# XenEnterprise Management API Draft

Version: API Revision 1.0 (Draft)
Date: 1st June 2006

Richard Sharp: `richard.sharp@xensource.com`
David Scott: `david.scott@xensource.com`

# Chapter 1

# Introduction

This document defines the XenEnterprise Management API—an API for remotely configuring and controlling virtualised guests running on a Xen-enabled cluster. The API is presented here as a set of Remote Procedure Calls. Although we adopt some terminology from object-orientated programming, this document does not specify how RPCs are exposed to a programmer by client-side language bindings. For example, the developer of a set of bindings may choose to wrap up the RPC calls described here in an object-orientated manner or a more procedural style as appropriate.
The API reference uses the terminology *classes* and *objects*. For our purposes a *class* is simply a hierarchical namespace; an *object* is an instance of a class with its fields set to specific values. Objects are persistant and exist on the server-side. Object fields are accessed from the client-side via get/set RPCs.
In each class there is a *uuid* field that assigns a *globally* unique name to each object. This *uuid* serves as an object reference on both client- and server-side, and is often included as an argument in RPC messages.
For each class we specify a list of fields along with their *types* and *qualifiers*. A qualifier is one of:

- $RO_{run}$: the field is Read Only. Furthermore, its value is automatically computed at runtime.
- $RO_{ins}$: the field must be manually set when a new object is created, but is then Read Only for the duration of the object's life.
- *RW*: the field is Read/Write.

A full list of types is given in Chapter 2. However, there are three types that require explicit mention:

- *t Ref*: signifies a reference to an object of type $t$.
- *t Set*: signifies a set containing values of type $t$.
- $(t_1, t_2)$ *Map*: signifies a mapping from values of type $t_1$ to values of type $t_2$.

Note that there are a number of cases where *Ref*s are *doubly linked*—e.g. a VM has a field called `groups` of type (*VMGroup Ref*) *Set*; this field lists the VMGroups that a particular VM is part of. Similarly, the VMGroups class has a field called `VMs` of type (*VM Ref*) *Set* that contains the VMs that are part of a particular VMGroup. These two fields are *bound together*, in the sense that adding a new VMGroup to a VM causes the VMs field of the corresponding VMGroup object to be updated automatically.
The API reference explicitly lists the fields that are bound together in this way. It also contains a diagram that shows relationships between classes. In this diagram an edge signifies the existance of a pair of fields that are bound together, using standard crows-foot notation to signify the type of relationship (e.g. one-many, many-many).

## 1.1 RPCs associated with fields

Each field, `f`, has an RPC accessor associated with it that returns `f`'s value:

- "`get_f(uuid u)`": takes a `uuid` that refers to an object and returns the value of `f`.

Each field, `f`, with attribute *RW* and whose outermost type is *Set* has the following additional RPCs associated with it:

- an "`add_to_f(uuid, v)`" RPC adds a new element v to the set[1];
- a "`remove_from_f(uuid, v)`" RPC removes element `v` from the list;

Each field, `f`, with attribute *RW* and whose outermost type is *Map* has the following additional RPCs associated with it:

- an "`add_to_f(uuid, k, v)`" RPC adds new pair `(k, v)` to the mapping stored in `f` in object `uuid`. Adding a new pair for duplicate key, `k`, overwrites any previous mapping for `k`.
- a "`remove_from_f(uuid, k)`" RPC removes the pair with key `k` from the mapping stored in `f` in object `uuid`.

Each field whose outermost type is neither *Set* nor *Map*, but whose attribute is *RW* has an RPC acessor associated with it that sets its value:

- For *RW* (*R*ead/*W*rite), a "`set_f(uuid, v)`" RPC function is also provided. This sets field `f` on object `uuid` to value `v`.

## 1.2 RPCs associated with classes

- Each class has a *constructor* RPC that takes as parameters all fields marked *RW* and $RO_{ins}$. The result of this RPC is that a new *persistent* object is created on the server-side with the specified field values.
- Each class has a "`get_all()`" RPC that returns a set of all persistent objects of that class that the system knows about. For example, `VM.get_all()` would return a list of `VM` objects that are currently installed.
- Each class has a `get_by_uuid(uuid)` RPC that returns the object of that class that has the specified `uuid`.
- Each class that has a `short_name` field has a "`get_by_short_name(name)`" RPC that returns a list of objects of that class that have the specified `name`.
- Each class has a "`to_XML()`" RPC that serialises the state of all fields as an XML string.
- Each class has a "`delete(uuid)`" RPC that explicitly deletes the persistent object specified by `uuid` from the system.

### 1.2.1 Additional RPCs

As well as the RPCs enumerated above, some classes have additional RPCs associated with them. For example, the `VM` class have RPCs for cloning, suspending, starting etc. Such additional RPCs are described explicitly in the API reference.

[1]Since sets cannot contain duplicate values this operation has no action in the case that `v` was already in the set.

## 1.3 Wire Protocol for Remote API Calls

API calls are sent over a network to a Xen-enabled host using the XML-RPC protocol. In this Section we describe how the higher-level types used in our API Reference are mapped to primitive XML-RPC types.
In our API Reference we specify the signatures of API functions in a Java-like manner. For example:

```
List<vm_id>   Host.ListAllVMs()
```

This specifies that the function with name `Host.ListAllVMs` takes no parameters and returns an list of `vm_ids`. These types are mapped onto XML-RPC types in a straight-forward manner:

- all our "`_id`" types (e.g. `vm_id` in the above example) map to XML-RPC's `String` type.
- for all our types, `t`, type `List<t>` simply maps to XML-RPC's `Array` type[2].
- our type `void` maps onto an empty XML-RPC `String`.

### 1.3.1 Return Values/Status Codes

The return value of an RPC call is an XML-RPC `Struct`.

- The first element of the struct is named `Status`; it contains a string value indicating whether the result of the call was a "`Success`" or a "`Failure`".

If `Status` was set to `Success` then the Struct contains a second element named `Value`:

- The element of the struct named `Value` contains the function's return value.

