  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  Volume 0     C   OS           NTFS   Partition     28 GB  Healthy    Boot
  Volume 1         SYSTEM       FAT32  Partition    260 MB  Healthy    System
  Volume 2         RECOVERY     NTFS   Partition    499 MB  Healthy    Hidden
  Volume 3     D                FAT32  Removable     29 GB  Healthy
PS C:\>
PS C:\>
```

Simply put, we have called diskpart from powershell and, we indicated we need "list volume" information, only (without any extra text)

If we had not use the **-match** option or **format-table**, our output might have looked as below, if reduced to simply invoking “list volume” for diskpart

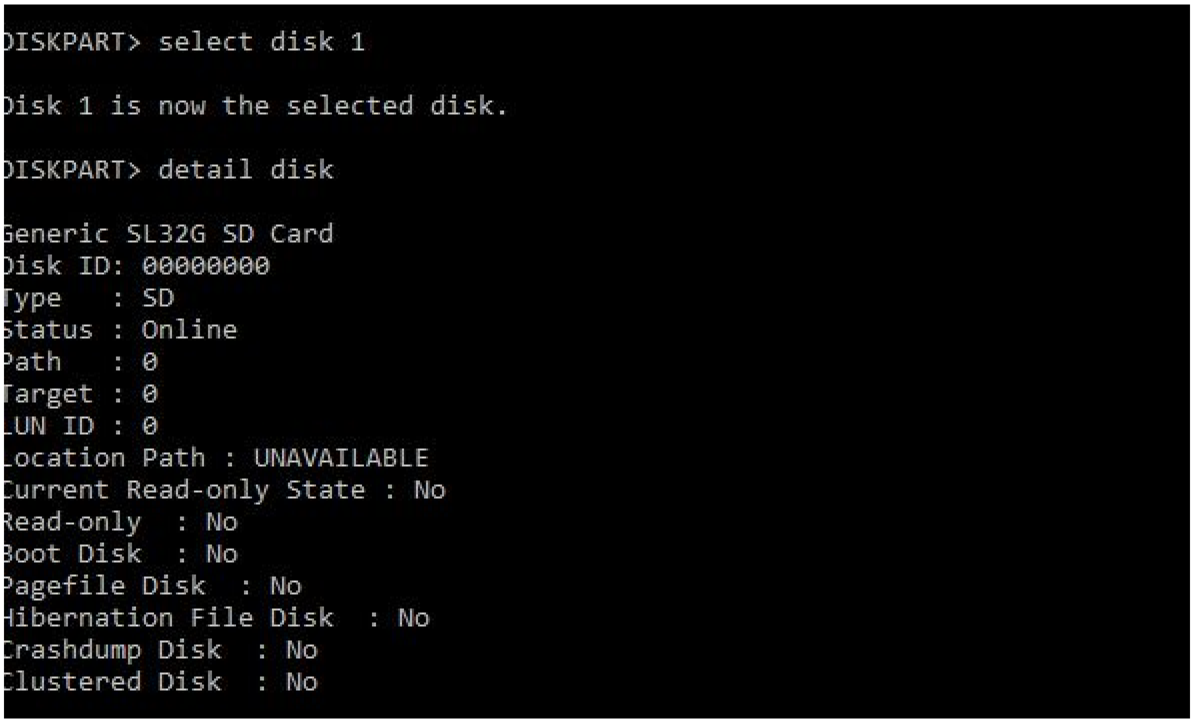


```
PS C:\> "list volume" | diskpart

Microsoft DiskPart version 10.0.10240

Copyright (C) 1999-2013 Microsoft Corporation.
On computer: DESKTOP-BAMTFHN

DISKPART>
  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  ----------  ---  -----------  -----  ----------  -------  ---------  --------
  Volume 0     C   OS           NTFS   Partition     28 GB  Healthy    Boot
  Volume 1         SYSTEM       FAT32  Partition    260 MB  Healthy    System
  Volume 2         RECOVERY     NTFS   Partition    499 MB  Healthy    Hidden
  Volume 3     D                FAT32  Removable     29 GB  Healthy

DISKPART> PS C:\> _
```

A bit messy, huh? Still useful. :)

And since we are at the scripting part, let’s try something. - listing disks with a script

First, in diskpart you can check disk details as below:



```
DISKPART> select disk 1

Disk 1 is now the selected disk.

