Алгоритм упорядочения логических томов

Код-прототип на JavaScript уверенно обгоняет по скорости базовую системную утилиту DISKPART.EXE при решении задачи упорядочения логических томов

Описан алгоритм, реализация которого в прототип-коде на JavaScript уверенно опережает по времени решения задачи опорную системную утилиту DISKPART.EXE. Описаны принципы на которых базируется алгоритм. Дано возможное объяснение, но не доказательство, тому, почему DISKPART.EXE решает задачу медленней, чем JavaScript код. Приводятся состязательные результаты в среде Windows различных изданий, включая Windows 10 RTM TH1. Описывается как изменения, вносимые в различных изданиях Windows 10 вынуждали несколько раз переделывать алгоритм с целью повышения его надежности, хотя и ценой некоторой потери скорости.

Вступление.

Логические тома лежат в основе любой ОС. До тех пор, пока нет логических томов, систему просто негде размещать. Подготовка логического тома включает выделение пространства на физическом устройстве хранения данных (то есть создание тома данных, Storage#Volume, зачастую не совсем правильно именуемого Разделом, Partition), а затем создание файловой системы, внутри которой и размещаются компоненты любой ОС. Ниже излагается алгоритм решения задачи упорядочения логических том применительно к ОС Windows, но в полном описании алгоритма, размещенном на GitHub, производится сравнение с другими ОС.

Принципы, на котором базируется описываемый алгоритм.

В основу описываемого алгоритма в качестве основных, заложены два принципа.

1. Использование “естественной сортировки” по уникальному, “генетическому” свойству, приобретаемому объектами-детьми от объектов-родителей.

2. Вычисление “адреса” объекта, носителя детальной информации об объекте, отстоящем от родителя на один уровень, на основе “генетического кода” родительского объекта.

Каждый объект, отстоящий на один уровень от родителя, приобретает целый ряд уникальных свойств, отличающих его от других объектов, порожденных тем же самым родителем, как-то: дата и время рождения, уникальный символьный идентификатор, уникальный числовой идентификатор, символьное, обобщенное имя, технические характеристики, связанные с объемом хранимых данных и др.

Во избежание коллизий в случае, если в системе появятся два устройства с абсолютно одинаковыми идентификаторами, присвоенными производителями оборудования, к каждому такому идентификатору при первом появлении устройства в системе, привязывается “бирка с номером”.

Идентификация устройств хранения данных производителями оборудования и операционными системами.

Производители оборудования, выпускающие жесткие диски, флэш-накопители, DVD и CD накопители разных подклассов (внутренние, внешние, виртуальные, только для чтения, для чтения и записи, с возможностью выбора носителя информации), Накопители на магнитной ленте, Флоппи-дисководы.

Производители оборудования (Manufactures), наделяют свои изделия уникальным, неуничтожимым и существующим с момента выпуска на все время жизни, идентификатором. Идентификатор, называемый в системе “PnPDeviceID”, представляется достаточно длинной текстовой строкой, разбитой на подгруппы символом “&”. Подгруппы содержат имя производителя, наименование конкретного изделия, номер версии, класс изделия и другое.

Как только любое из устройств хранения данных впервые было опознано конкретной операционной системой, соответствующее устройство (за одним исключением), приобретает еще один уникальный, существенно более короткий и строго формализованный идентификатор, называемый GUID. Длина GUID составляет 42 ASCII-символа из которых 6 символов являются служебными. В связи с этим значимыми являются лишь 32 16-ричных символа, немедленно превращаемых системой в 8-ми байтное число, QWORD, по определенному алгоритму. Эти восьмибайтные числа с использованием системного алгоритма хеширования используются далее для “прямого доступа” к соответствующим объектам и для упорядочения объектов, путем создания перечислений.

Исключение. Как минимум одним из устройств хранения данных, которому не присваивается GUID, является класс устройств с обобщенным именем “DVD-ROM”, обслуживаемый в среде ОС Windows любых изданий и редакций драйвером “cdrom”, HKLM\SYSTEM\CurrentControlClass\Services\cdrom\Enum.

Для таких устройств присвоенный ОС идентификатор PnPDeviceID является единственным идентификатором устройства в системе.

Разрешение коллизий в среде ОС Windows.

ОС Windows поступает точно так же, нянечки в родильных домах, привязывая некую бирку к руке новорожденного: ОС приписывает к идентификатору от производителя еще и “бирку с номером” на тот случай, если ОС придется опознавать и присваивать PnPDeviceID еще одному аналогичному устройству от того же производителя. Эта «бирка с номером” приписывается через стандартный разделитель “&” последней. Тем самым гарантируется, что в системе никогда не возникнет двух устройств с абсолютно одинаковыми PnPDeviceID.

PnPDeviceID создается абсолютно для всех устройств хранения данных, впервые опознанных системой. Но для всех устройств, отличных от “DVD-ROM”, как уже было сказано выше, система создает еще и GUID. Как же в этом случае различаются GUID для двух устройств с абсолютно одинаковыми идентификаторами от производителя, куда вешается “бирка с номером” в GUID? Ответ следующий: “бирка с номером” это последние два 16-ричных символа первого поля GUID. Более детальное пояснение будет дано ниже при сравнении отчетов программы в среде 8.1 и Windows 10 RTM TH1.

Дополнительные уникальные свойства устройств хранения данных.

Применительно к логическим томам:

Серийный номер тома, SerialNumber. Присваивается ОС абсолютно всем устройствам логическим томам, включая даже виртуальные “DVD-ROM”. Серийный номер представляется 8 16-тиричными символами. В этом поле возможны коллизии, от которых невозможно избавиться, и система не пытается это сделать, поскольку нет в том необходимости. Серийный номер “00000000” приобретают все логические тома, которые были полностью зашифрованы (Encrypted) тем или иным способом, и система не может прочитать серийный номер тома до тех пор, пока том не будет смонтирован с указанием кода декодирования. Таким образом 8 нулей в поле SerialNumber логического тома уверенно свидетельствуют о том, что этот том зашифрован.

Применительно к устройствам хранения данных:

Серийный номер устройства, SerialNumber. Присваивается производителем устройства. 8 16-ричных символов. Хранится в системе даже для устройств, единственный логический том на которых был тем или иным способом зашифрован.

Подпись диска, Signature. Присваивается производителем устройства. 8 16-ричных символов. Все ОС блокируют отображение в системе подписи диска в том случае, если на диске размещены Gpt-форматированные разделы. Это несомненно системный трюк, позволяющий без использования низкоуровневых операций ввода-вывода для диска, по наличию или отсутствию подписи диска определить Стиль (Style) форматирования разделов на диске: Mbr/Gpt. Очень изящно это демонстрируется в свойствах диска, хранимых в HKLM\HARDWARE\DESCRIPTION\System\MultifunctionAdapter\0\DiskController\0\DiskPeripheral, где хранится свойство "Identifier"="9cb5ff90-00000000-A". 8 нулей, несомненно специально записанных ОС на место реальной подписи диска, свидетельствуют о том, что разделы на диске имеют стиль форматирования Gpt.