In the case where `Status` is set to `Failure` then the struct contains a second element named `ErrorDescription`:

- The element of the struct named `ErrorDescription` contains an array of string values. The first element of the array represents an error code; the remainder of the array represents error parameters relating to that code.

For example, an XML-RPC return value from the `Host.ListAllVMs` function above may look like this:

```
<struct>
   <member> <name> Status </name>
            <value> Success </value>
   </member>
   <member> <name> Value </name>
      <array>
         <data>
           <value> vm-id-1 </value>
           <value> vm-id-2 </value>
           <value> vm-id-3 </value>
         </data>
      </array>
   </member>
</struct>
```

[2]XML-RPC does not explicitly support a parameterised array type so we have no means of specifying the type of elements at this level.

## 1.4 Making XML-RPC Calls

### 1.4.1 Session Layer

The XML-RPC interface is session-based; before you can make arbitrary RPC calls you must login and initiate a session. For example:

```
session_id    Session.LoginWithUsernamePassword(string uname, string pwd)
```

Where `uname` and `password` refer to your username and password respectively, as defined by the Xen cluster administrator. The `session_id` returned by `Session.Login` is passed to subequent RPC calls as an authentication token.
A session can be terminated with the `Session.Logout` function:

```
void          Session.Logout(session_id session)
```

### 1.4.2 Synchronous and Asynchronous invocation

Each method call (apart from those on "Session" and "Task" objects) can be made either synchronously or asynchronously. A synchronous RPC call blocks until the return value is received; the return value of a synchronous RPC call is exactly as specified in Section 1.3.1.
Each of the methods specified in the API Reference is synchronous. However, although not listed explicitly in this document, each method call has an asynchronous analogue in the `Async` namespace. For example, synchronous call `VM.Install(...)` (described in Chapter 2) has an asynchronous counterpart, `Async.VM.Install(...)`, that is non-blocking.
Instead of returning its result directly, an asynchronous RPC call returns a `task-id`; this identifier is subsequently used to track the status of a running asynchronous RPC. Note that an asychronous call may fail immediately, before a `task-id` has even been created—to represent this eventuality, the returned `task-id` is wrapped in an XML-RPC struct with a `Status`, `ErrorDescription` and `Value` fields, exactly as specified in Section 1.3.1.
The `task-id` is provided in the `Value` field if `Status` is set to `Success`.
Two special RPC calls are provided to poll the status of asynchronous calls:

```
Array<task_id>  Async.Task.GetAllTasks (session_id s)
task_status     Async.Task.GetStatus   (session_id s, task_id t)
```

`Async.Task.GetAllTasks` returns a list of the currently executing asynchronous tasks belong to the current user[3].
`Async.Task.GetStatus` returns a `task_status` result. This is an XML-RPC struct with two elements:

- The first element is named `Progress` and contains an `Integer` between 0 and 100 representing the estimated percentage of the task currently completed.
- The second element is named `Result`. If `Progress` is not 100 then `Result` contains the empty string. If `Progress` *is* set to 100, then `Result` contains the function's return result (as specified in Section 1.3.1)[4].

## 1.5 VM Lifecycle

Figure 1.1 shows the states that a VM can be in and the API calls that can be used to move the VM between these states.

[3]The current user is determined by the username that was provided to `Session.Login`.
[4]Recall that this itself is a struct potentially containing status, errorcode, value fields etc.

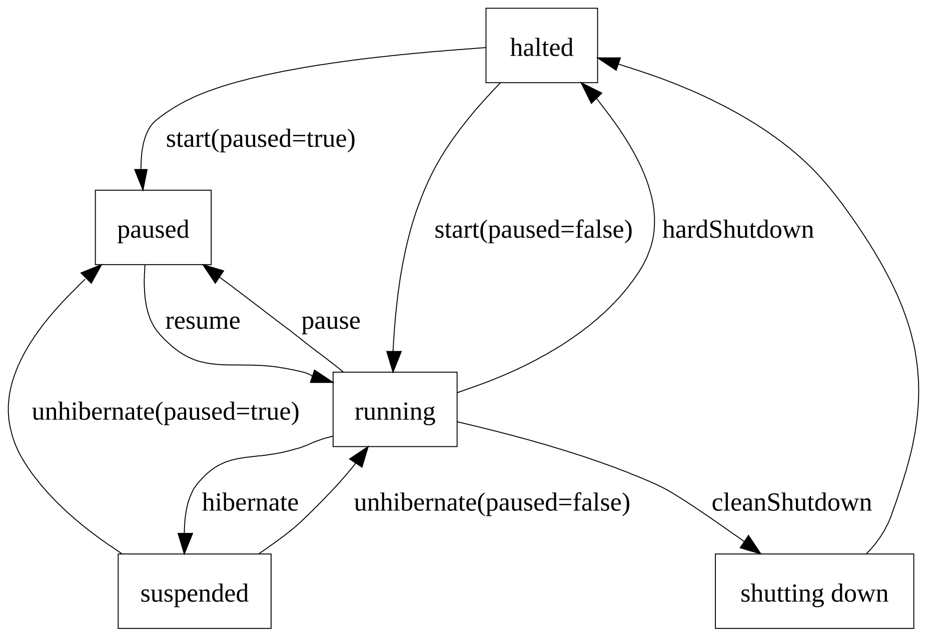


Figure 1.1: VM Lifecycle

# Chapter 2

# API Reference

This API Reference is autogenerated from datamodel specification and IDL — do not hand-edit.

