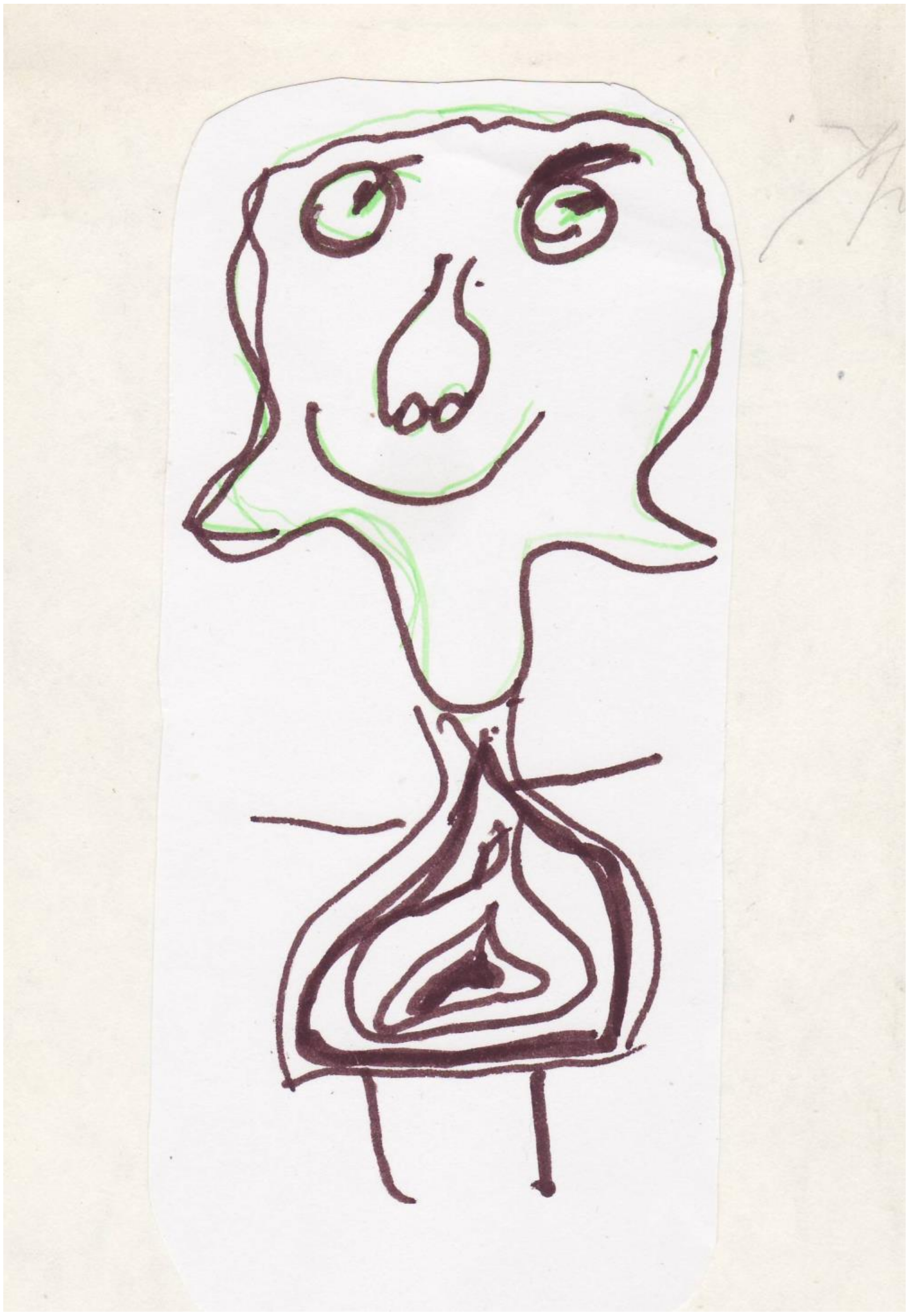
Low level design of the algorithm to enumerate Volumes in the Windows System



Oleg Kulikov, SYSPRG@LIVE.RU

July 2015

Preface

An algorithm to enumerate storage devices, storage volumes and volumes is described. Prototype code in JS, EnumDevicesAndVolumes.wsf, was created according this algorithm. It appeared that EnumDevicesAndVolumes runs perceptibly faster than the base MS Windows utility diskpart.exe under the control of different Windows editions: Windows 7, Windows 8.1, Windows 10 RTM TH1. An attempt to explain the results of the competition is provided but without real provident.

Introduction.

This document presents an algorithm to link physical storage Device with each of its child objects: Storage Volume and Volume.

Preceding giving the detailed account of the algorithm and data-sources on which it is based it is necessary to describe types of the physical objects which algorithm operates.

There are three of them:

- Devices objects, which includes floppies, DVD-ROMs, CD-Changers, Disks, Tapes.

- Storage Volumes, which resides on one of the Device described above. If the Parent object, is a Disk than such Storage Volume named Partition. DVD-ROMs and CD-changers has no child objects among Storage Volumes.

- Volumes: I will use sometimes a name Logical Volumes just for symmetry with Storage Volumes.

Volumes are the fundamental objects for each Operating System: without Volumes, there is no any OS as any OS can be installed into formatted Volume only.

Besides the physical objects mentioned above we will need to link to the Volumes the most valuable properties of the Volume: Mount Point and Mount Name.

Mount Point is the most significant property: it appears as soon as a Volume created and it exists until the Volume is deleted or formatted. There are some ways to present Mount Points but the correct one is based on the Volume GUID, which is created by the OS together with the Volume. GUID Mount Points for the Volumes looks like:

`\??\\Volume\{1036c1c4-0000-0000-007e-000000000000}`	Windows 10 special type of GUID, which contains Parent Disk Signature. Is not used anymore since Build 10240 TH1.
`\??\\Volume\{714ce432-d2a2-11e4-824f-806e6f6e6963}`	Standard Mbr-Style partition Volume GUID
`\??\\Volume\{83983fcd-129f-11e5-82c2-0c607688d174}`	Standard Removables Volume GUID
`\??\\Volume\{a1aeb03a-67c4-4feb-b392-a1a746d349a7}`	Standard Gpt-Style partition Volume GUID

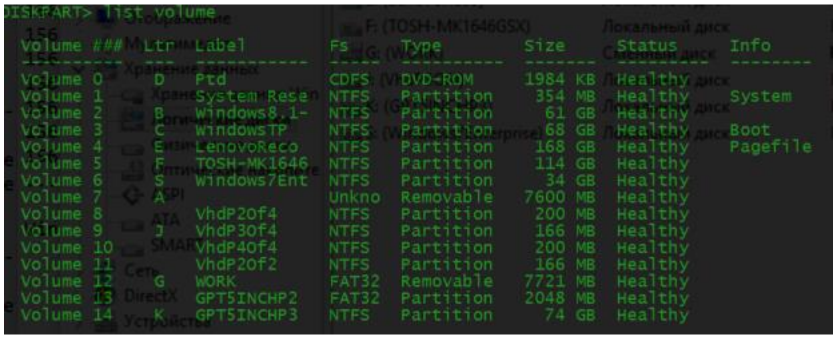
Table 1.

Storage Volumes also has a property, which is presented by GUID. This property is named DiskId despite it relates not only to the Storage Volumes originated from the HD-Type Devices but to the Removable-Type Devices. Removable-Type Devices include Flash-drives and different Memory-cards storage drives (SD-cards mostly these days). Again, it is possible from the 1-st glance to distinguish

GUID related to Type-Removable from the GUID related to Partition-type. Partition-Type Storage Volumes always corresponds to HD-Type Parent Devices and Removable-Type Storage Volumes corresponds to removable-Type Parent Devices. As it was already mentioned above, DVD-ROM-Type Devices has no child Storage Volume though they have child Logical Volume with a rather long PnPDeviceID property instead of the short DiskId GUID.

Logical Volumes GUIDs corresponds to the MountPoints, which were presented in the left column of the table above. MountNames are the short names for the MountPoints with a lot of shortage: any MountPoint can have some different MountNames or have none. MountName can be changed at any moment or even deleted. Certainly, it by no means concerns related MountPoints: any changes to MountNames by no means influence MountPoints.

It is high time to provide some illustrations.



```
DISKPART> list volume

  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  ----------  ---  -----------  -----  ----------  -------  ---------  --------
  Volume 0     D   Ptd          CDFS   DVD-ROM     1984 KB  Healthy
  Volume 1         System Rese  NTFS   Partition    354 MB  Healthy    System
  Volume 2     B   Windows8.1-  NTFS   Partition     61 GB  Healthy
  Volume 3     C   WindowsTP    NTFS   Partition     68 GB  Healthy    Boot
  Volume 4     E   LenovoReco   NTFS   Partition    168 GB  Healthy    Pagefile
  Volume 5     F   TOSH-MK1646  NTFS   Partition    114 GB  Healthy
  Volume 6     S   Windows7Ent  NTFS   Partition     34 GB  Healthy
  Volume 7     A                Unkno  Removable   7600 MB  Healthy
  Volume 8         VhdP2Of4     NTFS   Partition    200 MB  Healthy
  Volume 9     J   VhdP3Of4     NTFS   Partition    166 MB  Healthy
  Volume 10        VhdP4Of4     NTFS   Partition    200 MB  Healthy
  Volume 11        VhdP2Of2     NTFS   Partition    166 MB  Healthy
  Volume 12    G   WORK         FAT32  Removable   7721 MB  Healthy
  Volume 13        GPT5INCHP2   FAT32  Partition   2048 MB  Healthy
  Volume 14    K   GPT5INCHP3   NTFS   Partition     74 GB  Healthy
```

Picture 1.