Результаты выполнения под управлением разных версий Windows.

Windows 10 RTM TH1 Build 10240		
N	EnumDevicesAndVolumes.wsf	DISKPART.EXE
1	2.140	6.235
2	1.452	2.936
3	2.530	2.953
4	2.541	2.867
5	1.448	2.905

Windows 8.1 Build 9600		
N	EnumDevicesAndVolumes.wsf	DISKPART.EXE
1	1.243	5.274
2	1.222	2.558
3	1.218	1.289
4	1.200	1.379
5	1.379	1.425

Windows 7 SP1 Build 7601		
N	EnumDevicesAndVolumes.wsf	DISKPART.EXE
1	1.406	11.913
2	1.319	2.130
3	1.338	2.662
4	1.310	2.788
5	1.287	2.698

Небольшой по размеру JavaScript код, WhoIsFaster.js, последовательно запускал на выполнение (WShell.Exec) командные строки, соответствующие исполнению EnumDevicesAndVolumes.wsf и DISKPART.EXE. В ходе исполнение той и другой программы считывались строки, выводимые программами на StdOut и считанные

строки, буферизовались с целью предельно точно измерить процессорное время исполнения кода. Замеры производились с точностью до миллисекунды и в таблицах всюду фигурируют секунды и миллисекунды.

К сожалению, указанные цифры отображают не только процессорное время: EnumDevicesAndVolumes.wsf по завершению своей основной работы и вывода всех отчетов, создавал в текущей директории еще и входной файл, используемый в командной строке для запуска DISKPART.EXE. Так что к процессорному времени, использованному обеими программами следует добавить еще и время, затраченное на дисковый ввод-вывод. Поскольку время записи на диск заведомо больше, чем время считывания, то по этому параметру DISKPART.EXE имел некоторое преимущество.

Аппаратная платформа.

Thinkpad R500, 2.4Gz, 8 Gb RAM, ATI Mobility Radeon HD 3400, 1650x1080, два внутренних диска.
Первый, основной – скорость вращения 7200, четыре раздела.
Второй диск установлен в трее для DVD-ROM, скорость вращения 5400, два раздела.

Прикладные программы, установленные в различных версиях Windows.

Windows 7 Enterprise Evaluation. Практически “пуста”, установлены лишь Total Commander и Free Javascript Editor. Эти же программы установлены и в других системах.

Windows 8.1: MS Office 2013, Acronis True Image 2015, Acronis Disk Director 12, пара программ от AOMEI для создания бэкапов и для обслуживания разделов. Перечислены лишь те из прикладных программ, которые запускают собственные драйвер-фильтры, создающие перечисления в HKLM\SYSTEM\CurrentControlSet\Services.

Windows 10: начиная с Technical Preview Build 10049, когда стало возможным обновление Windows 8.1 без потери установленных прикладных программ и их настроек, актуальная на тот момент версия 8.1 была обновлена. В дальнейшем Windows 10 претерпевала обновления всеми официально выпускаемыми сборками и поэтому состав прикладного софта в средах 9600 и 10240, полностью совпадает.

Интерпретация результатов.

Соревновательные результаты в среде совершенно “пустой” Windows 7 Enterprise Evaluation, несколько уступающие аналогичным в среде хорошо “начиненной” приложениями с собственными драйвер-фильтрами, как представляется, развеивают миф о высокой скорости семерки в сравнении с восьмеркой. Алгоритмы, используемые в среде Build 9600, …,10240 – почти полностью одинаковы. В среде 7601 алгоритм незначительно усложнен обработкой данных сервиса sff_disk, обслуживающего SD-карты, но зато в среде 7601 не используется цикл

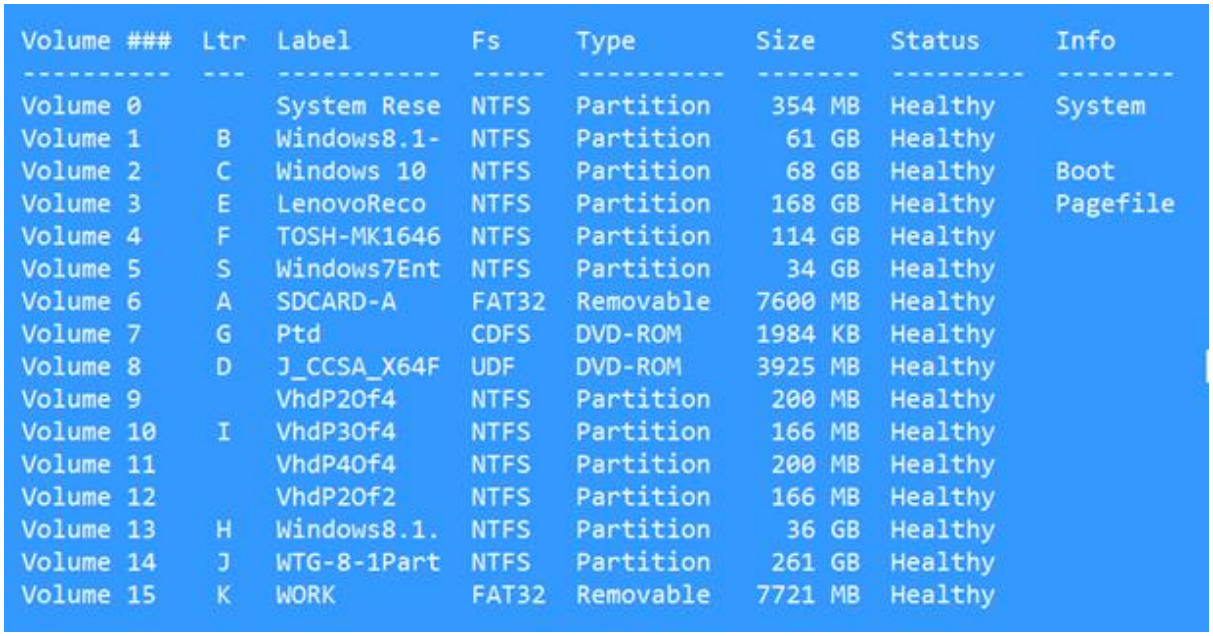
с вложенностью 4, используемый в среде 9600+, используемый для считывания уникальных свойств внутренних дисков, поскольку под 7601 этих свойств в Registry просто нет.