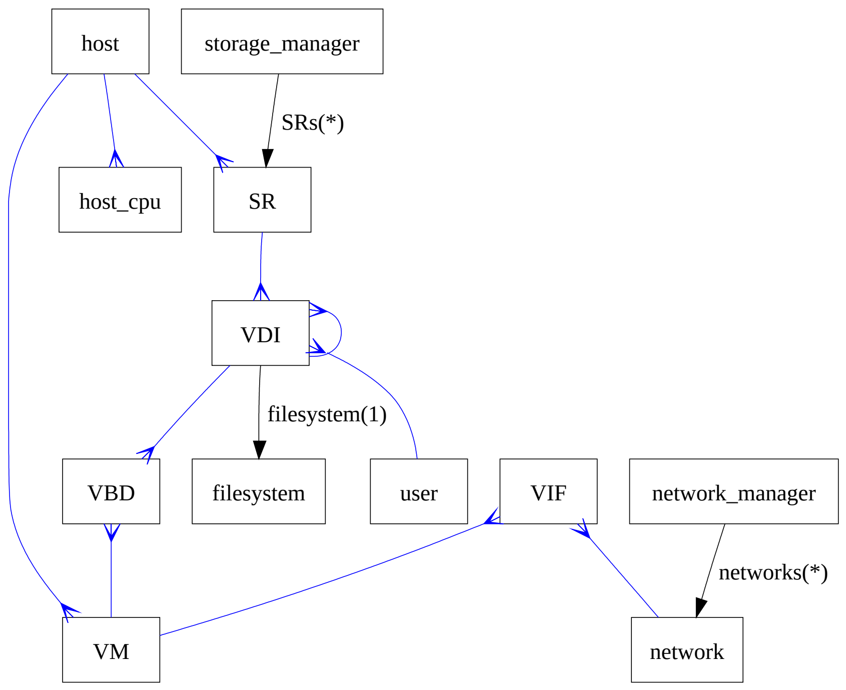
## 2.1 Classes

The following classes are defined:

| Name | Description |
|---|---|
| `VM` | a virtual machine (or 'guest') |
| `host` | a physical host |
| `host_cpu` | a physical CPU |
| `network_manager` | creates and manages virtual networks |
| `network` | a virtual network |
| `VIF` | a virtual network interface |
| `storage_manager` | creates and manages storage repositories |
| `SR` | a storage repository |
| `VDI` | a virtual disk image |
| `VBD` | a virtual block device |
| `filesystem` | an on-disk filesystem |
| `user` | a user of the system |

## 2.2 Relationships Between Classes

The relationship between classes is displayed in the following diagram. Nodes are classes while edges represent references between class instances. Black edges with arrows and labels represent the notion that an instance of one class contains a field referencing instances of the other class. The label indicates the field name and a (1) annotation indicates a single object is referenced while a (*) indicates multiple objects (e.g. through a list or map). Blue lines indicate bidirectional relationships between instances of the two classes using crows-feet notation to specify one-to-one, one-to-many or many-to-many



### 2.2.1 List of bound fields

| *object.field* | *object.field* | *relationship* |
|---|---|---|
| VDI.VBDs | VBD.VDI | many-to-one |
| VDI.parent | VDI.children | one-to-many |
| VDI.creator | user.created_VDIs | one-to-many |
| VBD.VM | VM.VBDs | one-to-many |
| VIF.VM | VM.VIFs | one-to-many |
| VIF.network | network.VIFs | one-to-many |
| SR.VDIs | VDI.SR | many-to-one |
| host.SRs | SR.local_node | many-to-one |
| host.resident_VMs | VM.running_on | many-to-one |
| host.host_CPUs | host_cpu.host | many-to-one |

## 2.3 Types

### 2.3.1 Primitives

The following primitive types are used to specify methods and fields in the API Reference:

| Type | Description |
|---|---|
| String | text strings |
| Int | 64-bit integers |
| Float | IEEE double-precision floating-point numbers |
| Bool | boolean |
| DateTime | date and timestamp |
| Ref (object name) | reference to an object of class name |

### 2.3.2 Higher order types

The following type constructors are used:

| Type | Description |
|---|---|
| List (t) | an arbitrary-length list of elements of type t |
| Map (a → b) | a table mapping values of type a to values of type b |

### 2.3.3 Enumeration types

The following enumeration types are used:

| enum power_behaviour | |
|---|---|
| destroy | destroy the VM state |
| restart | automatically restart the VM |
| preserve | leave VM running |
| rename_restart | leave VM running and restart a new one |

| enum bios_boot_option | |
|---|---|
| floppy | boot from emulated floppy |
| HD | boot from emulated HD |
| CDROM | boot from emulated CDROM |

| enum boot_type | |
|---|---|
| bios | boot an HVM guest using an emulated BIOS |
| grub | boot from inside the machine using grub |
| kernel_external | boot from an external kernel |
| kernel_internal | boot from a kernel inside the machine |

| enum cpu_feature | |
|---|---|
| FPU | Onboard FPU |
| VME | Virtual Mode Extensions |
| DE | Debugging Extensions |
| PSE | Page Size Extensions |
| TSC | Time Stamp Counter |
| MSR | Model-Specific Registers, RDMSR, WRMSR |
| PAE | Physical Address Extensions |
| MCE | Machine Check Architecture |
| CX8 | CMPXCHG8 instruction |
| APIC | Onboard APIC |
| SEP | SYSENTER/SYSEXIT |
| MTRR | Memory Type Range Registers |
| PGE | Page Global Enable |
| MCA | Machine Check Architecture |
| CMOV | CMOV instruction (FCMOVCC and FCOMI too if FPU present) |