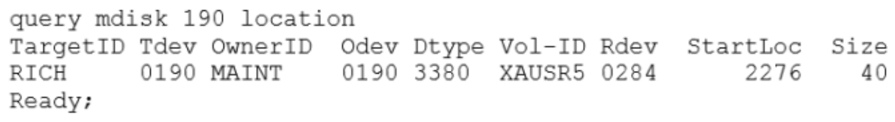
For sure, all of the Windows users perfectly know which Windows utility have produced an output presented by the Picture 1. It demonstrates in the column 2 Volumes names alternate to those mentioned above: no one Volume GUIDs, simple enumeration instead. But HOW this enumeration was created? An answer to the question is not as simple as it seems and the algorithm proposed and described further enables namely the same enumeration of the Volumes as DISKPART.EXE does.

Certainly together with a MountNames, that is Letters. By the way about the Letters: it seems that these days rather few of the readers know WHEN and HOW MountNames as Letters have appeared. Moreover, the most part of the readers are sure that it was MS DOS project, which have presented those notorious Letters for the 1-st time. But just look at the next illustration:

```
LABEL  VDEV M   STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
SMI191 191  A   R/W     25 3380 4096     1325      2747-73       1003       3750
OSTEST 18F1 B/B R/O   3339 3390            OS
VSAM01 300  C   R/W    200 3380           DOS
-      DIR  D   R/W      -    - 4096       31            -          -          -
ESA190 190  S   R/O     86 3380 4096      350      9197-71       3703      12900
ESA19E 19E  Y/S R/O    200 3380 4096      903     14939-50      15061      30000
```

Picture 2.

It is clear that Picture 2 presents practically the same data as the Picture 1. However, Picture 2 have appeared in the beginning of the 1972 when the IBM introduced VM/370 and VM/CMS Operating Systems. Well known fact that there was a rather short period when IBM and Microsoft programmers and programming architectures have worked together and thus it should be no illusions about HOW letters have appeared in the Microsoft Operating Systems. Moreover, Partitions in Microsoft OSes strictly corresponds to VM/370 mini-disks, which corresponds to extents on the parent Disk drive, see picture below.



```
query mdisk 190 location
TargetID Tdev OwnerID  Odev Dtype Vol-ID Rdev  StartLoc  Size
RICH     0190 MAINT    0190 3380  XAUSR5 0284      2276    40
Ready;
```

Picture 3.

Thus, pictures 2 and 3 proves that as Storage Volumes (Partitions) as MountNames (Letters) are originated from IBM VM/370, VM/CMS, April 1972.

In conclusion of the Introduction some words about data-sources: obviously the significant part of the readers know that the most reliable source of the information about Devices are the relevant Device drivers: disk service driver for the HD-Type Devices, cdrom service driver for the DVD-ROM-Type devices, disk or sffdisk driver for the Storage-memory cards and so on. For the Storage Volumes the most reliable source of information is volsnap filter driver and for the Logical Volumes the most reliable source of the data is volmgr filter driver.

However, the question arises: WHERE and HOW get data collected by the drivers mentioned above and which namely data they collects? Again one more analogy with mainframes with channel architecture. All of the OSes on mainframes use two types of the input/output interrupt handlers: FLIH, First Level Interrupt Handler and SLIH, Second Level Interrupt Handler. Execution duration of the 1-st ones costs very high price as when FLIH code executes no other I/O interrupt can happen in the same channel.

Thus FLIH code is a very short code, it collects rather few data and then posts (signals to) the SLIH code. SLIH code does not need to interfere even with an applications code. It has somewhat higher priority to collect detailed data about I/O event as soon as possible.

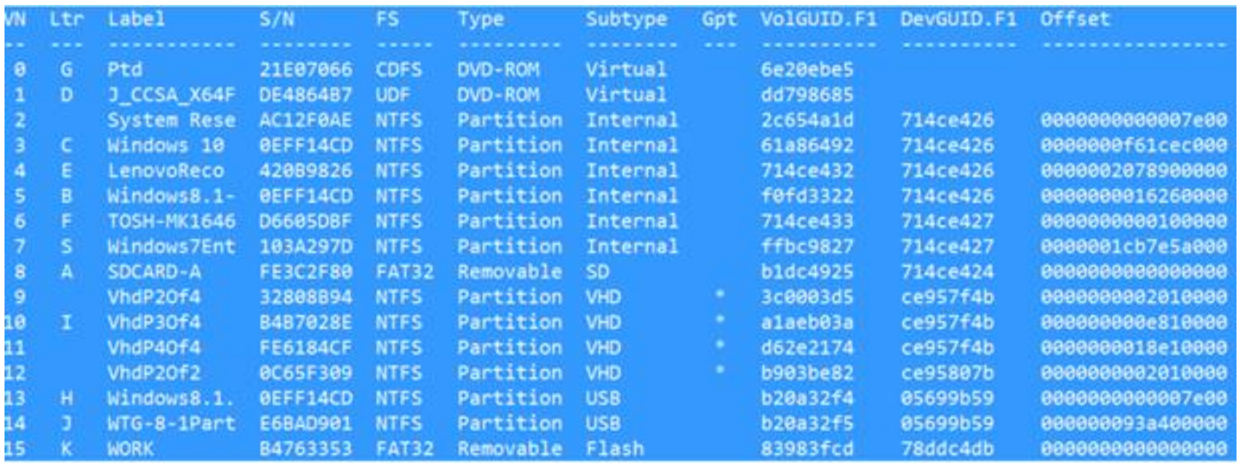
Now return to the Windows architecture which is lack of the channels but Devices Drivers are the same as FLIH in mainframe OSes: Devices drivers had to handle all I/O operations that they service and thus they collects during I/O interrupts as few data as possible to identify the Device which produced an interruption. This minimal data is PnPDeviceID, rather long string which content is defined by the manufacture and as a result appears such a monsters like:

USBSTOR\Disk&Ven_Android&Prod___UMS_Composite&Rev___00#8&a3351a9&0&304D19368321EE1F&1#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}, USBSTOR\Disk&Ven_FUJIFILM&Prod_USB-DRIVEUNIT&Rev_1.00#Y-752^^^^^040913XFPX0008002512&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

More information about service drivers and detailed data collected will be provided in the algorithm description.

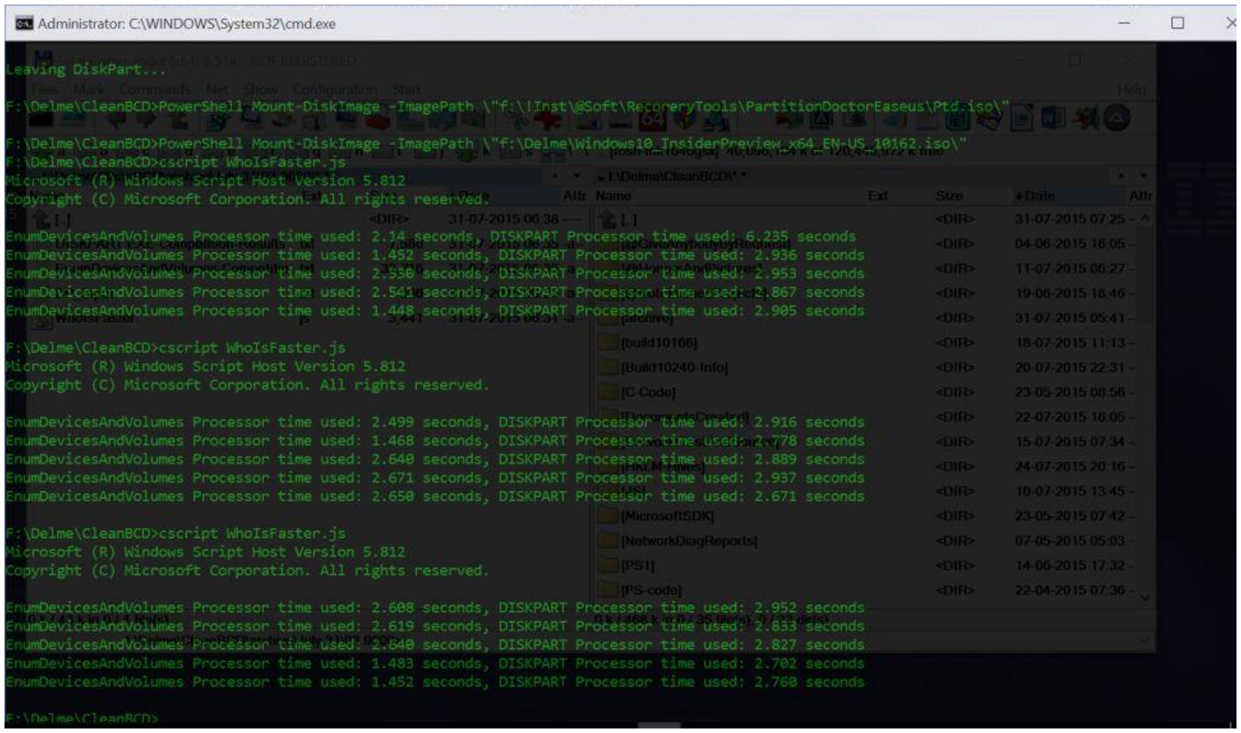
Enumerating Devices and Volumes algorithm description.

