API LIBVIRT

New features for Hyper-V driver

Abstract

List of new functionalities bring to API libvirt by Simon Rastello
and Adrien Kantcheff during their internship

Adrien Kantcheff

Contents

1. Introduction

The OpenCloudware project aims to propose a multi-IaaS compliant PaaS platform. Sirocco VMM is part of the IaaS range. Sirocco contains several drivers to interact with multiple hypervisors. A solution to interact with Microsoft hypervisor Hyper-V is through the library *libvirt*. Actually libvirt Hyper-V driver is few developed and no contribution added since September 2011. Our work was to bring new functionalities in order to do basic actions on virtual machines and also to encourage other developers to continue the development of the driver.

1.1. Intern functions

Hyper-V libvirt driver uses WMI instructions in order to consult and modify objects from CIM standard model, implemented by Windows and contented hypervisor information. The driver communicates with WMI providers through OpenWSman API, an implementation of WS-Management (Web Services for Management) specifications. WS-Management is based on SOAP transport protocol which manages (creation, enumeration, modification, destruction) distant resources. Consequently, when the libvirt driver needs to join WMI provider, it uses OpenWSman API which generates a SOAP formatted request. This request is sent to the hypervisor and then treated by WinRS service.

The libvirt driver already disposes functions to enumerate and get WMI classes by sending WQL requests to WMI provider and also to call WMI class methods. For that last kind of communication, a method requires input parameters. There are two kinds of argument: basic (integer, string…) and complex (objects, end point references, embedded instances). Actually, just the first argument passing mode is available with libvirt driver. But the second one is very useful because there is many WMI methods with complex types parameters, and for the moment it restraints developers to call WMI methods only with basic types parameters. So in order to expand WMI methods calls, we have implemented the second argument passing mode.

1.2. New libvirt functionalities

The new functionalities brought to libvirt are especially basic actions to manage virtual machines such as:

- Shutdown a VM
- Create, destroy a VM
- Update VM resources (vCPUs, memory, storage, network)
- Get VM resources information

We also add stubs for virt-manager utilization.

2. Intern functions

File name	Function name

`hyperv_wmi.c`	`hypervCreateXmlStruct()`
	`hypervGetPropType()`
	`hypervAddEmbeddedParam()`
	`hypervAddSimpleParam()`
	`hypervAddEprParam()`
	`hypervInvokeMethodXml()`
	`hypervInvokeMethod()`

3. List of new functionalities

3.1. VM core functionalities

3.1.1. Create a VM

File name	Function name
hyperv_driver.c	hypervDomainDefineXML()

This function defines a new VM but doesn't start it. VM properties are passed to the function through an XML description. VM properties are name, vCPU number, memory capacity, storage and network.

File name	Function name
hyperv_driver.c	hypervDomainCreateXML()

This function is similar as the one above but it launches the new VM. Normally libvirt distinguishes between two different types of guest domains (virtual machines): transient and persistent.

- Transient domains only exist until the domain is shutdown or when the host server is restarted.
- Persistent domains last indefinitely.

But with Hyper-V, VMs are only persistent domains. So the difference between the two functions depends on starting or not the VM.

An example of XML description:

```
<domain type='hyperv'>
  <name>test</name>
  <uuid>43b3f39e-2b2a-5f97-0a36-82937a779ee1</uuid>
  <memory>1048576</memory>
  <currentMemory>1048576</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64'>hvm</type>
  </os>
  <devices>
    <disk type='file'>
      <source file='D:\\Hyper-V\\Disks\\Disk.vhd'/>
      <target dev='hda'/>
    </disk>
    <interface type='network'>
      <source network='Internal'/>
    </interface>
  </devices>
</domain>
```

For further information about the function, see libvirt documentation here.

3.1.2. Destroy a VM

File name	Function name
hyperv_driver.c	hypervDomainUndefine() hypervDomainUndefineFlags()

It removes a previously defined VM from the management scope of the host system. Any associated resource definitions will also be removed. The VM is previously powered off if it is not in a saved state.

For further information about those functions, see libvirt documentation here.

3.1.3. Shutdown a VM

File name	Function name
hyperv_driver.c	hypervDomainShutdown() hypervDomainShutdownFlags()

It turns the VM off.

For further information about those functions, see libvirt documentation here.

3.1.4. Attach a virtual device

File name	Function name
hyperv_driver.c	hypervDomainAttachDevice() hypervDomainAttachDeviceFlags()

It creates a virtual device attachment to backend. Actually virtual devices supported are hard drive and virtual network. As the function to create a VM, properties are passed in the function by an XML description.

- An example of XML description for a virtual network:

```
<interface type='network'>
  <source network='Internal'/>
</interface>
```

- An example of XML description for a virtual hard drive:

```
<disk type='file'>
  <source file='D:\\Hyper-V\\Disks\\Disk.vhd'/>
  <target dev='hda'/>
</disk>
```

In Hyper-V disk drives (hard drive, CD/DVC, floppy) are attached to a VM using an IDE Controller. There are 2 IDE Controllers and each of them can host 2 disk drives. The emplacement is important because it is used by the VM for boot order: the device attached to IDE Controller 0 and location 0 is the primary one. The attachment emplacement is specified in the XML description with the tag “target” using the property “dev” which has 4 values:

- hda: IDE Controller 0, location 0
- hdb: IDE Controller 0, location 1
- hdc: IDE Controller 1, location 0
- hdd: IDE Controller 1, location 1

Warning: an error occurred when the user tries to add a drive when another one is attach to a controller with a higher priority. For example if the IDE Controller 0:1 possessed a drive, the attachment to the other controllers will fail except for the IDE Controller 0:0.

For further information about those functions, see libvirt documentation here.

3.1.5. Update the number of vCPUs

File name	Function name
`hyperv_driver.c`	`hypervDomainSetVcpus()` `hypervDomainSetVcpusFlags()`

It changes the number of virtual CPUs used by the VM.

For further information about those functions, see libvirt documentation here.

3.1.6. Get the number of vCPUs

File name	Function name
`hyperv_driver.c`	`hypervDomainGetVcpusFlags()`

It provides the number of virtual CPUs used by the VM.

For further information about those functions, see libvirt documentation here.

3.1.7. Get information about vCPUs

File name	Function name
`hyperv_driver.c`	`hypervDomainGetVcpus()`

It extracts information about virtual CPUs of the VM which is:

- Virtual CPU number
- Virtual CPU state: offline or running
- CPU time used, in nanoseconds
- Real CPU number

For further information about those functions, see libvirt documentation here.