| | |
|---|---|
| `PAT` | Page Attribute Table |
| `PSE36` | 36-bit PSEs |
| `PN` | Processor serial number |
| `CLFLSH` | Supports the CLFLUSH instruction |
| `DTES` | Debug Trace Store |
| `ACPI` | ACPI via MSR |
| `MMX` | Multimedia Extensions |
| `FXSR` | FXSAVE and FXRSTOR instructions (fast save and restore |
| `XMM` | Streaming SIMD Extensions |
| `XMM2` | Streaming SIMD Extensions-2 |
| `SELFSNOOP` | CPU self snoop |
| `HT` | Hyper-Threading |
| `ACC` | Automatic clock control |
| `IA64` | IA-64 processor |
| `SYSCALL` | SYSCALL/SYSRET |
| `MP` | MP Capable. |
| `NX` | Execute Disable |
| `MMXEXT` | AMD MMX extensions |
| `LM` | Long Mode (x86-64) |
| `3DNOWEXT` | AMD 3DNow! extensions |
| `3DNOW` | 3DNow! |
| `RECOVERY` | CPU in recovery mode |
| `LONGRUN` | Longrun power control |
| `LRTI` | LongRun table interface |
| `CXMMX` | Cyrix MMX extensions |
| `K6_MTRR` | AMD K6 nonstandard MTRRs |
| `CYRIX_ARR` | Cyrix ARRs (= MTRRs) |
| `CENTAUR_MCR` | Centaur MCRs (= MTRRs) |
| `K8` | Opteron, Athlon64 |
| `K7` | Athlon |
| `P3` | P3 |
| `P4` | P4 |
| `CONSTANT_TSC` | TSC ticks at a constant rate |
| `FXSAVE_LEAK` | FXSAVE leaks FOP/FIP/FOP |
| `XMM3` | Streaming SIMD Extensions-3 |
| `MWAIT` | Monitor/Mwait support |
| `DSCPL` | CPL Qualified Debug Store |
| `EST` | Enhanced SpeedStep |
| `TM2` | Thermal Monitor 2 |
| `CID` | Context ID |
| `CX16` | CMPXCHG16B |
| `XTPR` | Send Task Priority Messages |
| `XSTORE` | on-CPU RNG present (xstore insn) |
| `XSTORE_EN` | on-CPU RNG enabled |
| `XCRYPT` | on-CPU crypto (xcrypt insn) |
| `XCRYPT_EN` | on-CPU crypto enabled |
| `LAHF_LM` | LAHF/SAHF in long mode |
| `CMP_LEGACY` | If yes HyperThreading not valid |

```
enum vdi_type
```

| | |
|---|---|
| `system` | disks which are wiped on upgrade |
| `user` | user disks which are always preserved |
| `ephemeral` | disks which may be wiped on boot |

| `enum vbd_mode` | |
|---|---|
| `RO` | disk is mounted read-only |
| `RW` | disk is mounted read-write |

| `enum driver_type` | |
|---|---|
| `ioemu` | use hardware emulation |
| `paravirtualised` | use paravirtualised driver |

## 2.4 Class: VM

### 2.4.1 Fields for class: VM

| | | | |
|---|---|---|---|
| Name | **VM** | | |
| Description | *a virtual machine (or 'guest')* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| *RW* | `name/label` | string | a short human-readable label |
| *RW* | `name/short_description` | string | a short human-readable description |
| *RW* | `name/long_description` | string | a slightly longer human-readable description |
| *RW* | `name/detail` | string | everything you can think of, and then some more |
| *RW* | `user_version` | int | a user version number for this machine |
| *RW* | `is_a_template` | bool | true if this is a template, false a VM |
| *RW* | `running_on` | host ref | the host the VM is currently resident on |
| $RO_{ins}$ | `memory/static_max` | int | Statically-set (i.e. absolute) maximum |
| *RW* | `memory/dynamic_max` | int | Dynamic maximum |
| $RO_{run}$ | `memory/actual` | int | Guest's actual usage |
| *RW* | `memory/dynamic_min` | int | Dynamic minimum |
| $RO_{ins}$ | `memory/static_min` | int | Statically-set (i.e. absolute) mininum |
| *RW* | `VCPUs/policy` | string | the name of the VCPU scheduling policy to be applied |
| *RW* | `VCPUs/params` | string | string-encoded parameters passed to selected VCPU policy |
| $RO_{run}$ | `VCPUs/utilisation` | (int → float) Map | Utilisation for all of guest's current VCPUs |
| $RO_{ins}$ | `VCPUs/features/required` | cpu_feature Set | CPU features the guest demands the host supports |
| $RO_{ins}$ | `VCPUs/features/can_use` | cpu_feature Set | CPU features the guest can use if available |
| *RW* | `VCPUs/features/force_on` | cpu_feature Set | CPU features to hide from the guest |
| *RW* | `VCPUs/features/force_off` | cpu_feature Set | extra features to expose to the guest above the bare minimum |
| *RW* | `actions/power_off` | power_behaviour | action to take when the guest powers off |
| *RW* | `actions/reboot` | power_behaviour | action to take when the guest reboots |
| *RW* | `actions/crash` | power_behaviour | action to take if the guest crashes |
| *RW* | `actions/standby` | power_behaviour | action to take when the guest enters standby mode |
| *RW* | `coredump` | bool | set to true to generate coredump on poweroff, reboot, crash, standby |
| *RW* | `VIFs` | VIF ref Set | virtual network interfaces |
| *RW* | `VBDs` | VBD ref Set | virtual block devices |
| $RO_{ins}$ | `TPM/instance` | int | ? |
| $RO_{ins}$ | `TPM/backend` | int | ? |
| *RW* | `bios/cdrom` | string | path for emulated CDROM e.g. /dev/cdrom or /foo.iso |
| *RW* | `bios/boot` | bios_boot_option | default device to boot the guest from |

| | | | |
|---|---|---|---|
| *RW* | `platform/std_VGA` | bool | emulate standard VGA instead of cirrus logic |
| *RW* | `platform/SDL` | bool | enable the SDL console |
| *RW* | `platform/VNC` | bool | enable the VNC console |
| *RW* | `platform/serial` | string | redirect serial port to pty |
| *RW* | `platform/localtime` | bool | set RTC to local time |
| *RW* | `platform/clock_offset` | string | timeshift applied to guest's clock |
| *RW* | `platform/enable_audio` | bool | emulate audio |
| *RW* | `builder` | string | domain builder to use |
| $RO_{ins}$ | `console_port` | int | Xen port number to which console is redirected |
| *RW* | `boot_method` | boot_type | select how this machine should boot |
| *RW* | `kernel/kernel` | string | path to kernel e.g. /boot/vmlinuz |
| *RW* | `kernel/initrd` | string | path to the initrd e.g. /boot/initrd.img |
| *RW* | `kernel/args` | string | extra kernel command-line arguments |
| *RW* | `grub/cmdline` | string | grub command-line |
| $RO_{ins}$ | `PCI_bus` | string | PCI bus path for pass-through devices |
| $RO_{run}$ | `tools_version` | (string → string) Map | versions of installed paravirtualised drivers |