Picture 1 in Introduction have presented output of one of the fundamental MS OSes utilities: DISKPART.EXE which functionality permits to handle different types of the Devices and Volumes. In introduction was also said that the main target of the algorithm would be creation of the same Volumes enumeration as DISKPART.EXE does. Two pictures below proves that algorithm solves the task and thus works as designed. Picture 5 shows the competition results under Windows 10 RTM TH1 Build 10240. It proves that EnumDevicesAndVolumes.wsf runs faster than diskpart.exe.



```
VN Ltr Label        S/N      FS    Type      Subtype  Gpt VolGUID.F1 DevGUID.F1 Offset
-- --- ------------ -------- ----- --------- -------- --- ---------- ---------- ----------------
 0  G  Ptd          21E07066 CDFS  DVD-ROM   Virtual      6e20ebe5
 1  D  J_CCSA_X64F  DE4864B7 UDF   DVD-ROM   Virtual      dd798685
 2     System Rese  AC12F0AE NTFS  Partition Internal     2c654a1d   714ce426   0000000000007e00
 3  C  Windows 10   0EFF14CD NTFS  Partition Internal     61a86492   714ce426   0000000f61cec000
 4  E  LenovoReco   420B9826 NTFS  Partition Internal     714ce432   714ce426   0000002078900000
 5  B  Windows8.1-  0EFF14CD NTFS  Partition Internal     f0fd3322   714ce426   0000000016260000
 6  F  TOSH-MK1646  D6605DBF NTFS  Partition Internal     714ce433   714ce427   0000000000100000
 7  S  Windows7Ent  103A297D NTFS  Partition Internal     ffbc9827   714ce427   0000001cb7e5a000
 8  A  SDCARD-A     FE3C2F80 FAT32 Removable SD           b1dc4925   714ce424   0000000000000000
 9     VhdP2Of4     32808B94 NTFS  Partition VHD       *  3c0003d5   ce957f4b   0000000002010000
10  I  VhdP3Of4     B4B7028E NTFS  Partition VHD       *  a1aeb03a   ce957f4b   000000000e810000
11     VhdP4Of4     FE6184CF NTFS  Partition VHD       *  d62e2174   ce957f4b   0000000018e10000
12     VhdP2Of2     0C65F309 NTFS  Partition VHD       *  b903be82   ce95807b   0000000002010000
13  H  Windows8.1.  0EFF14CD NTFS  Partition USB          b20a32f4   05699b59   0000000000007e00
14  J  WTG-8-1Part  E6BAD901 NTFS  Partition USB          b20a32f5   05699b59   000000093a400000
15  K  WORK         B4763353 FAT32 Removable Flash        83983fcd   78ddc4db   0000000000000000
```

Picture 4.



```
Administrator: C:\WINDOWS\System32\cmd.exe

Leaving DiskPart...

F:\Delme\CleanBCD>PowerShell Mount-DiskImage -ImagePath \"f:\!Inst\@Soft\RecoveryTools\PartitionDoctorEaseus\Ptd.iso\"

F:\Delme\CleanBCD>PowerShell Mount-DiskImage -ImagePath \"f:\Delme\Windows10_InsiderPreview_x64_EN-US_10162.iso\"
F:\Delme\CleanBCD>cscript WhoIsFaster.js
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

EnumDevicesAndVolumes Processor time used: 2.14 seconds, DISKPART Processor time used: 6.235 seconds
EnumDevicesAndVolumes Processor time used: 1.452 seconds, DISKPART Processor time used: 2.936 seconds
EnumDevicesAndVolumes Processor time used: 2.536 seconds, DISKPART Processor time used: 2.953 seconds
EnumDevicesAndVolumes Processor time used: 2.541 seconds, DISKPART Processor time used: 2.867 seconds
EnumDevicesAndVolumes Processor time used: 1.448 seconds, DISKPART Processor time used: 2.905 seconds

F:\Delme\CleanBCD>cscript WhoIsFaster.js
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

EnumDevicesAndVolumes Processor time used: 2.499 seconds, DISKPART Processor time used: 2.916 seconds
EnumDevicesAndVolumes Processor time used: 1.468 seconds, DISKPART Processor time used: 2.778 seconds
EnumDevicesAndVolumes Processor time used: 2.640 seconds, DISKPART Processor time used: 2.889 seconds
EnumDevicesAndVolumes Processor time used: 2.671 seconds, DISKPART Processor time used: 2.937 seconds
EnumDevicesAndVolumes Processor time used: 2.650 seconds, DISKPART Processor time used: 2.671 seconds

F:\Delme\CleanBCD>cscript WhoIsFaster.js
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

EnumDevicesAndVolumes Processor time used: 2.608 seconds, DISKPART Processor time used: 2.952 seconds
EnumDevicesAndVolumes Processor time used: 2.619 seconds, DISKPART Processor time used: 2.833 seconds
EnumDevicesAndVolumes Processor time used: 2.640 seconds, DISKPART Processor time used: 2.827 seconds
EnumDevicesAndVolumes Processor time used: 1.483 seconds, DISKPART Processor time used: 2.702 seconds
EnumDevicesAndVolumes Processor time used: 1.452 seconds, DISKPART Processor time used: 2.760 seconds

F:\Delme\CleanBCD>
```

Picture 5.

WhoIsFaster.js runs command lines to start execution of EnumDevicesAndVolumes.wsf and then command line to run diskpart.exe which reads commands to execute from the file prepared by the EnumDevicesAndVolumes.wsf. As write operation is always longer than the read operation and thus diskpart.exe has small benefit.

An algorithm is based on two principles:

1. Use of a natural ordering of the objects according to unique "genetic" property, which child objects gets from the parent objects.
2. Calculating of the direct path to the "details" owner object of the same nature.

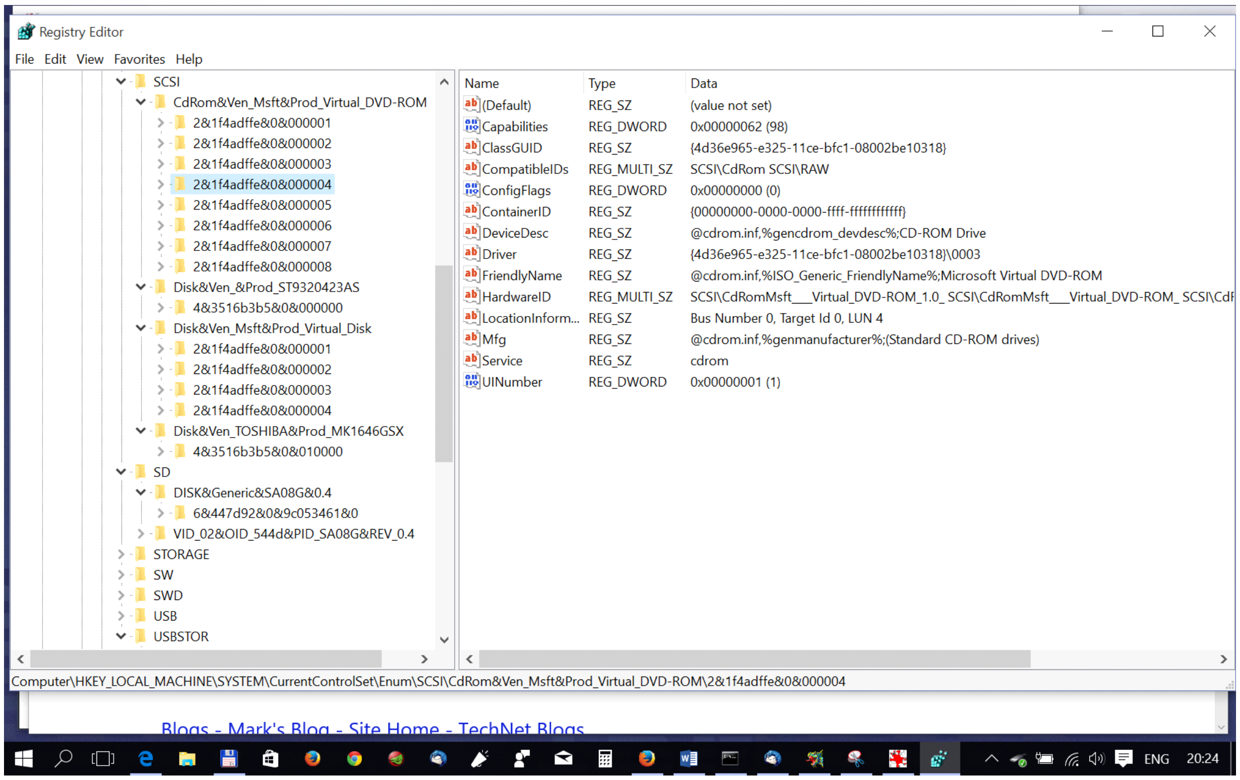
An algorithm described includes four steps. Steps 1, 2 and 3 starts with a read of an ordered enumeration which contains only text-strings. The text-strings parser defines some properties of the objects. On base of these properties, a path to the object, which contains some more properties of the same object, is calculated. These object properties are read and merged with properties of the original object. Step finishes by placement of the objects collected into the special collection of the objects: DevicesCollection on the Step 1, StorageVolumesCollection on the Step 2 and LogicalVolumesCollection on the Step 3.