3.1.8. Get the maximum of vCPUs

File name	Function name
`hyperv_driver.c`	`hypervDomainGetMaxVcpus()` `hypervConnectGetMaxVcpus()`

It provides the maximum number of virtual CPUs supported for a guest VM.

For further information about those functions, see libvirt documentation here.

3.1.9. Update memory capacity

File name	Function name
`hyperv_driver.c`	`hypervDomainSetMemory()` `hypervDomainSetMemoryFlags()`

It changes the target amount of physical memory allocated to a VM. The memory size must be in kibibytes.

For further information about those functions, see libvirt documentation here.

3.1.10. Update maximum memory capacity

File name	Function name
hyperv_driver.c	hypervDomainSetMaxMemory()

It changes the maximum amount of physical memory allocated to a VM. The memory size must be in kibibytes.

For further information about those functions, see libvirt documentation here.

3.1.11. Get host free memory

File name	Function name
hyperv_driver.c	hypervNodeGetFreeMemory()

It provides the free memory available on the host machine in bytes.

For further information about those functions, see libvirt documentation here.

3.1.12. Update automatic startup action

File name	Function name
hyperv_driver.c	hypervDomainSetAutostart()

It configures the VM to be automatically started when the host machine starts. With Hyper-V three automatic start actions are available for a VM: do not start, start if previously running or always start.

For further information about those functions, see libvirt documentation here.

3.1.13. Get automatic startup action

File name	Function name
hyperv_driver.c	hypervDomainGetAutostart()

It provides which automatic action is affected to the VM when the host machine starts. There are three actions: do not start, start if previously running or always start.

For further information about those functions, see libvirt documentation here.

3.1.14. Get Hyper-V version level

File name	Function name
hyperv_driver.c	hypervConnectGetVersion()

It provides the version level of the hypervisor Hyper-V running.

For further information about those functions, see libvirt documentation here.

3.1.15. Get Hyper-V capabilities

File name	Function name
hyperv_driver.c	hypervConnectGetCapabilities()

It provides capabilities of the hypervisor / driver. The capabilities are returned through an XML description.

For further information about those functions, see libvirt documentation here.

3.1.16. Get scheduler type

File name	Function name
hyperv_driver.c	hypervDomainGetSchedulerType()

It provides the scheduler type and the number of scheduler parameter. With Hyper-V the scheduler type is “allocation” and there are three scheduler parameters (limit, reservation and weight).

For further information about those functions, see libvirt documentation here.

3.1.17. Get scheduler parameters

File name	Function name
hyperv_driver.c	hypervDomainGetSchedulerParameters() hypervDomainGetSchedulerParametersFlags()

It provides the three scheduler parameters for Hyper-V which are limit, reservation and weight.

For further information about those functions, see libvirt documentation here.

3.2. VM network functionalities

3.2.1. Get the number of active networks

File name	Function name
hyperv_network_driver.c	hypervConnectNumOfNetworks()

It provides the number of active networks.

For further information about those functions, see libvirt documentation here.

3.2.2. Get active networks names

File name	Function name
hyperv_network_driver.c	hypervConnectListNetworks()

It provides the names of active networks.

For further information about those functions, see libvirt documentation here.

3.2.3. Get the number of inactive networks

File name	Function name
`hyperv_network_driver.c`	`hypervConnectNumOfDefinedNetworks()`

It provides the number of active networks.

For further information about those functions, see libvirt documentation here.

3.2.4. Get inactive networks names

File name	Function name
`hyperv_network_driver.c`	`hypervConnectListDefinedNetworks()`

It provides the names of active networks.

For further information about those functions, see libvirt documentation here.

3.2.5. Get a network by its name

File name	Function name
`hyperv_network_driver.c`	`hypervNetworkLookupByName()`

It returns the characteristics of a network given by its name.

For further information about those functions, see libvirt documentation here.

3.2.6. Get network XML description

File name	Function name
`hyperv_network_driver.c`	`hypervNetworkGetXMLDesc()`

It provides an XML description of the network.

Warning: this function is incomplete. The XML contains only the name and the UUID of the network.

For further information about those functions, see libvirt documentation here.

3.3. VM storage functionalities

3.3.1. Get the number of active storage pools

File name	Function name
`hyperv_storage_driver.c`	`hypervConnectNumOfStoragePools()`

It provides the number of active storage pools.

Warning: this function is a stub for virt-manager. It returns 1.

For further information about those functions, see libvirt documentation here.

3.3.2. Get active storage pools names

File name	Function name
`hyperv_storage_driver.c`	`hypervConnectListStoragePools()`

It provides the names of active storage pools.

Warning: this function is a stub for virt-manager. It returns one name of storage pool which is "Primordial".

For further information about those functions, see libvirt documentation here.

3.3.3. Get the number of inactive storage pools

File name	Function name
`hyperv_storage_driver.c`	`hypervConnectNumOfDefinedStoragePools()`

It provides the number of inactive storage pools.

Warning: this function is a stub for virt-manager. It returns 0.

For further information about those functions, see libvirt documentation here.

3.3.4. Get a storage pools by its name

File name	Function name
`hyperv_storage_driver.c`	`hypervStoragePoolLookupByName()`

It returns the characteristics of a storage pool given by its name.

Warning: this function is a stub for Sirocco Connection. It returns NULL.

For further information about those functions, see libvirt documentation here.

3.3.5. Get a storage volume by its path

File name	Function name
hyperv_storage_driver.c	hypervStorageVolLookupByPath()

It returns the characteristics of a storage volume given by its path.

Warning: this function is a stub for virt-manager. It returns NULL.

For further information about those functions, see libvirt documentation here.

3.4. SOAP XML generation

File name	Function name
hyperv_wmi.c	hypervCreateXmlStruct() hypervGetPropType() hypervAddEmbeddedParam() hypervAddSimpleParam() hypervAddEprParam() hypervInvokeMethodXml() hypervInvokeMethod()

Those functions are used to generate the SOAP XML needed to call WMI functions located on the hypervisor.

4. List of new libvirt Hyper-V functions

4.1. VM core functionalities

The following functions are written in the file *hyperv_driver.c*.