### 2.4.2 Additional RPCs associated with class: VM

**RPC name: clone**

**Overview:** Clones the specified VM, making a new VM. Clone automatically exploits the capabilities of the underlying storage repository in which the VM's disk images are stored (e.g. Copy on Write). (This function can only be called when the VM is in the Halted State).
**Signature:**

```
vm_id clone (session_id s, vm_id vm, string new_name)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to be cloned |
| `string` | new_name | The name of the cloned VM |

**Return Type:** `vm_id`
The ID of the newly created VM.

**RPC name: start**

**Overview:** Start the specified VM. (This function can only be called with the VM is in the Halted State).
**Signature:**

```
void start (session_id s, vm_id vm, bool start_paused)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to start |
| `bool` | start_paused | Instantiate VM in paused state if set to true. |

**Return Type:** `void`

**RPC name: pause**

**Overview:** Pause the specified VM. This can only be called when the specified VM is in the Running state.
**Signature:**

```
void pause (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to pause |

**Return Type:** `void`

**RPC name: unpause**

**Overview:** Resume the specified VM. This can only be called when the specified VM is in the Paused state.
**Signature:**

```
void unpause (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to pause |

**Return Type:** `void`

**RPC name: clean_shutdown**

**Overview:** Attempt to cleanly shutdown the specified VM. (Note: this may not be supported—e.g. if a guest agent is not installed). Once shutdown has been completed perform poweroff action specified in guest configuration.
**Signature:**

```
void clean_shutdown (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to shutdown |

**Return Type:** `void`

**RPC name: clean_reboot**

**Overview:** Attempt to cleanly shutdown the specified VM (Note: this may not be supported—e.g. if a guest agent is not installed). Once shutdown has been completed perform reboot action specified in guest configuration.
**Signature:**

```
void clean_reboot (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to shutdown |

**Return Type:** `void`

**RPC name: hard_shutdown**

**Overview:** Stop executing the specified VM without attempting a clean shutdown. Then perform poweroff action specified in VM configuration.
**Signature:**

```
void hard_shutdown (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to destroy |

**Return Type:** `void`

**RPC name: hard_reboot**

**Overview:** Stop executing the specified VM without attempting a clean shutdown. Then perform reboot action specified in VM configuration
**Signature:**

```
void hard_reboot (session_id s, vm_id vm)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to reboot |

**Return Type:** `void`

**RPC name: hibernate**

**Overview:** Hibernate the specified VM, suspending to disk.
**Signature:**

```
void hibernate (session_id s, vm_id vm, bool live)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to hibernate |
| `bool` | live | If set to true, perform a live hibernate; otherwise suspend the VM before commencing hibernate |

**Return Type:** `void`

**RPC name: unhibernate**

**Overview:** Awaken the specified VM from hibernation and resume it.
**Signature:**

```
void unhibernate (session_id s, vm_id vm, bool start_paused)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `vm_id` | vm | The VM to unhibernate |
| `bool` | start_paused | Unhibernate VM in paused state if set to true. |

**Return Type:** `void`

## 2.5 Class: host

### 2.5.1 Fields for class: host

| Name | **host** | | |
|---|---|---|---|
| Description | *a physical host* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RW$ | `name/label` | string | a short human-readable label |
| $RW$ | `name/short_description` | string | a short human-readable description |
| $RW$ | `name/long_description` | string | a slightly longer human-readable description |
| $RW$ | `name/detail` | string | everything you can think of, and then some more |
| $RO_{run}$ | `software_version` | (string → string) Map | version strings |
| $RW$ | `SRs` | SR ref Set | list of mounted storage repositories |
| $RO_{run}$ | `resident_VMs` | VM ref Set | list of VMs resident on host |
| $RO_{run}$ | `host_CPUs` | host_cpu ref Set | The physical CPUs on this host |

### 2.5.2 Additional RPCs associated with class: host

**RPC name: disable**

**Overview:** Puts the host into a state in which no new VMs can be started.
**Signature:**

```
void disable (session_id s, host_id host)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `host_id` | host | The Host to disable |

**Return Type:** `void`

**RPC name: enable**

**Overview:** Puts the host into a state in which new VMs can be started.
**Signature:**

```
void enable (session_id s, host_id host)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `host_id` | host | The Host to enable |

**Return Type:** `void`

**RPC name: shutdown**

**Overview:** Shutdown the host. (This function can only be called if there are no currently running VMs on the host and it is disabled.)
**Signature:**

```
void shutdown (session_id s, host_id host)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `host_id` | host | The Host to shutdown |

**Return Type:** `void`

**RPC name: reboot**

**Overview:** Reboot the host. (This function can only be called if there are no currently running VMs on the host and it is disabled.)
**Signature:**