Algorithm Step 1 Description

Creation of the DevicesCollection

`Data sources:`

1. `Two enumerations from HKLM\SYSTEM\CurrentControlSet\Services: cdrom\Enum and partmgr\Enum. These two enumerations provides full list of the PnPDeviceIDs for all of the storage devices which are currently present.`
2. `HKLM\SYSTEM\CurrentContolSet\Enum + “\” + PnPDeviceID read`
   `HKLM\SYSTEM\CurrentControlSet\Enum\PnPDeviceID\Device Parameters\Partmgr` – contains the most significant property, DiskId which is GUID. It is applicable to all types of the devices other then serviced by cdrom driver, that is DVD-ROM-Type devices.
   `HKLM\SYSTEM\CurrentControlSet\Enum\PnPDeviceID\Device Parameters\Storport –` contains property InitialTimestamp. It can be as timestamp of the device installation as the timestamp of the Manufacture and in this case presents a date when the device was produced.



```
SCSI
  CdRom&Ven_Msft&Prod_Virtual_DVD-ROM
    2&1f4adffe&0&000001
    2&1f4adffe&0&000002
    2&1f4adffe&0&000003
    2&1f4adffe&0&000004
    2&1f4adffe&0&000005
    2&1f4adffe&0&000006
    2&1f4adffe&0&000007
    2&1f4adffe&0&000008
  Disk&Ven_&Prod_ST9320423AS
    4&3516b3b5&0&000000
  Disk&Ven_Msft&Prod_Virtual_Disk
    2&1f4adffe&0&000001
    2&1f4adffe&0&000002
    2&1f4adffe&0&000003
    2&1f4adffe&0&000004
  Disk&Ven_TOSHIBA&Prod_MK1646GSX
    4&3516b3b5&0&010000
SD
  DISK&Generic&SA08G&0.4
    6&447d92&0&9c053461&0
  VID_02&OID_544d&PID_SA08G&REV_0.4
STORAGE
SW
SWD
USB
USBSTOR
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Capabilities	REG_DWORD	0x00000062 (98)
ClassGUID	REG_SZ	{4d36e965-e325-11ce-bfc1-08002be10318}
CompatibleIDs	REG_MULTI_SZ	SCSI\CdRom SCSI\RAW
ConfigFlags	REG_DWORD	0x00000000 (0)
ContainerID	REG_SZ	{00000000-0000-0000-ffff-ffffffffffff}
DeviceDesc	REG_SZ	@cdrom.inf,%gencdrom_devdesc%;CD-ROM Drive
Driver	REG_SZ	{4d36e965-e325-11ce-bfc1-08002be10318}\0003
FriendlyName	REG_SZ	@cdrom.inf,%ISO_Generic_FriendlyName%;Microsoft Virtual DVD-ROM
HardwareID	REG_MULTI_SZ	SCSI\CdRomMsft\___Virtual_DVD-ROM_1.0_ SCSI\CdRomMsft\___Virtual_DVD-ROM_ SCSI\CdF
LocationInform...	REG_SZ	Bus Number 0, Target Id 0, LUN 4
Mfg	REG_SZ	@cdrom.inf,%genmanufacturer%;(Standard CD-ROM drives)
Service	REG_SZ	cdrom
UINumber	REG_DWORD	0x00000001 (1)

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\SCSI\CdRom&Ven_Msft&Prod_Virtual_DVD-ROM\2&1f4adffe&0&000004

3. HKLM\HARDWARE\DESCRIPTION\System\MultifunctionAdapter\0\DiskController\0\DiskPeripheral – is applicable to internal disk drives only and for all Windows editions later then Build 7601 it contains property Identifier on base of which value it became possible to distinguish disks,

which contains Mbr-Style formatted partitions or Gpt-Style formatted partitions. It contains a disk Signature that by the “OS tricks” is hiding for Gpt and 8-zeroes are shown instead of the real disk Signature.

4. `HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id M`, the only property which require 4-level loop which is terminated as soon as N+1 reads run as N+1 returns null. N is number of internal disks. This path gives SerialNumber property which is available even for encrypted devices.

As soon as all of the properties described are collected for a current object, we can place it into the DevicesCollection using DevicesCollection.Count property as an index. But before incrementing Count property we need to initialize some properties for the next steps and create one or two Direct Access Keys in the collection which will be used on the following steps. List of the properties we initialize:

LVGroupStyle	– disk partitions formatting Style: Mbr, Gpt.
NumberOfLV	- number of Volumes on all of the device partitons (the only one for Removables).
LVGroup	- list of the comma-separated Offsets of the Volumes
NumberOfSV	- number of the Storage#Volumes on the device (the only one for Removables)
SVGroup	- list of the comma-separated Offsets of the STORAGE#Volumes

For the Removable-Type devices ‘00000000’ Offset value is used. For DVD-ROM-Type devices, empty Offset value is used.

DCUniqueID – Unuque ID of the Device object. Is equal to the Field 1 of the DiskIds GUID for all the Devices Types but DVD-ROMs and PnPDeviceID with all back slashes replaced by “#” for DVD-ROMs.

OrderIndex – Unique ordering index of the Device object in the DevicesCollection.

Direct access keys created in DevicesCollection

For HD-Type devices as internal as USB we use DiskId GUID Field 1, that is 8 bytes as a Direct Access Key, DevicesCollection[DiskId.F1] = DevicesCollection.Count

For Removable-Type and DVDs we use as a Direct Access Key PnPDeviceID with all slashes replaced by “#”.

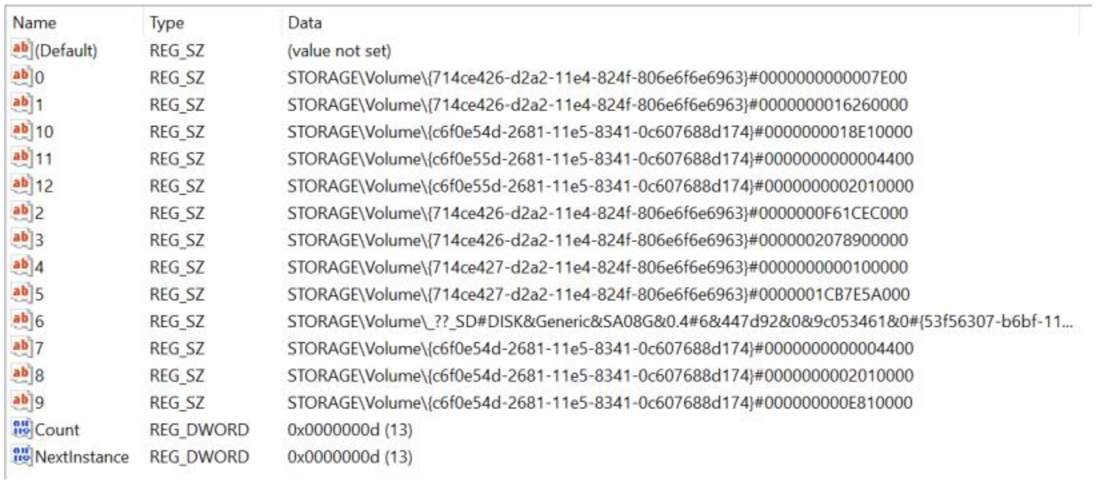
After Direct Access Keys are created we can increment DevicesCollection.Count and continue the loop. The likewise logic is used on the steps 2 and 3.

Only **DCUniqueID** and **OrderIndex** are set at Step 1 and then used at all of the following Steps. Values for the other properties are set on the following Steps.

Algorithm Step 2 Description

On this Step StorageVolumesCollection is created. The logic of this Collection creation is nearly the same as on the Step 1: get at first correctly enumerated long strings likewise PnPDeviceID strings in the Step 1. But here are unpleasant surprises one of which is present in Windows 10 Build 10166 in which environment this document is created. This surprise is really a serious BUG at 1-st appeared in the

Builds numbers 10158, 10159 and then was fixed in the Build number 10162. This BUG could be detected by the rather few of the Windows Insiders who looks into the HKLM\ SYSTEM\CurrentControlSet\Services\volsnap\Enum and here he will see that no one of his USB devices is present in this enumeration. Picture below proves this deplorable fact:



Name	Type	Data
(Default)	REG_SZ	(value not set)
0	REG_SZ	STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000000007E00
1	REG_SZ	STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000016260000
10	REG_SZ	STORAGE\Volume\{c6f0e54d-2681-11e5-8341-0c607688d174}#0000000018E10000
11	REG_SZ	STORAGE\Volume\{c6f0e55d-2681-11e5-8341-0c607688d174}#0000000000004400
12	REG_SZ	STORAGE\Volume\{c6f0e55d-2681-11e5-8341-0c607688d174}#0000000002010000
2	REG_SZ	STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000F61CEC000
3	REG_SZ	STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000002078900000
4	REG_SZ	STORAGE\Volume\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000000000100000
5	REG_SZ	STORAGE\Volume\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000001CB7E5A000
6	REG_SZ	STORAGE\Volume_??_SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0#{53f56307-b6bf-11...
7	REG_SZ	STORAGE\Volume\{c6f0e54d-2681-11e5-8341-0c607688d174}#0000000000004400
8	REG_SZ	STORAGE\Volume\{c6f0e54d-2681-11e5-8341-0c607688d174}#0000000002010000
9	REG_SZ	STORAGE\Volume\{c6f0e54d-2681-11e5-8341-0c607688d174}#000000000E810000
Count	REG_DWORD	0x0000000d (13)
NextInstance	REG_DWORD	0x0000000d (13)

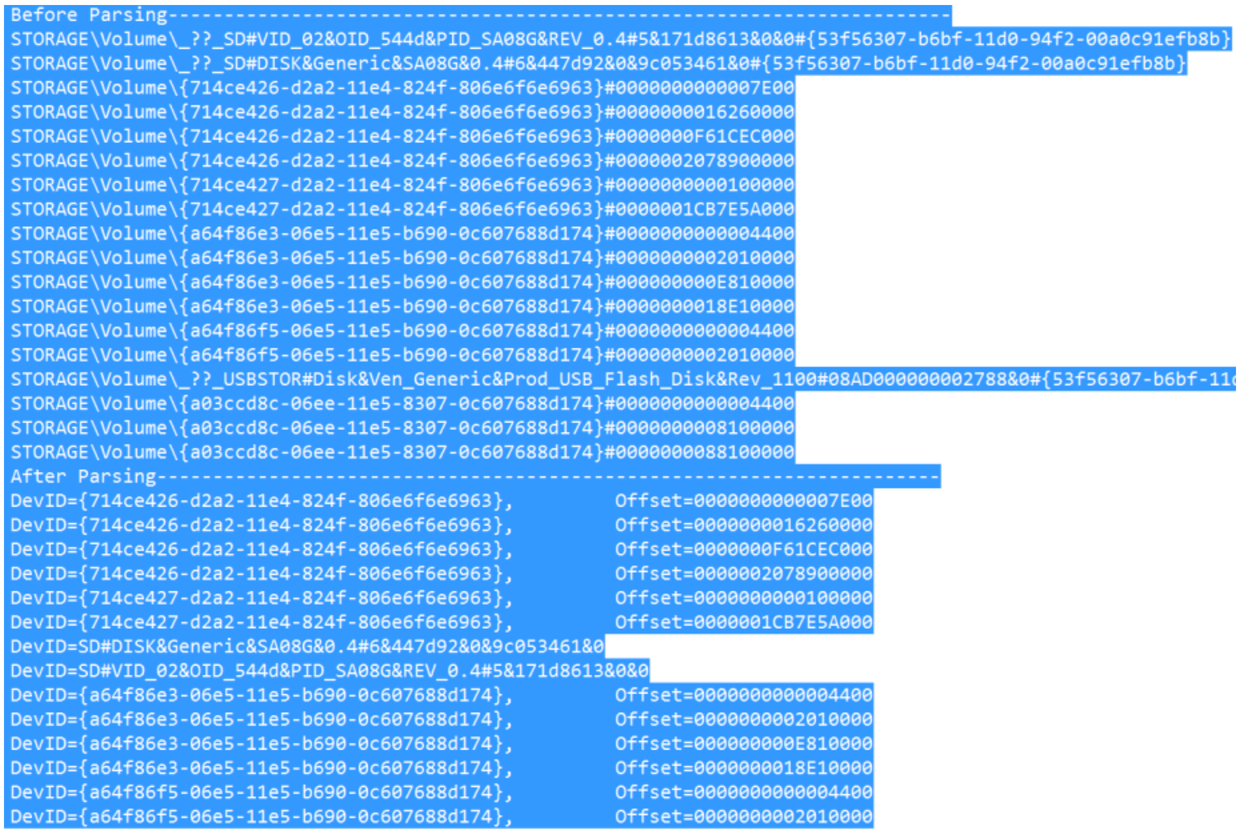
Picture 6.

In the following Windows 10 RTM TH1 Build 10240 this bug was fixed.

The other unpleasant surprise have happened many days ago and resulted in the algorithm redesign. This surprise unpredictably was detected in the Windows 8.1 Registry and even got a special name BecameCrasy, here is this fragment:

```
HKLM\SYSTEM\CurrentControlSet\Services\volsnap\Enum
"0"="STORAGE\\Volume\\_??_SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461#{53f56307-b6bf-11d0-94f2-
00a0c91efb8b}"
"Count"=dword:00000011
"NextInstance"=dword:00000011
"1"="STORAGE\\Volume\\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000000007E00"
"2"="STORAGE\\Volume\\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000016260000"
"3"="STORAGE\\Volume\\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000F61CEC000"
"4"="STORAGE\\Volume\\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000002078900000"
"5"="STORAGE\\Volume\\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000000000100000"
"6"="STORAGE\\Volume\\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000001CB7E5A000"
"7"="STORAGE\\Volume\\{714ce438-d2a2-11e4-824f-806e6f6e6963}#0000000000004400"
"8"="STORAGE\\Volume\\{714ce438-d2a2-11e4-824f-806e6f6e6963}#0000000002010000"
"9"="STORAGE\\Volume\\{714ce438-d2a2-11e4-824f-806e6f6e6963}#000000000E810000"
"10"="STORAGE\\Volume\\{714ce438-d2a2-11e4-824f-806e6f6e6963}#0000000018E10000"
"11"="STORAGE\\Volume\\{120e7228-e2f4-11e4-828d-0c607688d174}#0000000000004400"
"12"="STORAGE\\Volume\\{120e7228-e2f4-11e4-828d-0c607688d174}#0000000002010000"
"13"="STORAGE\\Volume\\_??_USBSTOR#Disk&Ven_Generic&Prod_USB_Flash_Disk&Rev_1100#08AD000000002788&0#{53f5
6307-b6bf-11d0-94f2-00a0c91efb8b}"
"14"="STORAGE\\Volume\\{13f29a4f-fa3f-11e4-82b8-0c607688d174}#0000000000004400"
"15"="STORAGE\\Volume\\{13f29a4f-fa3f-11e4-82b8-0c607688d174}#0000000008100000"
"16"="STORAGE\\Volume\\{13f29a4f-fa3f-11e4-82b8-0c607688d174}#0000000088100000"
```

If you don't have got the reason of the craziness from the 1-st glance – internal SD-card is the 1-st in the list preceding all the partitions of the two internal HDs! It should not happen but it happened. There are possible explanations but instead of wasting time on them let us better describe a logic of the algorithm which permits to forget about order of the data read from the volsnap enumeration and create correct own enumeration. It is more fruitful because of the same enumeration algorithm will be used on the 3-rd Step and thus both Step-2 and Step-3 can reuse the same code. The central role at as Step 2 as Step 3 plays Volumes strings parser, which detects Type of the Parent device presenting by the current string. Picture below demonstrates the results of the parser