4.1.1. hypervDomainDefineXML

```
virDomainPtr hypervDomainDefineXML(virConnectPtr conn, const char * xml)
```

Define a new VM but doesn't start it. VM properties are passed to the function through an XML description. VM properties are name, vCPU number, memory capacity, storage and network.

conn: pointer to the hypervisor connection
xml: the XML description for the domain, preferably in UTF-8
Returns: NULL in case of error, a pointer to the domain otherwise

WMI method used	WMI class	
DefineVirtualSystem	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_VirtualSystemGlobalSettingData	ElementName	Name of the VM

4.1.2. hypervDomainCreateXML

```
virDomainPtr hypervDomainCreateXML(virConnectPtr conn,
                                   const char * xmlDesc,
                                   unsigned int flags)
```

Create a new VM and start it. VM properties are passed to the function through an XML description. VM properties are name, vCPU number, memory capacity, storage and network.
If the VIR_DOMAIN_START_PAUSED flag is set, the VM will be started and then suspended.
If the VIR_DOMAIN_START_AUTODESTROY flag is set, the VM will be started and then automatically destroyed.

conn: pointer to the hypervisor connection
xmlDesc: string containing an XML description of the domain
flags: bitwise-OR of supported virDomainCreateFlags

- VIR_DOMAIN_NONE = 0 : Default behavior
- VIR_DOMAIN_START_PAUSED = 1 : Launch guest in paused state
- VIR_DOMAIN_START_AUTODESTROY = 2 : Automatically kill guest when virConnectPtr is closed

Returns: a new domain object or NULL in case of failure

WMI method used	WMI class	
DefineVirtualSystem	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_VirtualSystemGlobalSettingData	ElementName	Name of the VM

4.1.3. hypervDomainUndefine

```
int hypervDomainUndefine(virDomainPtr domain)
```

Remove a previously defined VM from the management scope of the host system. Any associated resource definitions will also be removed. The VM is previously powered off if it is not in a saved state.

domain: pointer to a defined domain
Returns: 0 in case of success, -1 in case of error

WMI method used	WMI class
DestroyVirtualSystem	Msvm_VirtualSystemManagementService

4.1.4. hypervDomainUndefineFlags

```
int hypervDomainUndefineFlags(virDomainPtr domain, unsigned int flags)
```

Similar as *hypervDomainUndefine()* because the use of flags is not implemented.

domain: pointer to a defined domain
flags: bitwise-OR of supported virDomainUndefineFlagsValues but never used
Returns: 0 in case of success, -1 in case of error

WMI method used	WMI class
DestroyVirtualSystem	Msvm_VirtualSystemManagementService

4.1.5. hypervDomainShutdown

```
int hypervDomainShutdown(virDomainPtr domain)
```

Shutdown the VM passed through the pointer in parameter.

domain: a domain object
Returns: 0 in case of success, -1 in case of failure

WMI method used	WMI class
RequestStateChange	Msvm_ComputerSystem

4.1.6. hypervDomainShutdownFlags

```
int hypervDomainShutdownFlags(virDomainPtr domain, unsigned int flags)
```

Similar as *hypervDomainShutdown()* because the use of flags is not implemented.

domain: a domain object
flags: bitwise-OR of virDomainShutdownFlagValues but never used
Returns: 0 in case of success, -1 in case of failure

WMI method used	WMI class
RequestStateChange	Msvm_ComputerSystem

4.1.7. hypervDomainGetVcpus

```
int hypervDomainGetVcpus(virDomainPtr domain,
                         virVcpuInfoPtr info,
                         int maxinfo,
                         unsigned char * cpumaps,
                         int maplen)
```

Extract information about virtual CPUs of the VM and store it in *info* array. Information are virtual CPU number, virtual CPU state (offline or running), CPU time used (in nanoseconds), real CPU number. Actually nothing is stored in *cpumaps* string.

domain: pointer to domain object, or NULL for Domain0
info: pointer to an array of virVcpuInfo structures (OUT)
maxinfo: number of structures in info array
cpumaps: pointer to a bit map of real CPUs for all vcpus of this domain (in 8-bit bytes) (OUT) but no cpumap information is returned
maplen: number of bytes in one cpumap, from 1 up to size of CPU

Returns: the number of info filled in case of success, -1 in case of failure

WMI class queried	Property	Comment
Win32_PerfRawData_HvStats_ HyperVHypervisorVirtualProcessor	Name	Name of each vCPU
	PercentTotalRunTime	vCPU run time

4.1.8. hypervDomainGetVcpusFlags

```
int hypervDomainGetVcpusFlags(virDomainPtr domain, unsigned int flags)
```

Return the number of virtual CPUs used by the domain.
If the VIR_DOMAIN_VCPU_MAXIMUM flag is set, then the maximum virtual CPU limit is queried.
If the VIR_DOMAIN_VCPU_LIVE is set, this will query a running domain (which will fail if domain is not active).

domain: pointer to domain object, or NULL for Domain0
flags: bitwise-OR of virDomainVcpuFlags

- VIR_DOMAIN_VCPU_CURRENT = 0 : Affect current domain state
- VIR_DOMAIN_VCPU_LIVE = 1 : Affect running domain state
- VIR_DOMAIN_VCPU_MAXIMUM = 4 : Max rather than current count

Returns: the number of vCPUs in case of success, -1 in case of failure

WMI class queried	Property	Comment
Msvm_ProcessorSettingData	VirtualQuantity	Number of vCPUs in the VM

4.1.9. hypervConnectGetMaxVcpus

```
int hypervConnectGetMaxVcpus(virConnectPtr conn,
                             const char * type)
```

Return the maximum number of virtual CPUs supported for a guest VM. The *type* parameter here is never used.

conn: pointer to the hypervisor connection
type: value of the 'type' attribute in the <domain> element but never used
Returns: the maximum of virtual CPU or -1 in case of error

WMI class queried	Property	Comment
Msvm_ProcessorSettingData	InstanceID	The maximum number of vCPUs is defined by an object with an ID like “Microsoft:Definition%Maximum”
	SocketCount	Number of processor sockets in the VM
	ProcessorsPerSocket	Number of processors, or cores, configured for each socket in the VM

4.1.10. hypervDomainGetMaxVcpus

```
int hypervDomainGetMaxVcpus(virDomainPtr domain)
```

Return the maximum number of virtual CPUs supported for the guest VM. If the guest is inactive, this is the same as *hypervConnectGetMaxVcpus()*. If the guest is running, this is same as *hypervDomainGetVcpusFlags()* with the flags VIR_DOMAIN_VCPU_LIVE and VIR_DOMAIN_VCPU_MAXIMUM set.

domain: pointer to domain object
Returns: the maximum of virtual CPU or -1 in case of error

WMI class queried	Property	Comment
Msvm_ProcessorSettingData	InstanceID	The maximum number of vCPUs is defined by an object with an ID like "Microsoft:Definition%Maximum"
	SocketCount	Number of processor sockets in the VM
	ProcessorsPerSocket	Number of processors, or cores, configured for each socket in the VM