```
void reboot (session_id s, host_id host)
```

**Arguments:**

| type | name | description |
|---|---|---|
| `host_id` | host | The Host to reboot |

**Return Type:** `void`

## 2.6 Class: host_cpu

### 2.6.1 Fields for class: host_cpu

| Name | **host_cpu** | | |
|---|---|---|---|
| Description | *a physical CPU* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RO_{ins}$ | `host` | host ref | the host the CPU is in |
| $RO_{ins}$ | `number` | int | the number of the physical CPU within the host |
| $RO_{ins}$ | `features` | cpu_feature Set | the features supported by the CPU |
| $RO_{run}$ | `utilisation` | float | the current CPU utilisation |

### 2.6.2 Additional RPCs associated with class: host_cpu

**Class host_cpu has no additional RPCs associated with it.**

## 2.7 Class: network_manager

### 2.7.1 Fields for class: network_manager

| Name | **network_manager** | | |
|---|---|---|---|
| Description | *creates and manages virtual networks* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| *RW* | `networks` | network ref Set | networks known to the network manager |

### 2.7.2 Additional RPCs associated with class: network_manager

**Class network_manager has no additional RPCs associated with it.**

## 2.8 Class: network

### 2.8.1 Fields for class: network

| Name | **network** | | |
|---|---|---|---|
| Description | *a virtual network* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| *RW* | `name/label` | string | a short human-readable label |
| *RW* | `name/short_description` | string | a short human-readable description |
| *RW* | `name/long_description` | string | a slightly longer human-readable description |
| *RW* | `name/detail` | string | everything you can think of, and then some more |
| *RW* | `VIFs` | VIF ref Set | list of connected vifs |
| *RW* | `NIC` | string | ethernet device to use to access this network. Note: in this revision of the API all hosts will use the specified NIC to access this network |
| *RW* | `VLAN` | string | VLAN tag to use to access this network. Note: in this revision of the API all hosts will use the specified VLAN tag to access this network |
| *RW* | `default_gateway` | string | default gateway IP address. Used for auto-configuring guests with fixed IP setting |
| *RW* | `default_netmask` | string | default netmask. Used for auto-configuring guests with fixed IP setting |

### 2.8.2 Additional RPCs associated with class: network

**Class network has no additional RPCs associated with it.**

## 2.9 Class: VIF

### 2.9.1 Fields for class: VIF

| Name | **VIF** | | |
|---|---|---|---|
| Description | *a virtual network interface* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RW$ | `name` | string | human-readable name of the interface |
| $RW$ | `type` | driver_type | interface type |
| $RW$ | `device` | string | network device to use e.g. eth0 |
| $RW$ | `network` | network ref | virtual network to which this vif is connected |
| $RW$ | `VM` | VM ref | virtual machine to which this vif is connected |
| $RW$ | `MAC` | string | ethernet MAC address |
| $RW$ | `MTU` | int | MTU in octets |
| $RO_{run}$ | `network_read_kbs` | float | Incoming network bandwidth |
| $RO_{run}$ | `network_write_kbs` | float | Outgoing network bandwidth |
| $RW$ | `qos/algorithm_type` | string | QoS algorithm to use |
| $RW$ | `qos/algorithm_params` | string | Paramters for chosen QoS algorithm |
| $RO_{run}$ | `IO_bandwidth/incoming_kbs` | float | Read bandwidth (Kb/s) |
| $RO_{run}$ | `IO_bandwidth/outgoing_kbs` | float | Write bandwidth (Kb/s) |

### 2.9.2 Additional RPCs associated with class: VIF

**Class VIF has no additional RPCs associated with it.**

## 2.10 Class: storage_manager

### 2.10.1 Fields for class: storage_manager

| Name | **storage_manager** | | |
|---|---|---|---|
| Description | *creates and manages storage repositories* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RW$ | `SRs` | SR ref Set | list of currently-known storage repositories |

### 2.10.2 Additional RPCs associated with class: storage_manager

**Class storage_manager has no additional RPCs associated with it.**

## 2.11 Class: SR

### 2.11.1 Fields for class: SR

| | | | |
|---|---|---|---|
| Name | **SR** | | |
| Description | *a storage repository* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RW$ | `name/label` | string | a short human-readable label |
| $RW$ | `name/short_description` | string | a short human-readable description |
| $RW$ | `name/long_description` | string | a slightly longer human-readable description |
| $RW$ | `name/detail` | string | everything you can think of, and then some more |
| $RW$ | `VDIs` | VDI ref Set | managed virtual disks |
| $RO_{run}$ | `total_promised` | int | total amount of space promised to virtual disks |
| $RO_{run}$ | `total_guaranteed` | int | total amount of space guaranteed to virtual disks |
| $RO_{ins}$ | `physical_size` | int | total physical size of the repository |
| $RO_{ins}$ | `type` | string | type? |
| $RO_{ins}$ | `location` | string | location? |
| $RO_{ins}$ | `globally_shared` | bool | true if the repository can be seen by all hosts; otherwise it is considered local |
| $RO_{ins}$ | `local_node` | host ref | host to which this repository is considered to be local |

### 2.11.2 Additional RPCs associated with class: SR

**Class SR has no additional RPCs associated with it.**

## 2.12 Class: VDI

### 2.12.1 Fields for class: VDI

| Name | **VDI** | | |
|---|---|---|---|
| Description | *a virtual disk image* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| *RW* | `name/label` | string | a short human-readable label |
| *RW* | `name/short_description` | string | a short human-readable description |
| *RW* | `name/long_description` | string | a slightly longer human-readable description |
| *RW* | `name/detail` | string | everything you can think of, and then some more |
| *RW* | `SR` | SR ref | storage repository to create the disk on |
| *RW* | `VBDs` | VBD ref Set | list of vbds which have mounted this disk |
| $RO_{run}$ | `filesystem` | filesystem ref | information about the filesystem (if known) |
| $RO_{ins}$ | `virtual_size` | int | size of disk to present to the guest |
| $RO_{run}$ | `guaranteed_size` | int | amount of space guaranteed by the storage repository |
| $RO_{ins}$ | `type` | vdi_type | type of the VDI |
| $RO_{ins}$ | `parent` | VDI ref | parent disk (e.g. in the case of copy on write |
| $RO_{ins}$ | `children` | VDI ref Set | child disks (e.g. in the case of copy on write |
| *RW* | `sharable` | bool | true if this disk may be shared |
| $RO_{run}$ | `creator` | user ref | person who created this disk |
| $RO_{run}$ | `creation_time` | datetime | time and date VDI was created |
| $RO_{run}$ | `last_mounted` | datetime | time the VDI was last mounted by a guest |

### 2.12.2 Additional RPCs associated with class: VDI

**Class VDI has no additional RPCs associated with it.**

## 2.13 Class: VBD

### 2.13.1 Fields for class: VBD

| | | | |
|---|---|---|---|
| Name | **VBD** | | |
| Description | *a virtual block device* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RW$ | `VM` | VM ref | the virtual machine |
| $RW$ | `VDI` | VDI ref | the virtual disk |
| $RW$ | `device` | string | device seen by the guest e.g. hda1 |
| $RW$ | `mode` | vbd_mode | the mode the disk should be mounted with |
| $RW$ | `driver` | driver_type | the style of driver |
| $RW$ | `qos/algorithm_type` | string | QoS algorithm to use |
| $RW$ | `qos/algorithm_params` | string | Paramters for chosen QoS algorithm |
| $RO_{run}$ | `IO_bandwidth/incoming_kbs` | float | Read bandwidth (Kb/s) |
| $RO_{run}$ | `IO_bandwidth/outgoing_kbs` | float | Write bandwidth (Kb/s) |

### 2.13.2 Additional RPCs associated with class: VBD

**Class VBD has no additional RPCs associated with it.**

## 2.14 Class: filesystem

### 2.14.1 Fields for class: filesystem

| Name | **filesystem** | | |
|---|---|---|---|
| Description | *an on-disk filesystem* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RO_{run}$ | `block_size` | int | block size |
| $RO_{run}$ | `total_blocks` | int | total blocks on disk |
| $RO_{run}$ | `available_blocks` | int | blocks available for allocation |
| $RO_{run}$ | `used_blocks` | int | blocks already in use |
| $RO_{run}$ | `percentage_free` | int | Percentage of free space left in filesystem |
| $RO_{run}$ | `type` | string | filesystem type |

### 2.14.2 Additional RPCs associated with class: filesystem

**Class filesystem has no additional RPCs associated with it.**

## 2.15 Class: user

### 2.15.1 Fields for class: user

| Name | **user** | | |
|---|---|---|---|
| Description | *a user of the system* | | |
| Quals | Field | Type | Description |
| $RO_{run}$ | `uuid` | string | globally-unique ID |
| $RO_{ins}$ | `short_name` | string | short name (e.g. userid) |
| *RW* | `fullname` | string | full name |
| *RW* | `created_VDIs` | VDI ref Set | the VDIs this user has created |

### 2.15.2 Additional RPCs associated with class: user

**Class user has no additional RPCs associated with it.**

## 2.16 DTD

General notes:

- Values of primitive types (int, bool, etc) and higher-order types (Sets, Maps) are encoded as simple strings, rather than being expanded into XML fragments. For example “5”, “true”, “1, 2, 3, 4”, “(1, 2), (2, 3), (3, 4)”
- Values of enumeration types are represented as strings (e.g. “PAE”, “3DNow!”)
- Object References are represented as UUIDs, written in string form