Чем можно объяснить ощутимо меньшую скорость работы DISKPART.EXE в сравнении с интерпретируемым кодом EnumDevicesAndVolumes.wsf в среде Windows 8.1? Есть лишь одно, бездоказательное объяснение. Как уже было упомянуто, для всех устройств хранения данных и созданных на них томах данных и логических томах, система использует QWORD-ключи (8 байт) для работы с хеш-таблицами.

Описываемый алгоритм использует для тех же целей лишь поле Field 1 GUID, превращаемого системой в QWORD. Не хватает знаний и данных для оценки зависимости скорости доступа к системной хеш-таблице, но представляется, что эта зависимость нелинейная и в лучшем случае обратно пропорциональна N*log(N), где N – длина ключа в битах или байтах. Таким образом именно короткие ключи, используемые описываемым алгоритмом, могут объяснить отставание DISKPART.EXE.

До появления Windows 10 RTM TH1 Build 10240, соревновательные результаты в среде различных сборок лишь незначительно отличались от аналогичных в среде 8.1. Однако в среде сборки 10240 был принципиально изменен алгоритм перечисления ключей HKU\SID\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\CPC\Volume: если начиная с 7601 и заканчивая 10166 “дети ходили семьями”, то есть все логические тома, созданные на одном и том же физическом диске, строго следовали один за другим в порядке возрастания смещения от начала диска, то в среде 10240 было впервые внедрено “естественное” упорядочение логических томов в котором тома перечисляются строго в порядке их монтирования в системе. Это, несомненно революционное изменение должно было ускорить работу системы с логическими томами. Вот только DISKPART.EXE, как представляется, еще не пользуется этим перечислением и все также выталкивает в первые ряды устройства, обслуживаемые драйвером cdrom. Соревновательные результаты в среде сборки Windows 10 RTM вполне сопоставимы с аналогичными в среде Windows 8.1.

Визуальное представление результатов решения задачи



Volume ###	Ltr	Label	Fs	Type	Size	Status	Info
Volume 0		System Rese	NTFS	Partition	354 MB	Healthy	System
Volume 1	B	Windows8.1-	NTFS	Partition	61 GB	Healthy	
Volume 2	C	Windows 10	NTFS	Partition	68 GB	Healthy	Boot
Volume 3	E	LenovoReco	NTFS	Partition	168 GB	Healthy	Pagefile
Volume 4	F	TOSH-MK1646	NTFS	Partition	114 GB	Healthy	
Volume 5	S	Windows7Ent	NTFS	Partition	34 GB	Healthy	
Volume 6	A	SDCARD-A	FAT32	Removable	7600 MB	Healthy	
Volume 7	G	Ptd	CDFS	DVD-ROM	1984 KB	Healthy	
Volume 8	D	J_CCSA_X64F	UDF	DVD-ROM	3925 MB	Healthy	
Volume 9		VhdP2Of4	NTFS	Partition	200 MB	Healthy	
Volume 10	I	VhdP3Of4	NTFS	Partition	166 MB	Healthy	
Volume 11		VhdP4Of4	NTFS	Partition	200 MB	Healthy	
Volume 12		VhdP2Of2	NTFS	Partition	166 MB	Healthy	
Volume 13	H	Windows8.1.	NTFS	Partition	36 GB	Healthy	
Volume 14	J	WTG-8-1Part	NTFS	Partition	261 GB	Healthy	
Volume 15	K	WORK	FAT32	Removable	7721 MB	Healthy	

Рисунок 1.

Именно этот вывод информации о логических томах программой DISKPART.EXE и явился “задачей для подражания” и создания быстрого алгоритма решения. Размещенные ниже рисуноки 2 и 3 представляют один из многих отчетов, создаваемых EnumDevicesAndVolumes.wsf. Это самые короткие из всех отчетов, создаваемых EnumDevicesAndVolumes.wsf. Тем не менее даже этот отчет содержит больше информации, чем приведенный выше отчет DISKPART и не содержит бессмысленной информации, как в столбце Status: естественно, что все тома здоровы, а иначе бы пришлось заниматься их лечением, а не соревнованиями в скорости решения задачи.

VN	Ltr	Label	S/N	FS	Type	Subtype	Gpt	VolGUID.F1	DevGUID.F1	Offset
0	H	Ptd	21E07066	CDFS	DVD-ROM	Virtual		83983fef		
1	I	J_CCSA_X64F	DE4864B7	UDF	DVD-ROM	Virtual		6e20ebe5		
2		System Rese	AC12F0AE	NTFS	Partition	Internal		2c654a1d	714ce426	0000000000007e00
3	C	Windows8.1-	0EFF14CD	NTFS	Partition	Internal		714ce430	714ce426	0000000016260000
4	D	Windows 10	0EFF14CD	NTFS	Partition	Internal		714ce431	714ce426	0000000f61cec000
5	E	LenovoReco	420B9826	NTFS	Partition	Internal		714ce432	714ce426	0000002078900000
6	F	TOSH-MK1646	D6605DBF	NTFS	Partition	Internal		714ce433	714ce427	0000000000100000
7	S	Windows7Ent	103A297D	NTFS	Partition	Internal		ffbc9827	714ce427	0000001cb7e5a000
8	A		00000000		Removable	SD		714ce42e	714ce424	0000000000000000
9		VhdP2Of4	32808894	NTFS	Partition	VHD	*	3c0003d5	714ce438	0000000002010000
10	G	VhdP3Of4	B4B7028E	NTFS	Partition	VHD	*	a1aeb03a	714ce438	000000000e810000
11		VhdP4Of4	FE6184CF	NTFS	Partition	VHD	*	d62e2174	714ce438	0000000018e10000
12		VhdP2Of2	0C65F309	NTFS	Partition	VHD	*	b903be82	120e7228	0000000002010000
13	J	Windows8.1.	0EFF14CD	NTFS	Partition	USB		b20a32f4	b20a32f0	0000000000007e00
14	K	WTG-8-1Part	E6BAD901	NTFS	Partition	USB		b20a32f5	b20a32f0	000000093a400000
15	L	WORK	B4763353	FAT32	Removable	Flash		83983fcd	83983fca	0000000000000000

Рисунок 2.

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

Рисунок 3.