4.1.11. hypervDomainSetVcpus

```
int hypervDomainSetVcpus(virDomainPtr domain, unsigned int nvcpus)
```

Change the number of virtual CPUs used by the VM.

domain: pointer to domain object
nvcpus: the new number of virtual CPUs for this domain
Returns: 0 in case of success, -1 in case of failure

WMI method used	WMI class	
ModifyVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_ProcessorSettingData	VirtualQuantity	Number of vCPUs in the VM

4.1.12. hypervDomainSetVcpusFlags

```
int hypervDomainSetVcpusFlags(virDomainPtr domain,
                              unsigned int nvcpus,
                              unsigned int flags)
```

Similar as *hypervDomainSetVcpus()* because the use of flags is not implemented.

domain: pointer to domain object
nvcpus: the new number of virtual CPUs for this domain, must be at least 1
flags: bitwise-OR of virDomainVcpuFlags but never used
Returns: 0 in case of success, -1 in case of failure

WMI method used	WMI class	
ModifyVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_ProcessorSettingData	VirtualQuantity	Number of vCPUs in the VM

4.1.13. hypervDomainSetMemory

```
int hypervDomainSetMemory(virDomainPtr domain, unsigned long memory)
```

Change the amount of physical memory allocated to the VM.

domain: pointer to domain object
memory: the memory size in kibibytes (blocks of 1024 bytes)
Returns: 0 in case of success and -1 in case of failure

WMI method used	WMI class	
ModifyVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_MemorySettingData	VirtualQuantity	Total amount of RAM in the VM

4.1.14. hypervDomainSetMemoryFlags

```
int hypervDomainSetMemoryFlags(virDomainPtr domain, unsigned long memory)
```

Similar as *hypervDomainSetMemory()* because the use of flags is not implemented.

domain: pointer to domain object
memory: the memory size in kibibytes (blocks of 1024 bytes)
Returns: 0 in case of success and -1 in case of failure

WMI method used	WMI class	
ModifyVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_MemorySettingData	VirtualQuantity	Total amount of RAM in the VM

4.1.15. hypervDomainSetMaxMemory

```
int hypervDomainSetMaxMemory(virDomainPtr domain,
                             unsigned long memory,
                             unsigned int flags)
```

Change the maximum amount of physical memory allocated to the VM.

domain: pointer to domain object
memory: the memory size in kibibytes (blocks of 1024 bytes)
Returns: 0 in case of success and -1 in case of failure

WMI method used	WMI class	
ModifyVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_MemorySettingData	Limit	Maximum amount of RAM available

4.1.16. hypervNodeGetFreeMemory

```
unsigned long long hypervNodeGetFreeMemory(virConnectPtr conn)
```

Return the free memory available on the guest machine, in bytes.

conn: pointer to the hypervisor connection
Returns: the available free memory in bytes or 0 in case of error

WMI class queried	Property	Comment
Win32_OperatingSystem	FreePhysicalMemory	Number, in kilobytes, of physical memory currently available

4.1.17. hypervDomainAttachDevice

```
int hypervDomainAttachDevice(virDomainPtr domain, const char * xml)
```

Attach a virtual device to the VM. Supported devices are only hard drives and virtual switches.

domain: pointer to domain object
xml: pointer to XML description of one device
Returns: 0 in case of success, -1 in case of failure

- **Hard drive device**

WMI method used	WMI class	
AddVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_ResourceAllocationSettingData (default disk drive)	Parent	Parent of the resource (an object Msvm_ResourceAllocationSettingData representing IDE Controller 0 or 1)
	Address	Address of the resource (0 or 1)
	ResourceType	= 22 (Disk)
	ResourceSubType	PATH string of an object Msvm_ResourceAllocationSettingData representing the default settings of a "Microsoft Synthetic Disk Drive"

WMI method used	WMI class	
AddVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**

Msvm_ResourceAllocationSettingData (virtual hard drive)	Parent	Parent of the resource (an object Msvm_ResourceAllocationSettingData representing the default disk drive previously added)
	Connection	Source of the new disk drive
	ResourceType	= 21 (Storage Extent)
	ResourceSubType	PATH string of an object Msvm_ResourceAllocationSettingData representing the default settings of a "Microsoft Virtual Hard Drive"

- **Virtual switch device**

WMI method used	WMI class	
CreateSwitchPort	Msvm_VirtualSwitchManagementService	
WMI class affected	**Property**	**Comment**
Msvm_VirtualSwitch (virtual switch port)	Name	A generated GUID
	FriendlyName	A string equals to "Dynamic Ethernet Switch Port"
	ScopeOfResidence	A required property equals to an empty string

WMI method used	WMI class	
AddVirtualSystemResources	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_SyntheticEthernetPortSettingData (synthetic ethernet port)	Connection	A reference of the new virtual switch
	ElementName	A string equals to "Network Adapter"
	VirtualSystemIdentifiers	A generated GUID
	ResourceType	= 10 (Ethernet Adapter)
	ResourceSubType	A string equals to "Microsoft Synthetic Ethernet Port"

4.1.18. hypervDomainAttachDeviceFlags

```
int hypervDomainAttachDeviceFlags(virDomainPtr domain,
                                  const char * xml,
                                  unsigned int flags)
```

Similar as *hypervDomainAttachDevice()* because the use of flags is not implemented.

domain: pointer to domain object
xml: pointer to XML description of one device
flags: bitwise-OR of virDomainDeviceModifyFlags but never used
Returns: 0 in case of success, -1 in case of failure

4.1.19. hypervDomainGetSchedulerParameters

```
int hypervDomainGetSchedulerParameters(virDomainPtr domain,
                                       virTypedParameterPtr params,
                                       int * nparams)
```

Get all three scheduler parameters for Hyper-V which are limit, reservation and weight.

domain: pointer to domain object
params: pointer to scheduler parameter objects (return value)
nparams: pointer to number of scheduler parameter objects (input and output)
Returns: -1 in case of error, 0 in case of success

WMI class queried	Property	Comment
Msvm_ProcessorSettingData	Limit	Maximum amount of CPU resources available by a VM
	Reservation	Amount of CPU resources reserved for use by a VM
	Weight	Weight for each VM processor

4.1.20. hypervDomainGetSchedulerParametersFlags

```
int hypervDomainGetSchedulerParametersFlags(virDomainPtr domain,
                                            virTypedParameterPtr params,
                                            int * nparams,
                                            unsigned int flags)
```

Similar as *hypervDomainGetSchedulerParameters()* because the use of flags is not implemented.