```
Before Parsing-----------------------------------------------------------------------
STORAGE\Volume\_??_SD#VID_02&OID_544d&PID_SA08G&REV_0.4#5&171d8613&0&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
STORAGE\Volume\_??_SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000000007E00
STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000016260000
STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000000F61CEC000
STORAGE\Volume\{714ce426-d2a2-11e4-824f-806e6f6e6963}#0000002078900000
STORAGE\Volume\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000000000100000
STORAGE\Volume\{714ce427-d2a2-11e4-824f-806e6f6e6963}#0000001CB7E5A000
STORAGE\Volume\{a64f86e3-06e5-11e5-b690-0c607688d174}#0000000000004400
STORAGE\Volume\{a64f86e3-06e5-11e5-b690-0c607688d174}#0000000002010000
STORAGE\Volume\{a64f86e3-06e5-11e5-b690-0c607688d174}#000000000E810000
STORAGE\Volume\{a64f86e3-06e5-11e5-b690-0c607688d174}#0000000018E10000
STORAGE\Volume\{a64f86f5-06e5-11e5-b690-0c607688d174}#0000000000004400
STORAGE\Volume\{a64f86f5-06e5-11e5-b690-0c607688d174}#0000000002010000
STORAGE\Volume\_??_USBSTOR#Disk&Ven_Generic&Prod_USB_Flash_Disk&Rev_1100#08AD00000002788&0#{53f56307-b6bf-11
STORAGE\Volume\{a03ccd8c-06ee-11e5-8307-0c607688d174}#0000000000004400
STORAGE\Volume\{a03ccd8c-06ee-11e5-8307-0c607688d174}#0000000008100000
STORAGE\Volume\{a03ccd8c-06ee-11e5-8307-0c607688d174}#0000000088100000
After Parsing-----------------------------------------------------------------------
DevID={714ce426-d2a2-11e4-824f-806e6f6e6963},         Offset=0000000000007E00
DevID={714ce426-d2a2-11e4-824f-806e6f6e6963},         Offset=0000000016260000
DevID={714ce426-d2a2-11e4-824f-806e6f6e6963},         Offset=0000000F61CEC000
DevID={714ce426-d2a2-11e4-824f-806e6f6e6963},         Offset=0000002078900000
DevID={714ce427-d2a2-11e4-824f-806e6f6e6963},         Offset=0000000000100000
DevID={714ce427-d2a2-11e4-824f-806e6f6e6963},         Offset=0000001CB7E5A000
DevID=SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0
DevID=SD#VID_02&OID_544d&PID_SA08G&REV_0.4#5&171d8613&0&0
DevID={a64f86e3-06e5-11e5-b690-0c607688d174},         Offset=0000000000004400
DevID={a64f86e3-06e5-11e5-b690-0c607688d174},         Offset=0000000002010000
DevID={a64f86e3-06e5-11e5-b690-0c607688d174},         Offset=000000000E810000
DevID={a64f86e3-06e5-11e5-b690-0c607688d174},         Offset=0000000018E10000
DevID={a64f86f5-06e5-11e5-b690-0c607688d174},         Offset=0000000000004400
DevID={a64f86f5-06e5-11e5-b690-0c607688d174},         Offset=0000000002010000
```

Picture 7.

Picture 7 demonstrates that a strings parser which gets as an argument a string, returns an object which owns property DevID (DeviceID) for each of the input strings and the property Offset for some of the objects. Obviously, that those objects which got a property Offset are the objects of the Partition-Type and the other ones are the objects of the Removable-Type.

Next Sub step of the Steps 2, Step 3 is the detecting of the Parent DevicesCollection object by the "genetic resemblance" key: that is DCUniqueID, which was defined at the Step 1. Very significant string of the Step 1 code was SPECIALLY omitted in the Description of the Step 1 algorithm simply because of this string is in the "JS interpreter Language", here is this string:

this[obj.DCUniqueID] = this.Count;

This string means that during Step 1 Direct Access keys corresponding to the PnPDeviceId was created for Removable-Type objects. PnPDeviceId was somewhat changed before creating those keys: the "#" character replaced Backslashes' in its value. Thus, these Direct Access keys strictly corresponds to the DevID values created by the parser. As for the Partition-Type objects then they use as a Direct Access Key Field 1 of the DevID GUID.
Thus, the most significant part of the task to link Storage Volumes, Logical Volumes to the Parent Devices is solved and we can now define for the Storage Volumes and Logical Volumes their own Unique Ids. At 1-st, each Removable-Type Storage/Logical Volume should get Parent Device Unique Id which name is DCUniqueID. It is easy: having an index returned by the Direct Access key, we can simply copy Parent DCUniqueID to the current object of the Storage Volumes Collection or Logical Volumes Collection. Then using the Parent "genetic card" we can create OWN "genetic card" adding child Unique property, Offset to the Parent genetics. Moreover at the same moment we can add our Offset

into the correspondent Group property of the Parent object and increment the number of the Childs in that group. That is we update Parent objects properties NumberOfSV, SVGroup at Step 2 and NumberOfLV, LVGroup at Step 3.

The description of the collecting detailed properties on the Step 2 is omitted because of nothing interesting is not read. It appeared somewhat interesting only detection of a shortest part of the ANY GUID Field 1, which value does not depend on the Windows version. More details are provided at the end of the document.

Algorithm description of the Enumerating of the objects on the Steps 2 and 3.

An algorithm is very simple: in the preceding part of the Steps 2 and 3 we haven't used Count property of the StorageVolumesCollection and LogicalVolumesCollection. Thus we start loop on the DevicesCollection Count property, for each object of the StorageVolumesCollection and LogicalVolumes collection select objects which Parent is an object of the DevicesCollection: we get an array of the child objects as we use DCUniqueID of the current object of the DevicesCollection. Then we loop on the objects of this array, assign correspondent Collection Count property as an OrderIndex an increment Collection Count property. At the end of the loop all the Childs appeared to be ordered according with Parents. Small exclusion for the StorageVolumesCollection: we omit Parent objects with Type=DVD-ROM as such a Parents could not have StorageVolumes Childs.

Algorithm name is "Algorithm to enumerate Volumes", prototype code name is also the same "EnumDevicesAndVolumes.wsf" then WHAT FOR at all it was necessary to waste a time on enumerating and ordering Storage Volumes which for sure are interesting for Backup / Restore tasks only? The answer is very simple: just for an HONEST competition with DISKPART.EXE despite it is not asked to execute "select part N, detail part" commands but it CAN DO IT. That is if a commercial code need be produced on base of the algorithm proposed then the Step 2 can be omitted or minimized – no one Details property is read from the Registry. As it will be shown during description of the Step 3, Step 2 provides a very useful information, which permit to set for Partition-type objects the Style property value "Mbr" or "Gpt".

Algorithm Step 3 Description

Despite the logic of the Step 3 was somewhat described in the description of the Step 2, it is necessary to start the description with an indication of the fact that on this step there no a correspondent objects as in HKLM\SYSTEM\CurrentControlSet\Services, as in the HKLM\SYSTEM\Enum. As Mounting Volumes operation is an actions of the USERS who can insert their Flash-drives, attach USB HDs, mount ISOs, then we had to search necessary data nor in HKLM at all but in the HKU. In an instrument StdRegWrapperClass, which was already mentioned above, it, was ESPECIALLY for the Step 3 implemented small code into the Constructor, which detects and build CurrentUser object. An object created contains three properties: CurrentUserName, CurrentUserSID and CurrentUserServer. CurrentUserName will be set as a property to the LogicalVolumesCollection objects but just for information of the Server Administrators if they decide to detect who of the Users creates a lot of the junk data in the MountedDevices ☺ Really we urgently need at the beginning of this Step CurrentlyUserSID property. A rather long Registry path is used at this Step:

HKU\<SID>\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\CPC\Volume

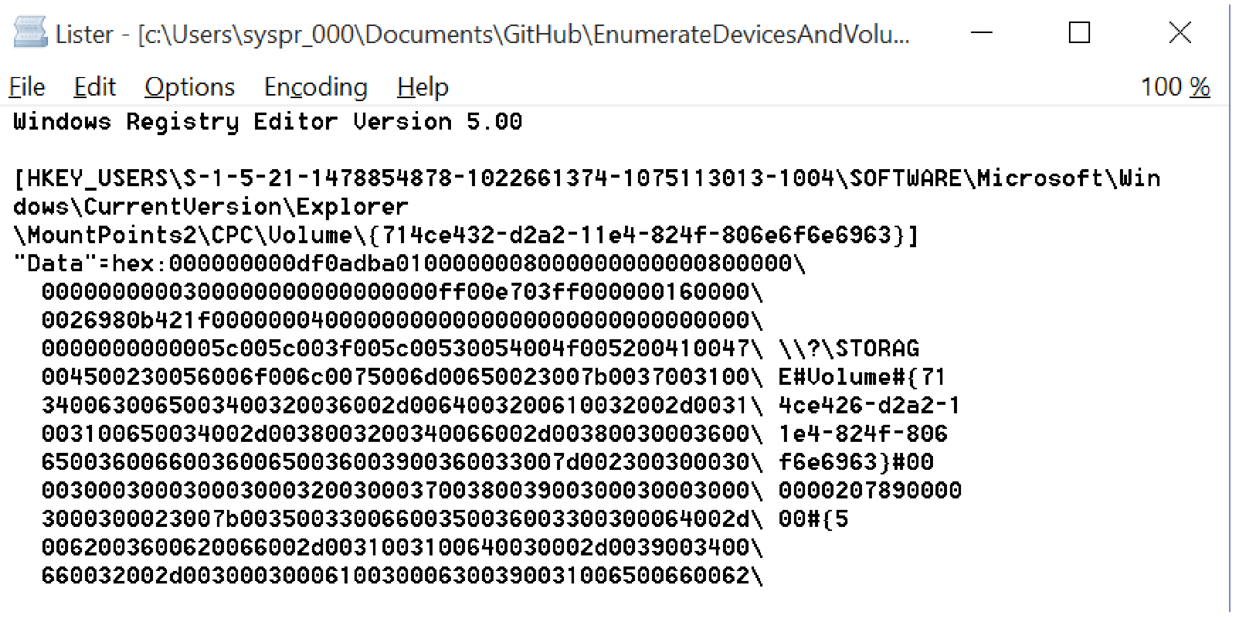
<SID> - stands for the CurrentUserSID which was mentioned above. At 1-st we had to enumerate the path shown above but the sub keys, which we will, get contains an urgently needed data: VolumeGUID. Moreover, enumerated data contains only ACTUAL data. Then we had to read the only value for each of the enumerated sub keys, namely the binary value with a name Data. This is a binary Blob. This Blob presents an object which structure is Windows version dependent. It has a fixed length binary data at the beginning and some UTF-16 strings which offsets in the Blob are Windows version dependent. To parse this Blob function GetBlobs() is used on this step. Function GetBlobs use an algorithm of the splitting Blob data on the parts, which doesn't, depends on the Windows version. GetBlobs function returns an array of the objects and the caller, ClassLV Constructor, assigns properties to the objects created by the Constructor merging to them additional properties delivered by the objects returned by GetBlobs. Because of the properties merging and parse of the DeviceIdentification string with same parser as described in Step 2, we get following properties for the LogicalVolume object:

DevID	- the same DevID as the Storage Volumes has
Offset	- corresponds to the Offset value of ONE of the objects in the StorageVolumesCollection
Type	- for LogicalVolumesCollection Type property includes Partition, Removable, DVD-ROM
VolumeGUID	- this is a property specific for the LogicalVolumes only
Label	- Volume Label
Fs	- File System name
VolumeSN	- Volume Serial Number
Boot	- boolean
System	- boolean
UserName	- Current User Name

Small remark about Current User Name: in Windows 10, don't remember already starting which of the Builds, instead of the User Name StdRegWrapperClass places nor the UserName but the name of the UserProfile in a case if the Microsoft account is used. This profile name is equal to the 1-st 5 characters substring of the Microsoft account name. Thus for SYSPRG@LIVE.RU a profile subdirectory syspr_000 is created during Windows 10 installation.

All other details relating to the linking of the object prepared for placing into the LogicalVolumesCollection follow description given in Step 2. With one small addition: at the end of the loop, we have an opportunity to distinguish Style of formatting partitions on USB disks. It is enough to compare number of the Childs in the SVGroup and LVGroup. If the same values then the disk has Mbr-Style of the formatting and Gpt-Style formatting otherwise. This is because of the 1-st partition on the Gpt-formatted disk can not contail Volume. This is MSR, Microsoft System Reserved partition.

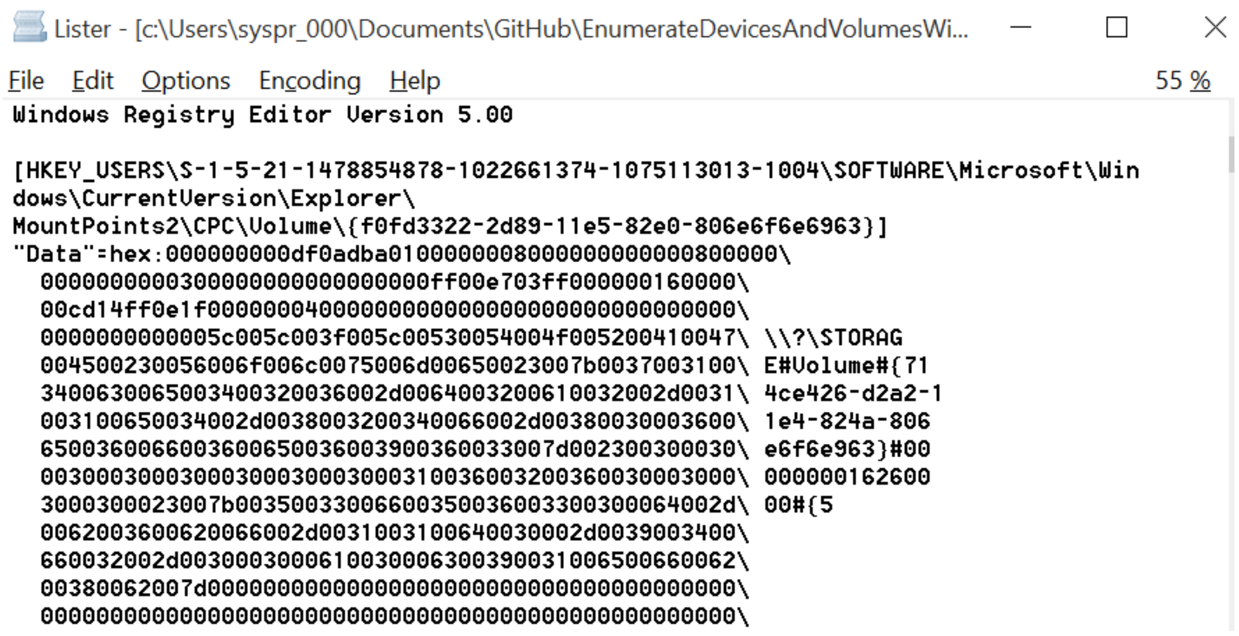
Text above was created two weeks ago when Windows 10 Build 10166 was installed. But then new release Windows 10 RTM TH1 Build 10240 was issued and enumeration `HKU\<SID>\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\CPC\Volume` was changed with a new algorithm. An idea is very pleasant, for the 1-st time Volume GUIDS follow each other according to the mounting of the Volumes. But a small bug happened: for the Mbr-Style formatted disks with 4 partitions (3 primary and one logical) last two correspondent objects offsets are misplaced: greater Offset is preceding the last one with lower Offset. Despite this fact was detected by the algorithm described, I prefer to demonstrate this incorrectness with a snapshots of the data from the Windows Registry to provide reader an opportunity to check it by himself.


Lister - [c:\Users\syspr_000\Documents\GitHub\EnumerateDevicesAndVolu...
File Edit Options Encoding Help
100 %