Рисунок 2 представляет краткий отчет EnumDevicesAndVolumes.wsf в среде Windows 8.1, а очень похожий рисунок 3 представляет аналогичный отчет в среде Windows 10 RTM TH1. Разница обнаруживается лишь в значениях полей F1: обратите внимание на “714ce4NN” и Вы заметите, что лишь последние два символа обеспечивают различие DevGUID.F1 для разных Mbr-форматированных дисков, SD-карты и разделов первого из двух Gpt-форматированных VHD. Более того, такой же формат имеют и разные VolGUID.F1, соответствующие разделам на Mbr-форматированных дисках. Но вот в среде Windows 10, где, как уже упоминалось, были изменены алгоритмы создания VolumeGUID, “714ce4NN” приписан лишь двум томам, одному на первом диске, метка System Reserved и одному на втором диске, метка TOSH-MK1646GSX. Более детальный анализ выявления “генетического родства” различных GUID, JS-скрипт EnumDevicesAndVolumes.wsf представляет в специально создаваемом отчете, создаваемом методом ShowResemblanse().

Заключительные замечания к первой части публикации.

Объясню использование “Windows Registry” вместо “Системный Реестр” тем, что в моем представлении русское слово “реестр” аналогично “амбарной книге” и именует “плоский файл”, а никак не иерархическую базу данных, коей является Windows Registry.

Почему “код-прототип”? Изначально в подражание DISKPART.EXE планировалось написать FAD-код, “Fast And Dirty”, но жизнь и Microsoft сначала вынудили убрать слово Fast, ну а потом уже я сам решил сделать код почище и пригодным для быстрого переписывания на C, C++, C#. Но все же код остается ПРОТОТИПОМ в том смысле, что не обеспечивает никакого функционала применительно к объектам, размещенным кодом в трех коллекциях.

Во второй части планируется описать четыре шага алгоритма, первые три из которых строго следуют двум принципам, провозглашенным в начале этой публикации. Каждый из первых трех шагов начинается со считывания некоторого перечисления, продолжается извлечением детальных свойств из некоторого “мусоросборника” и завершается размещением детализированных объектов, наделенных всеми необходимыми свойствами в соответствующей коллекции.

Впрочем желающие могут не дожидаясь второй части посмотреть и выкачать EnumDevicesAndVolumes.wsf, WhoIsFaster.js по ссылке

https://github.com/sysprg-46/EnumerateDevicesAndVolumes/tree/master

Часть 2, краткое описание алгоритма.

Как алгоритм хранит данные.

В процессе выполнения четырех шагов алгоритма, создаются три коллекции, содержащие объекты представляющие Устройства хранения данных, DevicesCollection, создается самой первой, на шаге 1 и постоянно используется на всех последующих шагах для обращения к родительскому объекту. На шаге 2 создается StorageVolumesCollection, а на шаге 3, LogicalVolumesCollection. Все коллекции организованы по единому принципу: все хранимые объекты проиндексированы и для того, чтобы обратиться непосредственно к хранимому объекту, надо просто обратиться к XXXXXCollection[index], где XXXXX: { Devices, StorageVolumes, LogicalVolumes }, а индекс – любое целое число в диапазоне [0, XXXXXCollection.Count]. Свойство Count содержит количество объектов соответствующего типа в коллекции.

Строки описания устройств, хранимые в бинарном блобе Data, не содержат DeviceGUID не только для DVD-ROM, но и для Removables. Упомянутый бинарный блоб содержит детальные характеристики логического тома и адресуется Registry путем:

HKU\SID\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\CPC\Volume\GUID, где GUID это VolumeGUID, SID – Security Identifier текущего пользователя. В связи с этим коллекция DevicesCollection и LogicalVolumesCollection содержат “ключи прямого доступа” как в виде коротких 8-ми символьных DeviceGUID.F1, так и длинных, непредсказуемой длины для DVD-ROM и Removables. Ключи прямого доступа возвращают индекс объекта, соответствующего указанному ключу. Таким образом используя длинный ключ, PnPDeviceID для обращения к DevicesCollection, можно получить индекс родительского устройства. Это индекс немедленно присваивается в качестве значения свойства ParentIndex тому объекту типа StorageVolume или LogicalVolume, который надо разместить в собственной коллекции.

Описание первого шага алгоритма: создание DevicesCollection.

Источники информации:

1. Перечисления, создаваемые службами cdrom и partmgr. Трудяга partmgr мало того, что получает информацию от нескольких драйверов устройств и выполняет свою главную задачу – создание перечисления томов памяти, дополнительно создает еще и перечисление все устройств кроме тех, которые обслуживает служба cdrom. Понятно, что речь идет о считывании всех REG_SZ значений по путям:

   HKLM\SYSTEM\CurrentControlSet\Services\cdrom и
   HKLM\SYSTEM\CurrentControlSet\Services\partmrg.

2. Извлечение детальной информации об устройствах на основу PnPDeviceID. Настал момент дополнить описание PnPDeviceID, представленное в первой части, еще и

упоминанием о том, что PnPDeviceID обязательно содержит в качестве префикса к строке описания устройства, еще и префикс, указывающий интерфейс, согласно которому устройство подключено. Причем я не могу сказать уверенно, приписывает ли этот префикс Windows к тому описанию устройства, которое предоставляет производитель или сам производитель. В любом случае, PnPDeviceID предоставляет точную информацию о том, по какому пути следует считать детальные данные из HKLM\SYSTEM\CurrentControlSet\Enum. Продемонстрируем этот важнейший источник информации об устройствах, снимком экрана.