domain: pointer to domain object
params: pointer to scheduler parameter objects (return value)
nparams: pointer to number of scheduler parameter objects (input and output)
flags: bitwise-OR of virDomainModificationImpact and virTypedParameterFlags but never used
Returns: -1 in case of error, 0 in case of success

WMI class queried	Property	Comment
Msvm_ProcessorSettingData	Limit	Maximum amount of CPU resources available by a VM
	Reservation	Amount of CPU resources reserved for use by a VM
	Weight	Weight for each VM processor

4.1.21. hypervDomainGetSchedulerType

```
char * hypervDomainGetSchedulerType(virDomainPtr domain, int * nparams)
```

Get the scheduler type and the number of scheduler parameters. With Hyper-V the scheduler type is "allocation" and there are three scheduler parameters (limit, reservation and weight).

domain: a domain object
nparams: pointer to number of scheduler parameters, can be NULL (return value)
Returns: 0 in case of success, -1 in case of failure

4.1.22. hypervConnectGetCapabilities

```
char * hypervConnectGetCapabilities(virConnectPtr conn)
```

Return capabilities of the hypervisor / driver.

conn: pointer to the hypervisor connection
Returns: NULL in case of error, or an XML string defining the capabilities. The client must free the returned string after use

4.1.23. hypervConnectGetVersion

```
int hypervConnectGetVersion(virConnectPtr conn, unsigned long * hvVer)
```

Get the version level of Hyper-V running.

conn: pointer to the hypervisor connection
hvVer: return value for the version of the running hypervisor (OUT)
Returns: -1 in case of error, 0 otherwise

WMI class queried	Property	Comment
CIM_DataFile	Name	The value queried is the pathname of Virtual Machine Management Service executable (vmms.exe)
	Version	Version of Virtual Machine Management Service

4.1.24. hypervDomainSetAutostart

```
int hypervDomainSetAutostart(virDomainPtr domain, int autostart)
```

Configure the VM to be automatically started when the host machine boots. With Hyper-V there are three actions to be taken when the host is started:

- None
- Restart if previously running
- Always startup

domain: a domain object
autostart: the action to be taken when the host is started:
- 0 = None
- 1 = Restart if previously running
- 2 = Always startup

Returns: -1 in case of error, 0 in case of success

WMI method used	WMI class	
ModifyVirtualSystem	Msvm_VirtualSystemManagementService	
WMI class affected	**Property**	**Comment**
Msvm_VirtualSystemGlobalSettingData	AutomaticStartupAction	Action to be taken when the host is started

4.1.25. hypervDomainGetAutostart

```
int hypervDomainGetAutostart(virDomainPtr domain, int * autostart)
```

Return a value indicating the action to be taken for the VM when the host machine boots.

domain: a domain object
autostart: the value returned:
- 0 = None
- 1 = Restart if previously running
- 2 = Always startup

Returns: -1 in case of error, 0 in case of success

WMI class queried	Property	Comment
Msvm_VirtualSystemGlobalSettingData	SystemName	VM UUID
	AutomaticStartupAction	Action to be taken when the host is started

4.2. VM network functionalities

The following functions are written in the file *hyperv_network_driver.c*.

4.2.1. hypervConnectNumOfNetworks

```
int hypervConnectNumOfNetworks(virConnectPtr conn)
```

Return the number of active networks.

conn: pointer to the hypervisor connection
Returns: the number of network found or -1 in case of error

WMI class queried	Property	Comment
Msvm_VirtualSwitch	HealthState	Equals to 5 (OK) for active networks

4.2.2. hypervConnectListNetworks

```
int hypervConnectListNetworks(virConnectPtr conn,
                              char ** const names,
                              int maxnames)
```

Collect the list of active networks, and store their names in the attribute *names*.

conn: pointer to the hypervisor connection
names: array to collect the list of names of active networks
maxnames: size of @names
Returns: the number of names provided in the array or -1 in case of error

WMI class queried	Property	Comment
Msvm_VirtualSwitch	HealthState	Equals to 5 (OK) for active networks
	ElementName	Virtual switch name

4.2.3. hypervConnectNumOfDefinedNetworks

```
int hypervConnectNumOfDefinedNetworks(virConnectPtr conn)
```

Return the number of inactive networks.

conn: pointer to the hypervisor connection
Returns: the number of network found or -1 in case of error

WMI class queried	Property	Comment
Msvm_VirtualSwitch	HealthState	Different from 5 (OK) for inactive networks

4.2.4. hypervConnectListDefinedNetworks

```
int hypervConnectListDefinedNetworks(virConnectPtr conn,
                                     char ** const names,
                                     int maxnames)
```

Collect the list of inactive networks, and store their names in the attribute *names*.

conn: pointer to the hypervisor connection
names: array to collect the list of names of active networks
maxnames: size of @names
Returns: the number of names provided in the array or -1 in case of error

WMI class queried	Property	Comment
Msvm_VirtualSwitch	HealthState	Different from 5 (OK) for active networks
	ElementName	Virtual switch name

4.2.5. hypervNetworkLookupByName

```
virNetworkPtr hypervNetworkLookupByName(virConnectPtr conn,
                                        const char * name)
```

Return a network given by its name.

conn: pointer to the hypervisor connection
name: name for the network
Returns: a new network object or NULL in case of failure

WMI class queried	Property	Comment
Msvm_VirtualSwitch	Description	Textual description of the virtual switch which is "Microsoft Virtual Switch"
	ElementName	Virtual switch name to query

4.2.6. hypervNetworkGetXMLDesc

```
char * hypervNetworkGetXMLDesc(virNetworkPtr network, unsigned int flags)
```

Return an XML description of the network.

Warning: this function is incomplete. The XML contains only the name and the UUID of the network.

network: a network object
flags: bitwise-OR of virNetworkXMLFlags
Returns: a 0 terminated UTF-8 encoded XML instance, or NULL in case of error

WMI class queried	Property	Comment
Msvm_VirtualSwitch	Name	Virtual switch UUID

4.3. VM storage functionalities

The following functions are written in the file *hyperv_storage_driver.c*.

4.3.1. hypervConnectNumOfStoragePools

```
int hypervConnectNumOfStoragePools(virConnectPtr conn)
```

Return the number of active storage pools.

Warning: this function is a stub for virt-manager. It returns 1.

conn: pointer to hypervisor connection
Returns: the number of pools found, or -1 on error

4.3.2. hypervConnectListStoragePools

```
int hypervConnectListStoragePools(virConnectPtr conn,
                                  char ** const names,
                                  int maxnames)
```

Collect the list of active storage, and store their names up to *maxnames* in the attribute *names*.