```
Windows Registry Editor Version 5.00

[HKEY_USERS\S-1-5-21-1478854878-1022661374-1075113013-1004\SOFTWARE\Microsoft\Win
dows\CurrentVersion\Explorer
\MountPoints2\CPC\Volume\{714ce432-d2a2-11e4-824f-806e6f6e6963}]
"Data"=hex:000000000df0adba0100000008000000000000000800000\
  00000000000300000000000000000ff00e703ff000000160000\
  0026980b421f000000040000000000000000000000000000000\
  0000000000005c005c003f005c00530054004f005200410047\ \\?\STORAG
  004500230056006f006c0075006d00650023007b0037003100\ E#Volume#{71
  340063006500340032003600 2d006400320061003200 2d0031\ 4ce426-d2a2-1
  0031006500340 02d003800320034006600 2d003800300036 00\ 1e4-824f-806
  650036006600360065003600390036003300 7d002300300030\ f6e6963}#00
  0030003000300030003200300037003800390030003000 3000\ 0000207890000
  3000300023007b003500330066003500360033003000 64002d\ 00#{5
  00620036006200 66002d00310031006400 30002d0039003400\
  660032002d003000300061003000630039003100650066 0062\
```

Picture 8

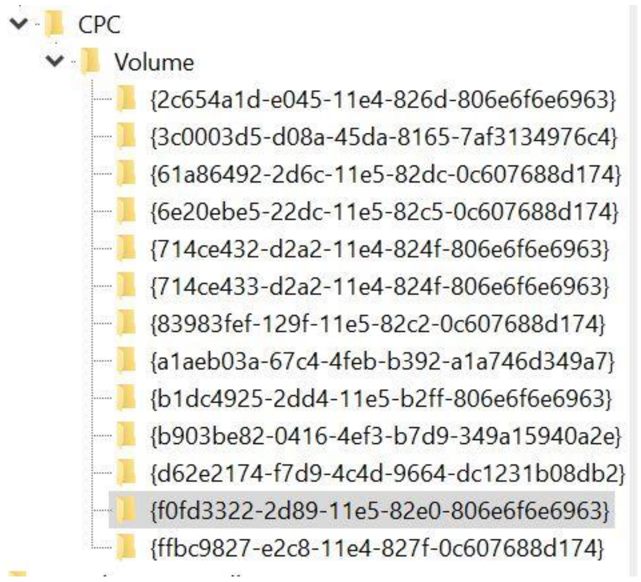

Lister - [c:\Users\syspr_000\Documents\GitHub\EnumerateDevicesAndVolumesWi...
File Edit Options Encoding Help
55 %

```
Windows Registry Editor Version 5.00

[HKEY_USERS\S-1-5-21-1478854878-1022661374-1075113013-1004\SOFTWARE\Microsoft\Win
dows\CurrentVersion\Explorer\
MountPoints2\CPC\Volume\{f0fd3322-2d89-11e5-82e0-806e6f6e6963}]
"Data"=hex:000000000df0adba0100000008000000000000000800000\
  00000000000300000000000000000ff00e703ff000000160000\
  00cd14ff0e1f000000040000000000000000000000000000000\
  0000000000005c005c003f005c00530054004f005200410047\ \\?\STORAG
  004500230056006f006c0075006d00650023007b0037003100\ E#Volume#{71
  3400630065003400320036002d006400320061003200 2d0031\ 4ce426-d2a2-1
  003100650034002d003800320034006600 2d003800300036 00\ 1e4-824a-806
  650036006600360065003600390036003300 7d002300300030\ e6f6e963}#00
  003000300030003000300030003100360032003600300030 00\ 000000162600
  3000300023007b003500330066003500360033003000 64002d\ 00#{5
  0062003600620066002d00310031006400 30002d0039003400\
  660032002d003000300061003000630039003100650066 0062\
  00380062007d000000000000000000000000000000000000000\
  0000000000000000000000000000000000000000000000000\
```

Picture 9

Picture 8 shows that Offset for the 3-rd Volume is equal 00000020789000000 and Picture 9 shows that Offset for the 4-th Volume is equal to 0000000016260000. Following below Picture 10 shows an ordering of the VolumeGUIDS. No comments anymore are necessary and each owner of the Mbr-Style disk with 4 partitions on it can verify by himself.



```
CPC
  Volume
    {2c654a1d-e045-11e4-826d-806e6f6e6963}
    {3c0003d5-d08a-45da-8165-7af3134976c4}
    {61a86492-2d6c-11e5-82dc-0c607688d174}
    {6e20ebe5-22dc-11e5-82c5-0c607688d174}
    {714ce432-d2a2-11e4-824f-806e6f6e6963}
    {714ce433-d2a2-11e4-824f-806e6f6e6963}
    {83983fef-129f-11e5-82c2-0c607688d174}
    {a1aeb03a-67c4-4feb-b392-a1a746d349a7}
    {b1dc4925-2dd4-11e5-b2ff-806e6f6e6963}
    {b903be82-0416-4ef3-b7d9-349a15940a2e}
    {d62e2174-f7d9-4c4d-9664-dc1231b08db2}
    {f0fd3322-2d89-11e5-82e0-806e6f6e6963}
    {ffbc9827-e2c8-11e4-827f-0c607688d174}
```

Picture 10

Step 4 description, “SetMountNamesForMountPoints”

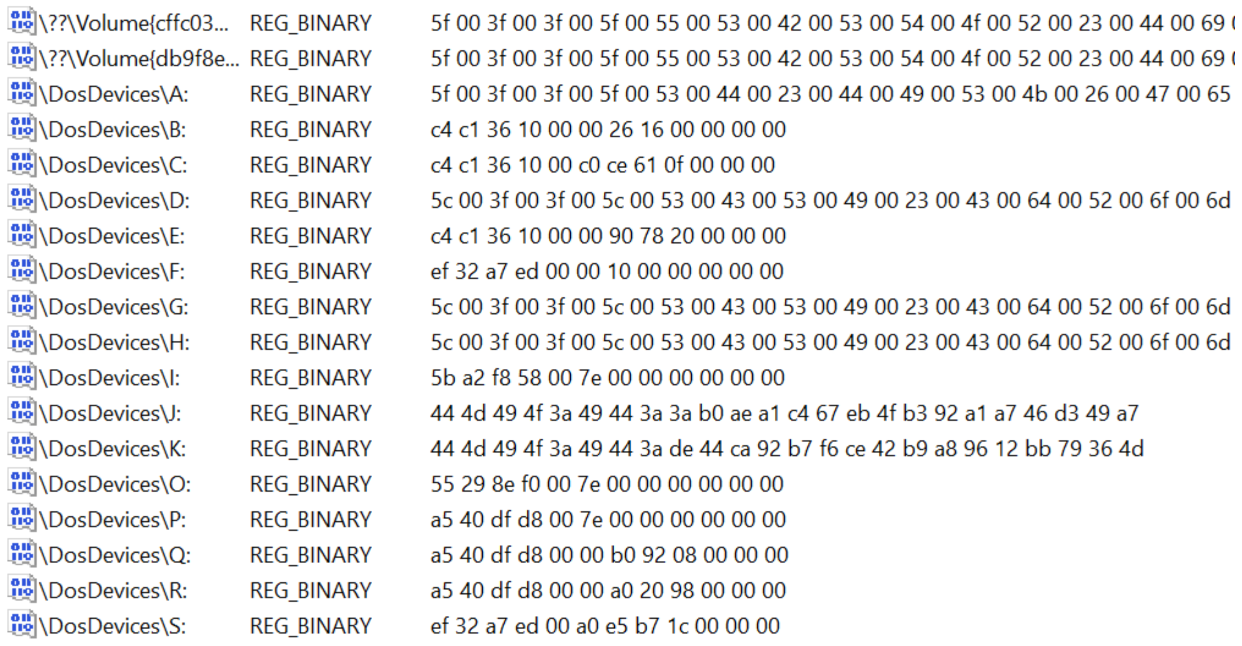
Would it be nor the Windows OS but *NIX then at this point the algorithm description was already finished. However, there are a lot of those who needs MountNames being assigned to the each of the MountPoints.

*NIX users are happy looking on the output of the LVM utility and they need not any Letters as they have PHYSICAL Devices addresses which is enough to address Logical Volumes.

OK, one more step is necessary, we need to read all of the values from the HKLM\SYSTEM\MountedDevices, parse the data read and ASSIGN those Letters which eager to see most part of the Windows users.

The source of information on this Step, well known by the most part of the Windows users is HKLM\SYSTEM\MountedDevices.

When we read all of the values from the MountedDevices then each array element will have two properties: name and value. On the base of these two some other properties could be derived. Look at the Picture 11 which shows part of the MountedDevices values presented by the Regedit.exe.