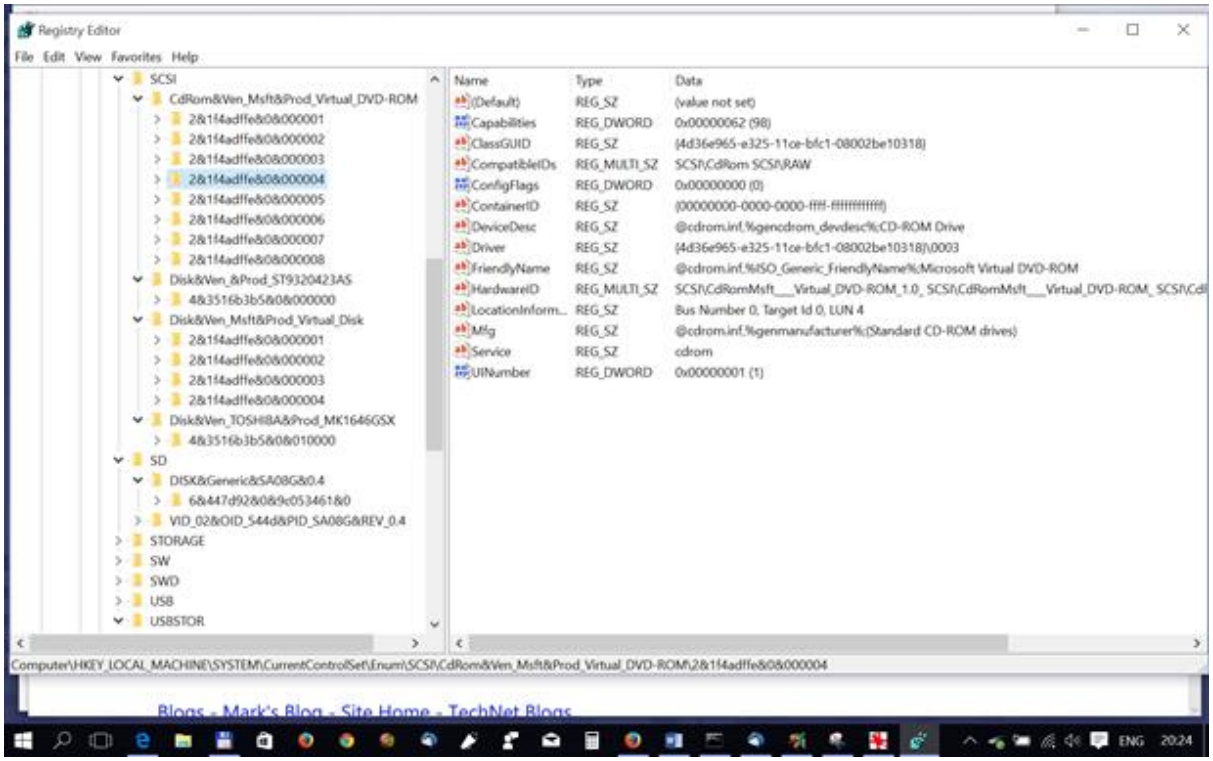


Рисунок 4.

Помимо HKLM\SYSTEM\CurrentControlSet\Enum\PnPDeviceID, дополнительно используется еще два пути:

HKLM\SYSTEM\CurrentControlSet\Enum\PnPDeviceID\Device Parameters\Partmgr, содержащий значение важнейшего свойства DiskId, которое всюду ранее упоминалось в качестве DevGUID.

Еще раз напомним, что для устройств, обслуживаемых службой cdrom, DiskId не создается и поэтому алгоритм присваивает свойству DevID значение PnPDeviceID с символами “\” замененными на символ “#”.

HKLM\SYSTEM\CurrentControlSet\Enum\PnPDeviceID\Device Parameters\Storport содержит InitialTimestamp, дату выпуска устройства производителем в формате REG_QWORD.

Дополнительными источниками информации исключительно для внутренних дисков, являются пути:

HKLM\HARDWARE\DESCRIPTION\System\MultifunctionAdapter\0\DiskController\0\DiskPeripheral
HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id M.

По первому из путей считывается REG_SZ свойство Identifier, представляющее для Mbr-форматированных дисков подпись диска, а для Gpt-форматированных содержит нули, что и свидетельствует о том, что диск содержит Gpt-форматированные разделы. Это очень важная информация для алгоритма, но, к сожалению, эта информация доступна лишь для внутренних дисков.

После того, как объект, представляющий устройство хранения данных полностью наделен всеми свойствами, считанными из Registry и дополнительными свойствами, нужными алгоритму, объект наделяется важнейшим свойством OrderIndex, значение которого устанавливается равным DevicesCollection.Count. После этого объект помещается в коллекцию по индексу равному значению свойства коллекции Count и Count инкрементируется. Дополнительно создаются ключи прямого доступа для всех Removables и DVD-ROM, а также инициализируются групповые свойства, реальные значения которых устанавливаются на втором и третьем шагах. Перечислим здесь эти свойства:
LVGroupStyle – стиль форматирования, возможные значения Mbr, Gpt.
NumberOfLV - количество логических томов, размещенных на разделах устройства.
LVGroup - список смещений логических томов
NumberOfSV - количество разделов, Storage#Volume, размещенных на устройстве
SVGroup - список смещений разделов

Описание второго шага алгоритма: создание StorageVolumesCollection.

Источники информации:

1. Перечисление HKLM\SYSTEM\CurrentControlSet\services\volsnap\Enum
   ОБЯЗАН предоставлять упорядоченный список Storage#Volume, томов памяти. Однако случаются казусы даже в такой вылизанной системе, как Windows 8.1. Самое время продемонстрировать сохраненный фрагмент Registry, хранимый с именем файла BecameCrazy-volsnap.reg. Вот фрагмент из этого файла.

   [HKEY_LOCAL_MACHINE\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"

   Обратите внимание на строку “0”, соответствующую SD-карте. Каким образом этой карте был присвоен такой индекс, можно объяснить тем, что видимо случился сбой при чтении второго раздела первого внутреннего диска, система обрабатывала сбой, выставляла бит Dirty для раздела и потому отчет partmgr о томе памяти на SD-карте опередил поступление отчетов о томах памяти на внутренних дисках. К чести системы

должен уточнить, что случилось это лишь однажды, могу назвать даже точную дату – 01.07.2015.

Но проще не сетовать на неожиданные глюки, а создавать алгоритмы, позволяющие правильно работать независимо от сюрпризов. Поэтому казус вынудил немедленно пересмотреть алгоритм: если раньше алгоритм всецело доверял порядку строк, считываемых по указанному пути, то после случившегося в алгоритм был заложен принцип самостоятельного упорядочения в соответствии с родительским индексом и собственным смещением.

Этот же принцип применяется и на шаге 3 при создании коллекции логических томов, и он гарантирует, что упорядочение будет правильным, если только не станет “дурить” еще и служба partmgr и смещения перестанут поступать в порядке возрастания. Именно это и случилось под Windows 10 RTM TH1, в которой, как уже было сказано, был изменен алгоритм упорядочения Volume GUID. Естественно, что в начале это вызвало у меня бурные эмоции и всякие недоброжелательные слова в адрес программистов от MS: не то, чтобы используемый алгоритм перестал работать, но уж очень резал глаз неупорядоченный список смещений для LVGroup. И лишь через пару часов я осознал, ЧЕМ было вызвано это изменение алгоритма, что оно принесло и с того момента лишь пою дифирамбы автору идеи нового алгоритма упорядочения.