DISKPART> detail disk

Generic SL32G SD Card
Disk ID: 00000000
Type   : SD
Status : Online
Path   : 0
Target : 0
LUN ID : 0
Location Path : UNAVAILABLE
Current Read-only State : No
Read-only  : No
Boot Disk  : No
Pagefile Disk  : No
Hibernation File Disk  : No
Crashdump Disk  : No
Clustered Disk  : No
```

So, we already have what we need, and we have seen how we could import from diskpart.
Let’s try now to list the disks, by automating DISKPART operations.

So, in our case, we should be using something like the below syntax:

```
Diskpart  /s getCommandFromHere.txt
```

And in that getCommandFromHere.txt, diskpart should be looking for commands like “list disks”, for instance.

Right until now, our script should look like this:

```
  new-item -Name DisksListed.txt -Itemtype file -force | out-null
  add-content -path DisksListed.txt "list disk"
  $DisksListed=(diskpart /s DisksListed.txt)
```

Let’s think about the output. When invoking diskpart, you have noticed in previous example, that it also provides extra text, that is a bit undesirable:

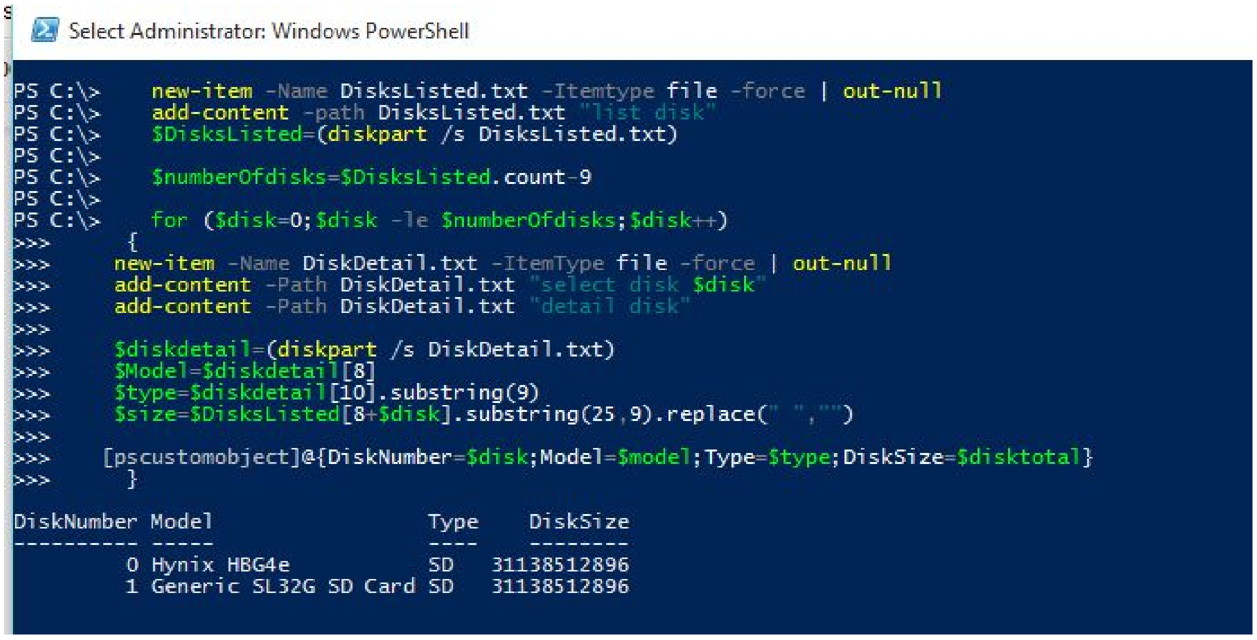
```
 $numberOfdisks=$DisksListed.count-9
```

Next, we need to list the disks, select them and access their details(model, type and size):

```
 for ($disk=0;$disk -le $numberOfdisks;$disk++)
          {
          new-item -Name DiskDetail.txt -ItemType file -force | out-null
          add-content -Path DiskDetail.txt "select disk $disk"
          add-content -Path DiskDetail.txt "detail disk"

          $diskdetail=(diskpart /s DiskDetail.txt)
          $Model=$diskdetail[8]
          $type=$diskdetail[10].substring(9)
          $size=$DisksListed[8+$disk].substring(25,9).replace(" ","")

  [pscustomobject]@{DiskNumber=$disk;Model=$model;Type=$type;DiskSize=$disktotal}
          }
```

...and that’s it!

Let’s run it directly, and see what can get:

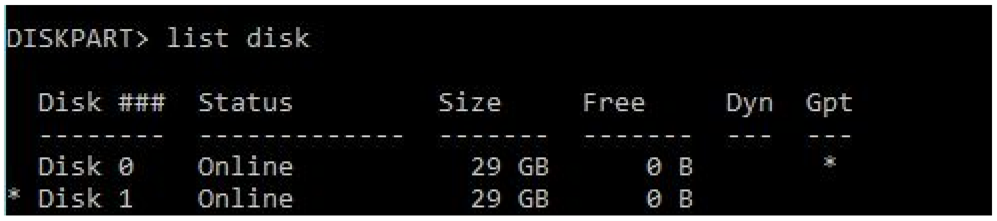


Select Administrator: Windows PowerShell