Warning: this function is a stub for virt-manager. It returns one name of storage pool which is "Primordial".

conn: pointer to hypervisor connection
names : array of char * to fill with pool names (allocated by caller)
maxnames: size of the names array
Returns: the number of pools found or -1 in case of error

4.3.3. hypervConnectNumOfDefinedStoragePools

```
int hypervConnectNumOfDefinedStoragePools(virConnectPtr conn)
```

Return the number of inactive storage pools.

Warning: this function is a stub for virt-manager. It returns 0.

conn: pointer to hypervisor connection
Returns: the number of pools found, or -1 on error

4.3.4. hypervStoragePoolLookupByName

```
virStoragePoolPtr hypervStoragePoolLookupByName(virConnectPtr conn,
                                                const char * name)
```

Return a storage pool given by its name.

Warning: this function is a stub for Sirocco Connection. It returns NULL.

conn: pointer to hypervisor connection
name: name of pool to fetch
Returns: a virStoragePoolPtr object, or NULL if no matching pool is found

4.3.5. hypervStorageVolLookupByPath

```
virStorageVolPtr hypervStorageVolLookupByPath(virConnectPtr conn,
                                              const char * path)
```

Return a storage volume given by its path.

Warning: this function is a stub for virt-manager. It returns NULL.

conn: pointer to hypervisor connection
path: locally unique path
Returns: a storage volume, or NULL if not found / error

5. Libvirt API support matrix

5.1. Hypervisor APIs

API	Version	hyperv
virConnectBaselineCPU	0.7.7	
virConnectClose	0.0.3	0.9.5
virConnectCompareCPU	0.7.5	
virConnectDomainEventDeregister	0.5.0	
virConnectDomainEventDeregisterAny	0.8.0	
virConnectDomainEventRegister	0.5.0	
virConnectDomainEventRegisterAny	0.8.0	
virConnectDomainXMLFromNative	0.6.4	
virConnectDomainXMLToNative	0.6.4	
virConnectGetCPUModelNames	1.1.3	
virConnectGetCapabilities	0.2.1	Bull
virConnectGetHostname	0.3.0	0.9.5
virConnectGetLibVersion	0.7.3	
virConnectGetMaxVcpus	0.2.1	Bull
virConnectGetSysinfo	0.8.8	
virConnectGetType	0.0.3	0.9.5
virConnectGetVersion	0.0.3	Bull
virConnectIsAlive	0.9.8	0.9.8
virConnectIsEncrypted	0.7.3	0.9.5
virConnectIsSecure	0.7.3	0.9.5
virConnectListAllDomains	0.9.13	0.10.2
virConnectListDefinedDomains	0.1.1	0.9.5
virConnectListDomains	0.0.3	0.9.5
virConnectNumOfDefinedDomains	0.1.5	0.9.5
virConnectNumOfDomains	0.0.3	0.9.5
virConnectOpen	0.0.3	0.9.5
virConnectOpenAuth	0.4.0	
virConnectOpenReadOnly	0.0.3	
virConnectSetKeepAlive	0.9.8	
virConnectSupportsFeature	0.3.2	
virDomainAbortJob	0.7.7	
virDomainAttachDevice	0.1.9	Bull
virDomainAttachDeviceFlags	0.7.7	Bull
virDomainBlockCommit	0.10.2	
virDomainBlockJobAbort	0.9.4	
virDomainBlockJobSetSpeed	0.9.4	
virDomainBlockPeek	0.4.2	
virDomainBlockPull	0.9.4	
virDomainBlockRebase	0.9.10	
virDomainBlockResize	0.9.8	
virDomainBlockStats	0.3.2	
virDomainBlockStatsFlags	0.9.5	
virDomainCoreDump	0.1.9	
virDomainCreate	0.1.1	0.9.5
virDomainCreateLinux	0.0.3	

virDomainCreateWithFiles	1.1.1	
virDomainCreateWithFlags	0.8.2	0.9.5
virDomainCreateXML	0.5.0	Bull
virDomainCreateXMLWithFiles	1.1.1	
virDomainDefineXML	0.1.1	Bull
virDomainDestroy	0.0.3	0.9.5
virDomainDestroyFlags	0.9.4	0.9.5
virDomainDetachDevice	0.1.9	
virDomainDetachDeviceFlags	0.7.7	
virDomainFSTrim	1.0.1	
virDomainGetAutostart	0.2.1	Bull
virDomainGetBlkioParameters	0.9.0	
virDomainGetBlockInfo	0.8.1	
virDomainGetBlockIoTune	0.9.8	
virDomainGetBlockJobInfo	0.9.4	
virDomainGetCPUStats	0.9.10	
virDomainGetControlInfo	0.9.3	
virDomainGetDiskErrors	0.9.10	
virDomainGetEmulatorPinInfo	0.10.0	
virDomainGetHostname	0.10.0	
virDomainGetInfo	0.0.3	0.9.5
virDomainGetInterfaceParameters	0.9.9	
virDomainGetJobInfo	0.7.7	
virDomainGetJobStats	1.0.3	
virDomainGetMaxMemory	0.0.3	
virDomainGetMaxVcpus	0.2.1	Bull
virDomainGetMemoryParameters	0.8.5	
virDomainGetMetadata	0.9.10	
virDomainGetNumaParameters	0.9.9	
virDomainGetOSType	0.0.3	0.9.5
virDomainGetSchedulerParameters	0.2.3	Bull
virDomainGetSchedulerParametersFlags	0.9.2	Bull
virDomainGetSchedulerType	0.2.3	Bull
virDomainGetSecurityLabel	0.6.1	
virDomainGetSecurityLabelList	0.10.0	
virDomainGetState	0.9.2	0.9.5
virDomainGetVcpuPinInfo	0.9.3	
virDomainGetVcpus	0.1.4	Bull
virDomainGetVcpusFlags	0.8.5	Bull
virDomainGetXMLDesc	0.0.3	0.9.5
virDomainHasCurrentSnapshot	0.8.0	
virDomainHasManagedSaveImage	0.8.0	0.9.5
virDomainInjectNMI	0.9.2	
virDomainInterfaceStats	0.3.2	
virDomainIsActive	0.7.3	0.9.5
virDomainIsPersistent	0.7.3	0.9.5
virDomainIsUpdated	0.8.6	0.9.5
virDomainListAllSnapshots	0.9.13	
virDomainLookupByID	0.0.3	0.9.5
virDomainLookupByName	0.0.3	0.9.5