```
<!ELEMENT networks (#PCDATA)>
<!ELEMENT guaranteed_size (#PCDATA)>
<!ELEMENT long_description (#PCDATA)>
<!ELEMENT PCI_bus (#PCDATA)>
<!ELEMENT required (#PCDATA)>
<!ELEMENT dynamic_max (#PCDATA)>
<!ELEMENT coredump (#PCDATA)>
<!ELEMENT boot_method (#PCDATA)>
<!ELEMENT TPM (instance, backend)>
<!ELEMENT network_manager (uuid, networks)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT MTU (#PCDATA)>
<!ELEMENT running_on (#PCDATA)>
<!ELEMENT physical_size (#PCDATA)>
<!ELEMENT total_promised (#PCDATA)>
<!ELEMENT total_guaranteed (#PCDATA)>
<!ELEMENT total_blocks (#PCDATA)>
<!ELEMENT crash (#PCDATA)>
<!ELEMENT name ((#PCDATA) | (label, short_description, long_description, detail))>
<!ELEMENT algorithm_params (#PCDATA)>
<!ELEMENT cdrom (#PCDATA)>
<!ELEMENT kernel ((kernel, initrd, args) | (#PCDATA))>
<!ELEMENT builder (#PCDATA)>
<!ELEMENT uuid (#PCDATA)>
```

```
<!ELEMENT user (uuid, short_name, fullname, created_VDIs)>
<!ELEMENT user_version (#PCDATA)>
<!ELEMENT IO_bandwidth (incoming_kbs, outgoing_kbs)>
<!ELEMENT VIFs (#PCDATA)>
<!ELEMENT force_off (#PCDATA)>
<!ELEMENT VM ((#PCDATA) | (uuid, name, user_version, is_a_template, running_on, memory,
VCPUs, actions, coredump, VIFs, VBDs, TPM, bios, platform, builder, console_port, boot_method,
kernel, grub, PCI_bus, tools_version))>
<!ELEMENT globally_shared (#PCDATA)>
<!ELEMENT VNC (#PCDATA)>
<!ELEMENT platform (std_VGA, SDL, VNC, serial, localtime, clock_offset, enable_audio)>
<!ELEMENT policy (#PCDATA)>
<!ELEMENT standby (#PCDATA)>
<!ELEMENT actual (#PCDATA)>
<!ELEMENT host_cpu (uuid, host, number, features, utilisation)>
<!ELEMENT SRs (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT actions (power_off, reboot, crash, standby)>
<!ELEMENT VLAN (#PCDATA)>
<!ELEMENT VIF (uuid, name, type, device, network, VM, MAC, MTU, network_read_kbs, network_write_kbs,
qos, IO_bandwidth)>
<!ELEMENT creation_time (#PCDATA)>
<!ELEMENT algorithm_type (#PCDATA)>
<!ELEMENT network_read_kbs (#PCDATA)>
<!ELEMENT VDIs (#PCDATA)>
<!ELEMENT dynamic_min (#PCDATA)>
<!ELEMENT resident_VMs (#PCDATA)>
<!ELEMENT storage_manager (uuid, SRs)>
<!ELEMENT cmdline (#PCDATA)>
<!ELEMENT power_off (#PCDATA)>
<!ELEMENT used_blocks (#PCDATA)>
<!ELEMENT tools_version (#PCDATA)>
<!ELEMENT sharable (#PCDATA)>
<!ELEMENT VDI ((#PCDATA) | (uuid, name, SR, VBDs, filesystem, virtual_size, guaranteed_size,
type, parent, children, sharable, creator, creation_time, last_mounted))>
<!ELEMENT outgoing_kbs (#PCDATA)>
<!ELEMENT static_max (#PCDATA)>
<!ELEMENT features ((#PCDATA) | (required, can_use, force_on, force_off))>
<!ELEMENT incoming_kbs (#PCDATA)>
<!ELEMENT percentage_free (#PCDATA)>
<!ELEMENT memory (static_max, dynamic_max, actual, dynamic_min, static_min)>
<!ELEMENT initrd (#PCDATA)>
<!ELEMENT last_mounted (#PCDATA)>
<!ELEMENT bios (cdrom, boot)>
<!ELEMENT device (#PCDATA)>
<!ELEMENT MAC (#PCDATA)>
<!ELEMENT default_gateway (#PCDATA)>
<!ELEMENT qos (algorithm_type, algorithm_params)>
<!ELEMENT software_version (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT instance (#PCDATA)>
<!ELEMENT short_name (#PCDATA)>
<!ELEMENT params (#PCDATA)>
<!ELEMENT serial (#PCDATA)>
```