2. HKLM\SYSTEM\CurrentControlSet\Enum\STORAGE\Volume_??_SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}

   Содержит следующие свойства:

```
"Capabilities"=dword:000000b0
"ContainerID"="{33b560f6-21b5-11e5-b2ff-806e6f6e6963}"
"HardwareID"=hex(7):530054004f0052004100470045005c0056006f006c0075006d00650000000000
"ClassGUID"="{71a27cdd-812a-11d0-bec7-08002be2092f}"
"Service"="volsnap"
"DeviceDesc"="@volume.inf,%storage\\volume.devicedesc%;Generic volume"
"Driver"="{71a27cdd-812a-11d0-bec7-08002be2092f}\\0000"
"Mfg"="@volume.inf,%msft%;Microsoft"
"ConfigFlags"=dword:00000000
```

Большинство из этих свойств может быть заранее “вписано ручками”. Интерес представил лишь ContainerID, уникальное свойство, но лень было терять время на поиск применения этого GUID, скорее всего представляющего GUID драйвера. Так что не уверен, что вообще имеет смысл тратить драгоценные миллисекунды на считывание этих свойств. Гораздо важнее немедленно связать объект, представленный строкой описания тома памяти с родительским устройством. Поэтому еще до того, как считывается информация из HKLM\SYSTEM\CurrentControlSet\Enum\STORAGE\Volume\, осуществляется разбор текстовой строки, описывающей том памяти с целью определить значения следующих свойств:

Type = { “Partition”, “Removable” },
Subtype = { “SD”, “Flash”, “Virtual” },
Offset, DevID

Последнее из свойств устанавливается равным полю F1 GUID, предшествующего смещению для Type=”Partition”. В противном случае отсекается префикс “STORAGE\Volume\” и суффикс “#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}” так что приведенная выше строка из файла с именем “crazy” для SD-карты превращается в:

_??_SD#DISK&Generic&SA08G&0.4#6&447d92&0&9c053461&0

Эта строка присваивается в качестве свойства DevID. Ну а далее вычисляется важнейшее свойство для любого из “детей”, определение родительского устройства:

obj.ParentIndex = DevicesCollection[obj.DevID]

Узнав индекс родительского устройства, можно для Removables узнать DevGUID родителя, точнее DevGUID алгоритму не нужен, а нужно лишь значение поля F1 от DevGUID. Это значение родитель хранит в качестве значения собственного свойства DCUniqueID. Зная родительский DCUniqueID, можно сформировать и значение собственного уникального идентификатора:

obj.SVUniqueID = DevicesCollection[obj.ParentIndex].DCUniqueID + obj.Offset

Для томов памяти типа “Partition” используется реальное смещение, а для томов памяти, размещенных на Removables, смещение устанавливается равным “00000000”.
Теперь можно обновить свойства родительского объекта, описывающих SVGroup и количество томов памяти, размещенных на родительском устройстве. То есть инкрементировать счетчик DevicesCollection[obj.parentIndex].NumberOfSV и включить смещение текущего объекта в счетчик DevicesCollection[obj.parentIndex].SVGroup.

Совершенно аналогичные действия выполняются и на шаге три, применительно к логическим томам.

Теперь напомню, что мы находимся в цикле обработки строк, считанных из перечисления. Мы уже определили все необходимые свойства объекта класса StorageVolumesCollection, но памятуя о том, что нельзя доверять правильности перечисления, сохраним объект не в целевой коллекции, а в простенькой, временной. Но текущий объект вообще говоря является членом группы и потому по родительскому индексу во временной коллекции размещается не сам объект, а группа объектов, каждый из которых имеет того же родителя.

По завершению цикла считывания перечисления, мы имеем возможность создать правильно упорядоченную целевую коллекцию StorageVolemesCollection. Для этого мы выполним цикл по родительской коллекции, DevicesCollection и несколько коротеньких вложенных циклов по группам, имеющим одного и того же родителя. Внутри цикла мы осуществляем размещение объектов из временной коллекции в StorageVolumesCollection, используя в качестве индекса для размещения StorageVolumesCollection.Count и инкрементируя это свойство после размещения объекта. StorageVolumesCollection самая простая по принципам формирования, не нуждается в создании дополнительных ключей прямого доступа.

Описание третьего шага алгоритма: создание LogicalVolumesCollection.

Источники информации:

Перечисление
HKU\SID\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2\CPC\Volume\GUID

где GUID это VolumeGUID, SID – Security Identifier текущего пользователя, предоставляет список VolumeGUID, а контент бинарного блоба с именем Data, адресуемый указанным ключом, содержит детальные данные. Блоб Data имеет достаточно сложную структуру, содержащую как UTF-16 текстовые строки, так и двоичные данные. Кроме того, размер и смещения текстовых полей внутри этого блоба зависят от версии Windows, младше сборки 9600 или старше. В коде-прототипе разборку контента этого блоба осуществляет функция GetBlobs. Используемый алгоритм построен так, что отсутствует зависимость от версии Windows, не используются уже известные смещения текстовых полей. К уже сказанному выше при описании шага два, добавлю лишь то, что в данном случае устройства типа DVD-ROM не игнорируются, поскольку они содержат логический том.

На шаге три важнейшее значение имеет создание ключей прямого доступа вида:

для Mbr-форматированных: подпись диска + смещение
для Mbr-форматированных: VolGUID.F1 + смещение
для Gpt-форматированных: VolumeGUID.F1

Однако в тот момент, когда выполняется цикл обработки объектов для включения в коллекцию LogicalVolumesCollection, еще не известны подписи для USB-дисков и, кроме того, неизвестны СТИЛИ форматирования Mbr/Gpt как для разделов на VHD, так и на внешних дисках.

Поэтому стоит упомянуть специальный вид VolumeGUID, который появился первоначально в WMI классе MSFT_Volume в поле Path, где наряду с реальным VolumeGUID, присутствовал и фиктивный, имеющий формат {1036c1c4-0000-0000-007e-000000000000}. Затем в одной из сборок, может даже 9926, такой формат GUID появился и просуществовал во всех сборках от 9879 до 10166, а вот в 10240 – опять исчез в связи с изменением алгоритма генерации VolumeGUID.

Упоминаю указанный формат потому, что поле F1 этого GUID представляет собой подпись диска, а поля F4, F5 – смещение логического тома. Так что обнаружив в системе такой VolumeGUID, становилось возможным присвоить уникальные ключи прямого доступа к объектам LogicalVolumesCollection всем членам LVGroup. Еще одной из причин, по которой я упоминаю здесь приведенный формат VolumeGUID состоит в том, что не смотря на несомненную революционность идеи об упорядочении VolumeGUID в соответствии с порядком монтирования логических томов, впервые внедренную в сборке 10240, тем не менее алгоритм генерации самих GUID оказался неудачным в том смысле, что для Mbr-дисков,

имеющих четыре раздела и соответственно четыре логических тома, два последних поступают в НЕПРАВИЛЬНОМ ПОРЯДКЕ: последний имеет МЕНЬШЕЕ смещение, чем предпоследний.

Желающие могут сами в этом убедиться, самостоятельно разобрав контент Data, а в описываемом алгоритме пришлось вставить оператор копирования списка SVGroup на место списка LVroup, что существенно быстрее, чем превращение списка LVGroup в массив, сортировка массива и последующая сборка в список.