virDomainLookupByUUID	0.0.5	0.9.5
virDomainLxcOpenNamespace	1.0.2	
virDomainManagedSave	0.8.0	0.9.5
virDomainManagedSaveRemove	0.8.0	0.9.5
virDomainMemoryPeek	0.4.2	
virDomainMemoryStats	0.7.5	
virDomainMigrate	0.3.2	
virDomainMigrateBegin3	0.9.2	
virDomainMigrateBegin3Params	1.1.0	
virDomainMigrateConfirm3	0.9.2	
virDomainMigrateConfirm3Params	1.1.0	
virDomainMigrateFinish	0.3.2	
virDomainMigrateFinish2	0.5.0	
virDomainMigrateFinish3	0.9.2	
virDomainMigrateFinish3Params	1.1.0	
virDomainMigrateGetCompressionCache	1.0.3	
virDomainMigrateGetMaxSpeed	0.9.5	
virDomainMigratePerform	0.3.2	
virDomainMigratePerform3	0.9.2	
virDomainMigratePerform3Params	1.1.0	
virDomainMigratePrepare	0.3.2	
virDomainMigratePrepare2	0.5.0	
virDomainMigratePrepare3	0.9.2	
virDomainMigratePrepare3Params	1.1.0	
virDomainMigratePrepareTunnel	0.7.2	
virDomainMigratePrepareTunnel3	0.9.2	
virDomainMigratePrepareTunnel3Params	1.1.0	
virDomainMigrateSetCompressionCache	1.0.3	
virDomainMigrateSetMaxDowntime	0.8.0	
virDomainMigrateSetMaxSpeed	0.9.0	
virDomainOpenChannel	1.0.2	
virDomainOpenConsole	0.8.6	
virDomainOpenGraphics	0.9.7	
virDomainPMSuspendForDuration	0.9.10	
virDomainPMWakeup	0.9.11	
virDomainPinEmulator	0.10.0	
virDomainPinVcpu	0.1.4	
virDomainPinVcpuFlags	0.9.3	
virDomainQemuAgentCommand	0.10.0	
virDomainQemuAttach	0.9.4	
virDomainQemuMonitorCommand	0.8.3	
virDomainReboot	0.1.0	
virDomainReset	0.9.7	
virDomainRestore	0.0.3	
virDomainRestoreFlags	0.9.4	
virDomainResume	0.0.3	0.9.5
virDomainRevertToSnapshot	0.8.0	
virDomainSave	0.0.3	
virDomainSaveFlags	0.9.4	
virDomainSaveImageDefineXML	0.9.4	

virDomainSaveImageGetXMLDesc	0.9.4	
virDomainScreenshot	0.9.2	
virDomainSendKey	0.9.3	
virDomainSendProcessSignal	1.0.1	
virDomainSetAutostart	0.2.1	Bull
virDomainSetBlkioParameters	0.9.0	
virDomainSetBlockIoTune	0.9.8	
virDomainSetInterfaceParameters	0.9.9	
virDomainSetMaxMemory	0.0.3	
virDomainSetMemory	0.1.1	Bull
virDomainSetMemoryFlags	0.9.0	Bull
virDomainSetMemoryParameters	0.8.5	
virDomainSetMemoryStatsPeriod	1.1.1	
virDomainSetMetadata	0.9.10	
virDomainSetNumaParameters	0.9.9	
virDomainSetSchedulerParameters	0.2.3	
virDomainSetSchedulerParametersFlags	0.9.2	
virDomainSetVcpus	0.1.4	Bull
virDomainSetVcpusFlags	0.8.5	Bull
virDomainShutdown	0.0.3	Bull
virDomainShutdownFlags	0.9.10	Bull
virDomainSnapshotCreateXML	0.8.0	
virDomainSnapshotCurrent	0.8.0	
virDomainSnapshotDelete	0.8.0	
virDomainSnapshotGetParent	0.9.7	
virDomainSnapshotGetXMLDesc	0.8.0	
virDomainSnapshotHasMetadata	0.9.13	
virDomainSnapshotIsCurrent	0.9.13	
virDomainSnapshotListAllChildren	0.9.13	
virDomainSnapshotListChildrenNames	0.9.7	
virDomainSnapshotListNames	0.8.0	
virDomainSnapshotLookupByName	0.8.0	
virDomainSnapshotNum	0.8.0	
virDomainSnapshotNumChildren	0.9.7	
virDomainSuspend	0.0.3	0.9.5
virDomainUndefine	0.1.1	Bull
virDomainUndefineFlags	0.9.4	Bull
virDomainUpdateDeviceFlags	0.8.0	
virNodeDeviceDetachFlags	1.0.5	
virNodeDeviceDettach	0.6.1	
virNodeDeviceReAttach	0.6.1	
virNodeDeviceReset	0.6.1	
virNodeGetCPUMap	1.0.0	
virNodeGetCPUStats	0.9.3	
virNodeGetCellsFreeMemory	0.3.3	
virNodeGetFreeMemory	0.3.3	Bull
virNodeGetInfo	0.1.0	0.9.5
virNodeGetMemoryParameters	0.10.2	
virNodeGetMemoryStats	0.9.3	
virNodeGetSecurityModel	0.6.1	

virNodeSetMemoryParameters	0.10.2	
virNodeSuspendForDuration	0.9.8	

5.2. Host Interface APIs

API	Version	hyperv
virConnectListAllInterfaces	0.10.2	
virConnectListDefinedInterfaces	0.7.0	
virConnectListInterfaces	0.6.4	
virConnectNumOfDefinedInterfaces	0.7.0	
virConnectNumOfInterfaces	0.6.4	
virInterfaceChangeBegin	0.9.2	
virInterfaceChangeCommit	0.9.2	
virInterfaceChangeRollback	0.9.2	
virInterfaceCreate	0.6.4	
virInterfaceDefineXML	0.6.4	
virInterfaceDestroy	0.6.4	
virInterfaceGetXMLDesc	0.6.4	
virInterfaceIsActive	0.7.3	
virInterfaceLookupByMACString	0.6.4	
virInterfaceLookupByName	0.6.4	
virInterfaceUndefine	0.6.4	

5.3. Network Filter APIs

API	Version	hyperv
virConnectListAllNWFilters	0.10.2	
virConnectListNWFilters	0.8.0	
virConnectNumOfNWFilters	0.8.0	
virNWFilterDefineXML	0.8.0	
virNWFilterGetXMLDesc	0.8.0	
virNWFilterLookupByName	0.8.0	
virNWFilterLookupByUUID	0.8.0	
virNWFilterUndefine	0.8.0	