```
PS C:\>    new-item -Name DisksListed.txt -Itemtype file -force | out-null
PS C:\>    add-content -path DisksListed.txt "list disk"
PS C:\>    $DisksListed=(diskpart /s DisksListed.txt)
PS C:\>
PS C:\>    $numberOfdisks=$DisksListed.count-9
PS C:\>
PS C:\>    for ($disk=0;$disk -le $numberOfdisks;$disk++)
>>>       {
>>>      new-item -Name DiskDetail.txt -ItemType file -force | out-null
>>>      add-content -Path DiskDetail.txt "select disk $disk"
>>>      add-content -Path DiskDetail.txt "detail disk"
>>>
>>>      $diskdetail=(diskpart /s DiskDetail.txt)
>>>      $Model=$diskdetail[8]
>>>      $type=$diskdetail[10].substring(9)
>>>      $size=$DisksListed[8+$disk].substring(25,9).replace(" ","")
>>>
>>>     [pscustomobject]@{DiskNumber=$disk;Model=$model;Type=$type;DiskSize=$disktotal}
>>>       }

DiskNumber Model                 Type    DiskSize
---------- -----                 ----    --------
         0 Hynix HBG4e           SD   31138512896
         1 Generic SL32G SD Card SD   31138512896
```

So, our disk size is 31138512896 bytes, that makes about **29gigs.**

… and on diskpart, the provided info is indeed 29gigs. We`re good!

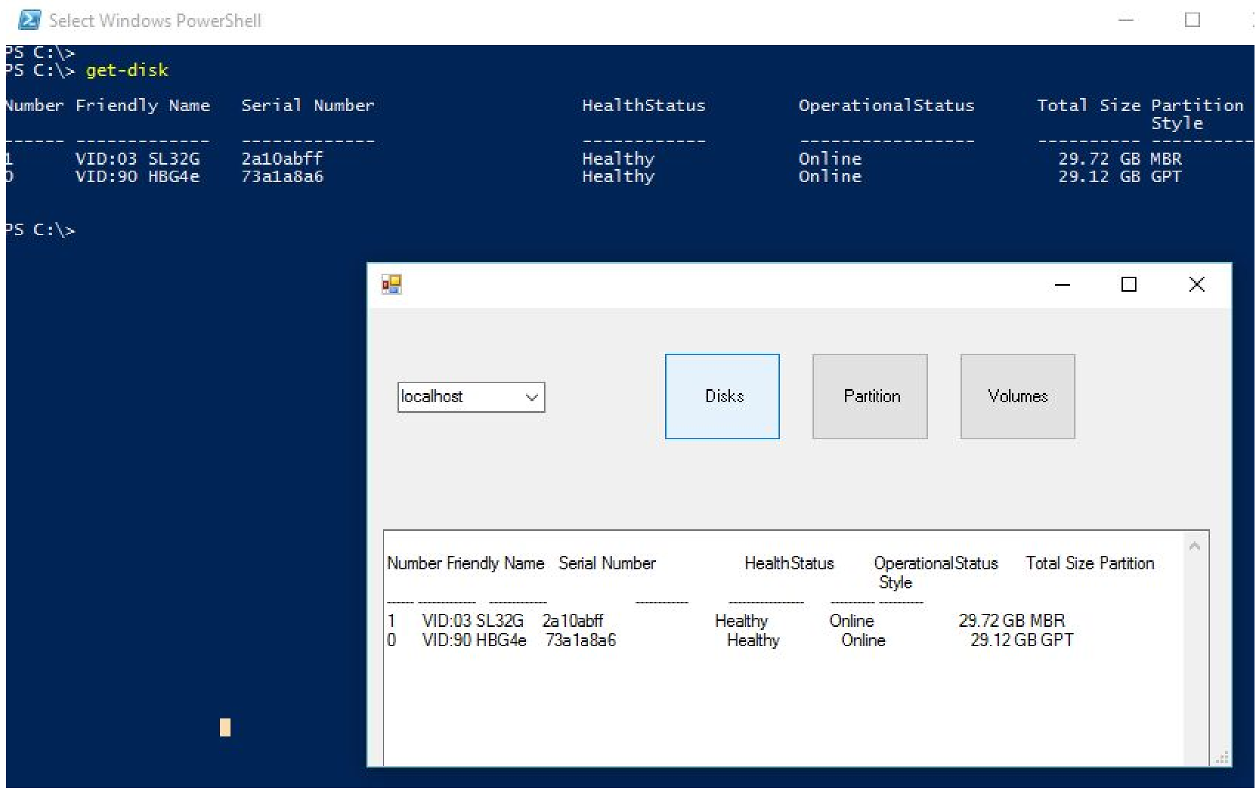