```
\??\Volume{cffc03...   REG_BINARY    5f 00 3f 00 3f 00 5f 00 55 00 53 00 42 00 53 00 54 00 4f 00 52 00 23 00 44 00 69 (
\??\Volume{db9f8e...   REG_BINARY    5f 00 3f 00 3f 00 5f 00 55 00 53 00 42 00 53 00 54 00 4f 00 52 00 23 00 44 00 69 (
\DosDevices\A:         REG_BINARY    5f 00 3f 00 3f 00 5f 00 53 00 44 00 23 00 44 00 49 00 53 00 4b 00 26 00 47 00 65
\DosDevices\B:         REG_BINARY    c4 c1 36 10 00 00 26 16 00 00 00 00
\DosDevices\C:         REG_BINARY    c4 c1 36 10 00 c0 ce 61 0f 00 00 00
\DosDevices\D:         REG_BINARY    5c 00 3f 00 3f 00 5c 00 53 00 43 00 53 00 49 00 23 00 43 00 64 00 52 00 6f 00 6d
\DosDevices\E:         REG_BINARY    c4 c1 36 10 00 00 90 78 20 00 00 00
\DosDevices\F:         REG_BINARY    ef 32 a7 ed 00 00 10 00 00 00 00 00
\DosDevices\G:         REG_BINARY    5c 00 3f 00 3f 00 5c 00 53 00 43 00 53 00 49 00 23 00 43 00 64 00 52 00 6f 00 6d
\DosDevices\H:         REG_BINARY    5c 00 3f 00 3f 00 5c 00 53 00 43 00 53 00 49 00 23 00 43 00 64 00 52 00 6f 00 6d
\DosDevices\I:         REG_BINARY    5b a2 f8 58 00 7e 00 00 00 00 00 00
\DosDevices\J:         REG_BINARY    44 4d 49 4f 3a 49 44 3a 3a b0 ae a1 c4 67 eb 4f b3 92 a1 a7 46 d3 49 a7
\DosDevices\K:         REG_BINARY    44 4d 49 4f 3a 49 44 3a de 44 ca 92 b7 f6 ce 42 b9 a8 96 12 bb 79 36 4d
\DosDevices\O:         REG_BINARY    55 29 8e f0 00 7e 00 00 00 00 00 00
\DosDevices\P:         REG_BINARY    a5 40 df d8 00 7e 00 00 00 00 00 00
\DosDevices\Q:         REG_BINARY    a5 40 df d8 00 00 b0 92 08 00 00 00
\DosDevices\R:         REG_BINARY    a5 40 df d8 00 00 a0 20 98 00 00 00
\DosDevices\S:         REG_BINARY    ef 32 a7 ed 00 a0 e5 b7 1c 00 00 00
```

Picture 11.

At the left side of the 'REG_BINARY' there are distinguished 3 types of the values: long ones starting with \??\Volume with a following GUID. It is clear that such values of the name property presents standard LogicalVolumes strings. Other type of the name property values starts with '\DosDevices\X:', where X is one of the Letters. It includes as all the existent letters and Volume GUIDs as Volumes and Letters for the devices which ever were used in the running Windows System. The 3-rd type which starts with '#' we will ignore – these are junk elements.

At the right side of the 'REG_BINARY' corresponds to the value property. Three types of the binary strings can be distinguished:

a) 24-heximal characters strings
b) 48-heximal characters strings
c) Unpredictable length string but which length is greater than 48 heximal characters.

This simple analysis provides us the following:

a) corresponds to inverted Parent Disk Signature (8 heximal characters) + 16 bytes Offset of the correspondent LogicalVolume
b) corresponds to the Gpt-Style LogicalVolumes. It is incorrectly to refer Volumes as Mbr/Gpt Style properties are related StorageVolumes according to DISKPART which propose to choose Style of the Partition created as an Mbr or Gpt. But the content of the value field says that the same Style value is applicable to the LogicalVolumes.
c) corresponds to the UTF-16 PnPDeviceIDs with the Backslashes replaced by the "#" character.

48-characters strings presents fixed length header with a fixed content, 16 bytes and 32 characters which presents Gpt-Style Volume GUID in QWORD format. An algorithm for backward converting is given below.

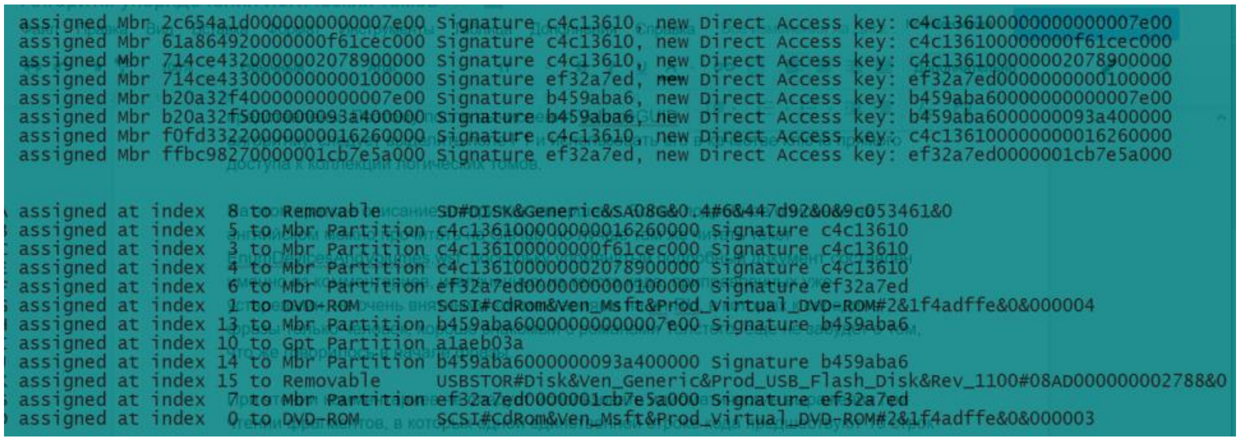
```
String.prototype.ConvertQWORD = function()// converts 32-bytes heximal string
{
//                                 .  .  .  .  x  x  y  y  z  z  t  t  t  t  t  t
//                                 00 02 04 06 08 10 12 14 16 18 20 22 24 26 28 30
// 3ab0aea1c467eb4fb392a1a746d349a7 3a b0 ae a1 c4 67 eb 4f b3 92 a1 a7 46 d3 49 a7
      var t = this.toLowerCase();
      return t.substr( 06, 02 ) + t.substr( 04, 02 ) + t.substr( 02, 02 ) +   t.substr( 00, 02 ) +
             t.substr( 10, 02 ) + t.substr( 08, 02 ) + // 4 xx
             t.substr( 14, 02 ) + t.substr( 12, 02 ) + // 4 yy
             t.substr( 16, 02 ) + t.substr( 18, 02 ) + // 4 zz
             t.substr( 20, 12 ); // 12
}
```

I'm sure that the name of the String.prototype function provides full explanation. One more very simple function converts a value returned by the ConvertQWORD into the valid GUID.

```
String.prototype.buildGuid = function()
{
   if ( this.length != 32 ) return this;

   //   1.3.5.7 8    12   16   20
   // {53f56307-b6bf-11d0-94f2-00a0c91efb8b}
   return "{" + this.substr(  0,  8 ) + "-" +
                this.substr(  8,  4 ) + "-" +
                this.substr( 12,  4 ) + "-" +
                this.substr( 16,  4 ) + "-" +
                this.substr( 20, 12 ) + "}";
}
```

All the other actions on the Step 3 are very simple: each Mbr-Style GUID Field 1 + INVERTED offset value from the value property are used to get an index key in the LogicalVolumesCollection. That is why we have created those short keys containing GUID Field 1 + Offset. If the value is not 'number' then we continue the loop as the current object does not corresponds to any of the actual Volumes. The same for the DosDevices strings which contains Signature and offset in the value: invert Signature 8-bytes, invert Offset 16 bytes, join the string and use as a Direct Access Key to the LogicalVolumesCollection. If the value for the key has type 'number' than we need this object, otherwise – continue the loop.



```
assigned Mbr 2c654a1d0000000000007e00 Signature c4c13610, new Direct Access key: c4c136100000000000007e00
assigned Mbr 61a864920000000f61cec000 Signature c4c13610, new Direct Access key: c4c136100000000f61cec000
assigned Mbr 714ce43200000002078900000 Signature c4c13610, new Direct Access key: c4c1361000000002078900000
assigned Mbr 714ce43300000000000100000 Signature ef32a7ed, new Direct Access key: ef32a7ed000000000100000
assigned Mbr b20a32f40000000000007e00 Signature b459aba6, new Direct Access key: b459aba60000000000007e00
assigned Mbr b20a32f5000000093a400000 Signature b459aba6, new Direct Access key: b459aba6000000093a400000
assigned Mbr f0fd33220000000016260000 Signature c4c13610, new Direct Access key: c4c1361000000000016260000
assigned Mbr ffbc98270000001cb7e5a000 Signature ef32a7ed, new Direct Access key: ef32a7ed0000001cb7e5a000


assigned at index  8 to Removable     SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0
assigned at index  5 to Mbr Partition c4c1361000000000016260000 Signature c4c13610
assigned at index  3 to Mbr Partition c4c136100000000f61cec000 Signature c4c13610
assigned at index  4 to Mbr Partition c4c1361000000002078900000 Signature c4c13610
assigned at index  6 to Mbr Partition ef32a7ed000000000100000 Signature ef32a7ed
assigned at index  1 to DVD-ROM       SCSI#CdRom&Ven_Msft&Prod_Virtual_DVD-ROM#2&1f4adffe&0&000004
assigned at index 13 to Mbr Partition b459aba60000000000007e00 Signature b459aba6
assigned at index 10 to Gpt Partition a1aeb03a
assigned at index 14 to Mbr Partition b459aba6000000093a400000 Signature b459aba6
assigned at index 15 to Removable     USBSTOR#Disk&Ven_Generic&Prod_USB_Flash_Disk&Rev_1100#08AD00000002788&0
assigned at index  7 to Mbr Partition ef32a7ed0000001cb7e5a000 Signature ef32a7ed
assigned at index  0 to DVD-ROM       SCSI#CdRom&Ven_Msft&Prod_Virtual_DVD-ROM#2&1f4adffe&0&000003
```

Picture 12.

Picture 12 demonstrates assignments on the step described above.

Many thanks for your patience. Title page picture is portrait of the GUID as imagined by Anna Shakh-Nazarova, 6 years old.