5.4. Virtual Network APIs

API	Version	hyperv
virConnectListAllNetworks	0.10.2	
virConnectListDefinedNetworks	0.2.0	Bull
virConnectListNetworks	0.2.0	Bull
virConnectNumOfDefinedNetworks	0.2.0	Bull
virConnectNumOfNetworks	0.2.0	Bull
virNetworkCreate	0.2.0	
virNetworkCreateXML	0.2.0	
virNetworkDefineXML	0.2.0	
virNetworkDestroy	0.2.0	
virNetworkGetAutostart	0.2.1	
virNetworkGetBridgeName	0.2.0	
virNetworkGetXMLDesc	0.2.0	Bull

virNetworkIsActive	0.7.3	
virNetworkIsPersistent	0.7.3	
virNetworkLookupByName	0.2.0	Bull
virNetworkLookupByUUID	0.2.0	
virNetworkSetAutostart	0.2.1	
virNetworkUndefine	0.2.0	
virNetworkUpdate	0.10.2	

5.5. Host Device APIs

API	Version	hyperv
virConnectListAllNodeDevices	0.10.2	
virNodeDeviceCreateXML	0.6.3	
virNodeDeviceDestroy	0.6.3	
virNodeDeviceGetParent	0.5.0	
virNodeDeviceGetXMLDesc	0.5.0	
virNodeDeviceListCaps	0.5.0	
virNodeDeviceLookupByName	0.5.0	
virNodeDeviceLookupSCSIHostByWWN	1.0.3	
virNodeDeviceNumOfCaps	0.5.0	
virNodeListDevices	0.5.0	
virNodeNumOfDevices	0.5.0	

5.6. Secret APIs

API	Version	hyperv
virConnectListAllSecrets	0.10.2	
virConnectListSecrets	0.7.1	
virConnectNumOfSecrets	0.7.1	
virSecretDefineXML	0.7.1	
virSecretGetValue	0.7.1	
virSecretGetXMLDesc	0.7.1	
virSecretLookupByUUID	0.7.1	
virSecretLookupByUsage	0.7.1	
virSecretSetValue	0.7.1	
virSecretUndefine	0.7.1	

5.7. Storage Pool APIs

API	Version	hyperv
virConnectFindStoragePoolSources	0.4.5	
virConnectListAllStoragePools	0.10.2	
virConnectListDefinedStoragePools	0.4.1	
virConnectListStoragePools	0.4.1	stub
virConnectNumOfDefinedStoragePools	0.4.1	stub
virConnectNumOfStoragePools	0.4.1	stub
virStoragePoolBuild	0.4.1	
virStoragePoolCreate	0.4.1	
virStoragePoolCreateXML	0.4.1	
virStoragePoolDefineXML	0.4.1	

virStoragePoolDelete	0.4.1	
virStoragePoolDestroy	0.4.1	
virStoragePoolGetAutostart	0.4.1	
virStoragePoolGetInfo	0.4.1	
virStoragePoolGetXMLDesc	0.4.1	
virStoragePoolIsActive	0.7.3	
virStoragePoolIsPersistent	0.7.3	
virStoragePoolListAllVolumes	0.10.2	
virStoragePoolListVolumes	0.4.1	
virStoragePoolLookupByName	0.4.1	stub
virStoragePoolLookupByUUID	0.4.1	
virStoragePoolLookupByVolume	0.4.1	
virStoragePoolNumOfVolumes	0.4.1	
virStoragePoolRefresh	0.4.1	
virStoragePoolSetAutostart	0.4.1	
virStoragePoolUndefine	0.4.1	
virStorageVolCreateXML	0.4.1	
virStorageVolCreateXMLFrom	0.6.4	
virStorageVolDelete	0.4.1	
virStorageVolDownload	0.9.0	
virStorageVolGetInfo	0.4.1	
virStorageVolGetPath	0.4.1	
virStorageVolGetXMLDesc	0.4.1	
virStorageVolLookupByKey	0.4.1	
virStorageVolLookupByName	0.4.1	
virStorageVolLookupByPath	0.4.1	stub
virStorageVolResize	0.9.10	
virStorageVolUpload	0.9.0	
virStorageVolWipe	0.8.0	
virStorageVolWipePattern	0.9.10	

6. Validation of new functionalities

We validate the new functionalities using virt-manager. The results are shown below.

6.1. VM core functionalities

Functionality	Result	Comment
Create a VM	Incorrect	This functionality results by the creation of two VMs instead of a unique one. To create a VM virt-manager calls first *virDomainCreateXML()* and then *virDomainDefineXML()*. The XML parameter is the same for the both functions but virt-manager defines itself the attribute representing the UUID of the new VM which cannot be forced with Hyper-V. Indeed WMI does not allow the developer to create a VM with a predefined UUID. The hypervisor chooses itself the UUID of the new VM which is different from the one defined by virt-manager. So instead of updating the VM properties, *virDomainDefineXML()* creates a new VM because it does not recognize the previous VM created by *virDomainCreateXML()*.
Destroy a VM	Correct	This functionality works fine.

Shutdown a VM	Correct	This functionality works fine.
Attach virtual hard drive	Failed	virt-manager does not find the associated file of the virtual hard drive to attach.
Connect virtual network	Correct	This functionality works fine.
Update vCPUs number	Correct	This functionality works fine.
Get vCPUs number	None	Never used by virt-manager. It calls *virDomainGetInfo()* instead.
Get information about vCPUs	Correct	This functionality works fine.
Get maximum of vCPUs	None	Never used by virt-manager. It calls *virNodeGetInfo()* instead.
Update memory capacity	Correct	This functionality works fine.
Update maximum memory capacity	Correct	This functionality works fine.
Get host free memory	None	Never used by virt-manager.
Update automatic startup action	Correct	This functionality works fine but does not consider all three actions available because it passes a boolean to the function.
Get automatic startup action	Correct	This functionality works fine.
Get Hyper-V version level	Correct	This functionality works fine.
Get Hyper-V capabilities	Correct	This functionality works fine.
Get scheduler type	None	Never used by virt-manager.
Get scheduler parameters	None	Never used by virt-manager.

6.2. VM network functionalities

Functionality	Result	Comment
Get number of active networks	Correct	This functionality works fine.
Get active networks names	Correct	This functionality works fine.
Get number of inactive networks	Correct	This functionality works fine.
Get inactive networks names	None	Never used by virt-manager.
Get a networks by its name	Correct	This functionality works fine.
Get network XML description	Correct	This functionality works fine.

6.3. VM storage functionalities

Four out of the five new storage functionalities are stubs for virt-manager. There is no need to validate them.