И наконец стоит отметить, что по окончании цикла по объектам, предназначенным для размещения в коллекции логических томов, у нас появляется возможность окончательно разобраться со стилями форматирования разделов на внешних дисках и на VHD. Для этого достаточно сравнить значения свойств NumberOfSV и NumberOfLV. Совпадение значений однозначно свидетельствует о том, что разделы на диске имеют стиль форматирования Mbr, а вот NumberOfLV < NumberOfSV однозначно свидетельствует о стиле форматирования Gpt, поскольку первый из Gpt-разделов, MSR, не может содержать логического тома.

Описание четвертого шага алгоритма: назначение букв логическим томам.

Источник информации: HKLM\SYSTEM\MountedDevices

Даже начинающим пользователям Windows хорошо известен этот путь, хранящий, однако наряду с нужной информацией еще и очень много совершенно не нужной. Особенно это касается USB-флэшек, а также телефонов, большинство из которых сейчас оснащено micro-SD. Информация о всех таких устройствах очень быстро засоряет MountedDevices. Следует иметь в виду, что на каждое из таких устройств в MountedDevices заводится по две записи: одна, в которой в поле name имеет значение \??\Volume\VolumeGUID и вторая, в которой поле name имеет значение формата \DosDevices\Q:. Поле value для обоих записей имеет очень длинный 16-ричный текст, представляющий в UTF-16 уже хорошо знакомый нам PnPDeviceID, причем в формате с “\” замененным на “#”. Такие ключи прямого доступа создавались в LogicalVolumesCollection и потому становится легко и просто отфильтровывать ненужные Removables.

Все вышесказанное относится и к разделам дисков, как внутренних, так и внешних: сменили Вы один из своих внутренних дисков, но информация обо всех его разделах продолжает храниться в MountedDevices до тех пор, пока Вы не установите, к примеру, Windows с нуля, то есть не уничтожите существующий Registry, а при всех обновлениях, контент MountedDevices будет переезжать из обновляемой системы в обновленную.

Теперь о том, как по значениям поля value в записях MountedDevices различать логические тома, соответствующие Mbr-форматированным дискам и Gpt-форматированным. Предельно просто: если поле value имеет длину 24 символа, то оно представляет собой Signature, подпись родительского диска, 8 символов и смещение, 16 символов.

Вот только использовать в качестве ключа для доступа к LogicalVolumesCollection еще нельзя, предварительно надо инвертировать и поле Signature и поле Offset, поскольку по непонятной

мне до сих пор причине, эти поля хранят инвертированную по порядку следования байтов информацию. Так что выполнив инвертирование, мы можем проверить наличие ключа прямого доступа в коллекции логических томов и, если нам повезет, то присвоить букву объекту из LogicalVolumesCollection.

Но если нам не повезло с записью MountedDevices из подмножества \DosDevices\x:, то, возможно, нам повезет с ключом, составленным из VolumeGUID.F1 + смещение из поля value. И вот если нам повезет с ключом VolumeGUID.F1 + смещение, то нам нужно немедленно создать в коллекции логических томов дополнительные ключи прямого доступа, составленные из подписи диска + смещение.

И наконец о логических томах на Gpt-форматированных дисках. Поле value для таких логических томов имеет длину 48 байт, из которых первые 16 содержат фиксированную информацию, представленную 8-ю utf-16 символами и 32 16-ричных символа в виде строки, представляющей QWORD, сформированный на основе информации из полей VolumeGUID. Вот алгоритм обратной конвертации:

```
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 }
```

В отличии от логических томов для Mbr-форматированных дисков, для каждого из томов на Gpt-форматированном диске в MountedDevices создается только одна запись. Причем тома Gpt-форматированных дисков, не имеющие буквы, в MountedDevices не представлены. Поэтому после вычисления VolumeGUID по описанному выше алгоритму, следует выделить поле F1 и использовать его в качестве ключа прямого доступа к коллекции логических томов.

Рисунок 5.

Рисунок 5 хорошо демонстрирует логику исполнения шага 4: сначала при обработке подмножества MountedDevices, содержащего VolumeGUID, создаются новые ключи прямого доступа на основе Signature+Offset, а затем при обработке подмножества DosDevices, эти новые ключи позволяют “SetMountNameForMountPoint”.

На этом краткое описание алгоритма завершено. На GitHub еще месяц назад было выложено существенно более подробное описание с массой ссылок на VM/CMS, в которой впервые появились буквы, на *NIX системы, которые прекрасно обходятся и без букв, но лучше там же читать текст EnumDevicesAndVolumes.wsf, поскольку упомянутый подробный документ составлен именно из комментариев, извлеченных из текста кода, приправленных уже устарелыми, не очень внятными комментариями на en-RU, в которых к середине фразы только человек, хорошо знакомый с романами Толстого, еще не забудет о том, что же говорилось в начале фразы. Все же стиль мышления изменить намного труднее, чем стиль форматирования разделов на диске.

При чтении комментариев не торопитесь использовать идиоматические выражения при чтении фрагментов, в которых одной единственной строке кода предшествуют 10 строк пояснительных комментариев: это означает, что выполняется некий очень важный для алгоритма код. Так же как полезны и разметочные теги в конце строк, обозначающие имя класса, которому принадлежит данный фрагмент. Поскольку три класса DevicesCollection, StorageVolumesCollection и LogicalVolumesCollection очень похожи как по логике выполнения, так и по составу кода, то такие теги позволяют легко ориентироваться в достаточно длинном тексте. Общие для всех классов методы и функции, естественно вынесены наверх, но вот к примеру, парсинг текстовых строк описания устройств разнится от класса к классу и присутствует на всех четырех шагах алгоритма.

Об инструментально классе StdRegWrapperClass, используемом всеми классами и функциями для извлечения данных из Registry.

Это сейчас я его так именую, “инструментальный”, а вот когда писал, то очень гордился, а вот код для извлечения данных из Registry, презрительно именовал TestCase. Но вот сейчас мне выгодней именовать его инструментальным по той простой причине, что я элементарно не успел включить в его состав Set, Delete методы и потому название инструментальный легко оправдывает это название - НЕ ТРЕБУЮТСЯ Set / Delete методы для решения описанной выше задачи. Но я собираюсь поддерживать этот класс и непременно включу в его состав все методы из StdRegProv. Надеюсь, что даже в течение августа.

Об ошибках и недоработках в алгоритме и коде.