```
DISKPART> list disk

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
  Disk 0    Online           29 GB      0 B        *
* Disk 1    Online           29 GB      0 B
```

And now, a bit of GUI Powershell. I will not put accent on this as I did with the previous ones, but the below graphics offers same information as presented in previous commands.

If you do not like typing it, you might as well enjoy clicking it. xD

Please take in consideration this is only a prototype. A first idea on what powershell could do at storage level.
I do intend to make it look better, and with more options (format, boot from usb, delete partitions, etc)



```
Select Windows PowerShell
PS C:\>
PS C:\> get-disk

Number Friendly Name   Serial Number                    HealthStatus         OperationalStatus      Total Size Partition
                                                                                                               Style
------ -------------   -------------                    ------------         -----------------      ---------- ---------
1      VID:03 SL32G    2a10abff                         Healthy              Online                   29.72 GB MBR
0      VID:90 HBG4e    73a1a8a6                         Healthy              Online                   29.12 GB GPT

PS C:\>
```

```
Number Friendly Name  Serial Number          HealthStatus      OperationalStatus    Total Size Partition
                                                                 Style
------ ------------   ------------      ------------   --------------   --------- ---------
1    VID:03 SL32G    2a10abff                 Healthy        Online              29.72 GB MBR
0    VID:90 HBG4e    73a1a8a6                  Healthy        Online              29.12 GB GPT
```

(source code on next pages)

```
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Drawing")
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")

$Window = New-Object System.Windows.Forms.Form
$Window.Size = New-Object System.Drawing.Size(600,400)

########Function Disks##########

function procInfo {
$computer=$DropDownBox.SelectedItem.ToString() #populate the var with the value you
selected

$diskResult=get-disk |out-string;
$outputBox.text=$diskResult
                    }

#######Function Partition########

function procInfoOne {
$computer=$DropDownBox.SelectedItem.ToString() #populate the var with the value you
selected

$partitionResult=get-partition |out-string;
$outputBox.text=$partitionResult
                    }

#####Function Volume##########

function procInfoTwo {
$computer=$DropDownBox.SelectedItem.ToString() #populate the var with the value you
selected

$volumeResult=get-volume |out-string;
$outputBox.text=$volumeResult
                    }

#####################more functions#######################

$DropDownBox = New-Object System.Windows.Forms.ComboBox
$DropDownBox.Location = New-Object System.Drawing.Size(20,50)
$DropDownBox.Size = New-Object System.Drawing.Size(100,20)
$DropDownBox.DropDownHeight = 200
$Window.Controls.Add($DropDownBox)

$wksList=@("localhost")

foreach ($wks in $wksList) {
                    $DropDownBox.Items.Add($wks)
                    } #end foreach


$outputBox = New-Object System.Windows.Forms.TextBox
$outputBox.Location = New-Object System.Drawing.Size(10,150)
$outputBox.Size = New-Object System.Drawing.Size(560,200)
$outputBox.MultiLine = $True
$outputBox.ScrollBars = "Vertical"
$Window.Controls.Add($outputBox)


###################buttons#############################
$Button = New-Object System.Windows.Forms.Button
$Button.Location = New-Object System.Drawing.Size(200,30)
$Button.Size = New-Object System.Drawing.Size(80,60)
$Button.Text = "Disks"
$Button.Add_Click({procInfo})
$Window.Controls.Add($Button)

$ButtonOne = New-Object System.Windows.Forms.Button
$ButtonOne.Location = New-Object System.Drawing.Size(300,30)
$ButtonOne.Size = New-Object System.Drawing.Size(80,60)
$ButtonOne.Text = "Partition"
$ButtonOne.Add_Click({procInfoOne})
$Window.Controls.Add($ButtonOne)

$ButtonTwo= New-Object System.Windows.Forms.Button
$ButtonTwo.Location = New-Object System.Drawing.Size(400,30)
$ButtonTwo.Size = New-Object System.Drawing.Size(80,60)
$ButtonTwo.Text = "Volumes"
$ButtonTwo.Add_Click({procInfoTwo})
$Window.Controls.Add($ButtonTwo)

$Window.Add_Shown({$Window.Activate()})
[void] $Window.ShowDialog()



###############EOF################
```

Thank you for your time