Есть ошибка в коде на шаге 4 в цикле обработки строк, считанных из MountedDevices: неожиданно в ходе отладки обнаружилось, что строка с VolumeGUID и, соответственно без буквы, считывается позже, чем обработаны все записи DosDevices. Не стал терять время, ошибка непонятно где прячется, вполне возможно, что в инструментальном классе. Потому лишь увеличил сложность алгоритма на O(N), где N - число строк в MountedDevices, разбрасывая считанные строки по двум массивам, первый из которых хранит лишь

VolumeGUID строки, а второй лишь DosDevices строки. Затем в еще одном цикле длины N последовательно считываются и обрабатываются строки сначала из первого массива, а затем из второго. Непременно разберусь и исправлю.

В инструментальном классе метод интерпретации даты в формате REG_QWORD работает неправильно. Потратил на отладку несколько часов, но разобраться не смог. Был бы очень признателен, если бы кто-то прислал мне точно известную интерпретацию хоть одной из дат в Registry, хранимых в формате REG_QWORD.
Эту “больное место”, очень хочется зафиксить.

Совершенно непонятно для меня, что же хранится в первых восьми байтах свойства Identifier, HKLM\HARDWARE\DESCRIPTION\System\MultifunctionAdapter\0\DiskController\0\DiskPeripheral, "Identifier"="9cb5ff90-00000000-A". Никаких собственных идей на это счет нет, поиски в Registry дают единственный результат. Если кто-то сможет подсказать, то буду очень признателен.

Не до конца разобрался со структурой блоба Data из MountPoints2\CPC\Volume. Но особой причины тратить на это время, не вижу. Да, хорошо было бы извлечь оттуда размеры томов, но вот не нашел подходящего поля среди тех, что еще не идентифицировал. Но все же пару часов потрачу.

Комментированный список полезных ссылок по теме публикации.

http://www.script-coding.com/WMI.html - отличная статья о WMI coding, отличный российский сайт, посвященный Scripting, содержит массу аналогичный статей.

Именно благодаря публикациям на этом сайте я стал использовать структурированный WSF-код вместо “простынь” на чистом JS. Замечательный сайт.

https://www.microsoft.com/en-us/download/details.aspx?id=12028 - загрузка лучшего в моем представлении “учебника” по WMI-методам, ScriptomaticV2.hta. MUST HAVE для программистов на JS, VBS, Perl с использованием WMI-методов. Выводит исходник кода для извлечения результатов работы выбранного из громаднейшего дерева WMI-классов на одном из выбранных языков.

Цикл из нескольких статей на портале http://www.forensicmag.com

http://www.forensicmag.com/articles/2011/12/windows-7-registry-forensics-part-2
http://www.forensicmag.com/articles/2012/02/windows-7-registry-forensics-part-3
http://www.forensicmag.com/articles/2012/06/retrieving-digital-evidence-methods-techniques-and-issues-part-3
http://www.forensicmag.com/articles/2012/04/windows-7-registry-forensics-part-4

В принципе интересный портал, потратил несколько часов, читая публикации на этом портале. Но сразу отмечу, что к моему счастью я слишком поздно его обнаружил, когда уже собственными “глазенками” успел разобраться в том, где именно в Registry хранятся

перечисления актуальной информации, а где наличествуют “мусоросборки”, хранящие информацию о том, что хоть однажды побывало в системе. Так вот, автор приведенных выше публикаций вываливает на неподготовленного читателя такое количество путей в Registry, что потребуется очень много времени на то, чтобы разобраться в большинстве мусорных куч, адресуемых указанными ссылками.

http://winintro.ru/diskmgt.ru/html/ebd2bd6e-8b1e-43b6-a2e3-483def6ad763.htm - внятная статья о составных томах (Spanned Volumes) и динамических дисках.

http://xp-7.ru/blog/2011-01-13-101 - еще одна и тоже внятная, но увидел противоречия с первой.

http://www.webtropy.com/articles/art14-2.asp?Interop=WbemScripting - хороший Reference по WbemScripting для C# and VB.NET

Для самых терпеливых, добравшихся до этого места.

```
//  JUST FOR FUN: about the role of the Case sensitivity in the Registry :)

                        +----------- Upper Case 's' in services
//                      |       +-- Upper Case 1-st character in the driver name
// Windows XP:          V       V
// SYSTEM\CurrentControlSet\Services\Cdrom
// SYSTEM\CurrentControlSet\Services\Disk
// SYSTEM\CurrentControlSet\Services\PartMgr
// SYSTEM\CurrentControlSet\Services\VolSnap

                        +------------ Low Case 's' in services
//                      |       +--- Low Case 1-st character in the driver name
// Windows 7:           V       V
// SYSTEM\CurrentControlSet\services\cdrom
// SYSTEM\CurrentControlSet\services\disk
// SYSTEM\CurrentControlSet\services\sffdisk
// SYSTEM\CurrentControlSet\services\partmgr
// SYSTEM\CurrentControlSet\services\volsnap

                        +------------- Upper Case 's' in services
//                      |       +---- Low Case 1-st character in the driver name
// Windows 8 and above: V       V
// SYSTEM\CurrentControlSet\Services\cdrom
// SYSTEM\CurrentControlSet\Services\disk
// SYSTEM\CurrentControlSet\Services\rdyboost
// SYSTEM\CurrentControlSet\Services\volmgr
// SYSTEM\CurrentControlSet\Services\volsnap
```

“**Отходы производства**”

Как часто случается, при поиске нужной информации натыкаешься на ту, которая в данный момент не нужна, но интересна и потому репу почесав, немедленно начинаешь ваять код. Вот такой пример, история обновлений моей системы, представленная кодом, написанным за пару часов.

2015.03.24 22:10:19 - 2015.03.31 15:45:14 Windows 8.1 Pro 9600
2015.03.31 16:26:15 - 2015.04.23 02:56:17 Windows 10 Pro Technical Preview 10049
2015.04.23 03:44:13 - 2015.04.29 06:36:13 Windows 10 Pro Technical Preview 10061
2015.04.29 07:25:43 - 2015.05.21 11:32:07 Windows 10 Pro Insider Preview 10074
2015.05.21 14:21:40 - 2015.05.28 21:13:37 Windows 10 Pro Insider Preview 10122
2015.05.28 22:04:16 - 2015.05.30 19:11:22 Windows 10 Pro Insider Preview 10125
2015.05.30 20:16:43 - 2015.07.01 11:17:37 Windows 10 Pro Insider Preview 10130
2015.07.01 12:11:53 - 2015.07.01 13:25:29 Windows 10 Pro Insider Preview 10158
2015.07.01 23:22:08 - 2015.07.03 22:25:40 Windows 10 Pro Insider Preview 10159
2015.07.03 23:21:16 - 2015.07.09 24:24:29 Windows 10 Pro Insider Preview 10162
2015.07.10 01:21:50 - Windows 10 Pro Insider Preview 10166

И в заключение ссылка на github:

https://github.com/sysprg-46/EnumerateDevicesAndVolumes/tree/master