```
<!ELEMENT boot (#PCDATA)>
<!ELEMENT console_port (#PCDATA)>
<!ELEMENT short_description (#PCDATA)>
<!ELEMENT NIC (#PCDATA)>
<!ELEMENT virtual_size (#PCDATA)>
<!ELEMENT children (#PCDATA)>
<!ELEMENT driver (#PCDATA)>
<!ELEMENT host ((#PCDATA) | (uuid, name, software_version, SRs, resident_VMs, host_CPUs))>
<!ELEMENT is_a_template (#PCDATA)>
<!ELEMENT force_on (#PCDATA)>
<!ELEMENT local_node (#PCDATA)>
<!ELEMENT std_VGA (#PCDATA)>
<!ELEMENT network_write_kbs (#PCDATA)>
<!ELEMENT localtime (#PCDATA)>
<!ELEMENT args (#PCDATA)>
<!ELEMENT parent (#PCDATA)>
<!ELEMENT network ((#PCDATA) | (uuid, name, VIFs, NIC, VLAN, default_gateway, default_netmask))>
<!ELEMENT VBDs (#PCDATA)>
<!ELEMENT clock_offset (#PCDATA)>
<!ELEMENT static_min (#PCDATA)>
<!ELEMENT grub (cmdline)>
<!ELEMENT fullname (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT SDL (#PCDATA)>
<!ELEMENT VCPUs (policy, params, utilisation, features)>
<!ELEMENT host_CPUs (#PCDATA)>
<!ELEMENT created_VDIs (#PCDATA)>
<!ELEMENT reboot (#PCDATA)>
<!ELEMENT available_blocks (#PCDATA)>
<!ELEMENT enable_audio (#PCDATA)>
<!ELEMENT can_use (#PCDATA)>
<!ELEMENT SR ((#PCDATA) | (uuid, name, VDIs, total_promised, total_guaranteed, physical_size,
type, location, globally_shared, local_node))>
<!ELEMENT block_size (#PCDATA)>
<!ELEMENT utilisation (#PCDATA)>
<!ELEMENT filesystem ((uuid, block_size, total_blocks, available_blocks, used_blocks,
percentage_free, type) | (#PCDATA))>
<!ELEMENT mode (#PCDATA)>
<!ELEMENT VBD (uuid, VM, VDI, device, mode, driver, qos, IO_bandwidth)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT backend (#PCDATA)>
<!ELEMENT default_netmask (#PCDATA)>
<!ELEMENT detail (#PCDATA)>
```

### 2.16.1 XML configuration examples

### 2.16.2 A virtual network interface

```
<vif>
    <name = "virtual interface 0">
    <type = "paravirtualised">
    <device = "eth0">
    <!-- The virtual machine to which the vif is attached -->
    <vm = "uuid-1579-1324-1597-2911">

    <!-- The virtual network the vif is attached to
         (This tells the host on which the guest is running
          how packets to/from this vif should be dealt with) -->
    <network = "uuid-1234-5678-1234-5678">

    <!-- VIF parameters -->
    <mac = "00:11:22:33:44:55">
    <mtu = "1500">

    <!-- Specify leaky bucket qos and give parameters -->
    <qos>
       <algorithm_type = "leaky_bucket">
       <algorithm_params = "r=0.15, b=3">
    </qos>
</vif>
```

### 2.16.3 A paravirtualised guest

```
<vm>
  <name>
    <label>debian</label>
    <shortDescription>Debian etch unstable</shortDescription>
    <longDescription>A completely vanilla install of Debian etch unstable</longDescription>
    <detail>I could write a long description here, including versions of all the software
                  installed, all the licenses, everything I can think of.</detail>
  </name>
  <memory>
    <staticmax>1G</staticmax>
    <staticmin>128M</staticmin>
    <!-- actual is a runtime property -->
    <dynamicmax>512M</dynamicmax>
    <dynamicmin>128M</dynamicmin>
  </memory>
  <vcpus>

    <policy="variable_within_range">
    <params="range=(0,5)">

    <features>
      <required> FPU, P4 </required>
      <canuse> SSE, 3DNow! </canuse>
      <force_on> SSE </force_on>
      <force_off> 3DNow! </force_off>
    </features>
  </vcpus>

  <!-- set how guest behaves on poweroff, reboot, crash, standby -->
  <actions>
    <poweroff> destroy </poweroff>
    <reboot> restart </reboot>
    <crash> preserve </crash>
    <standby> rename-restart </standby>
  </actions>

  <!-- generate coredump on poweroff, reboot, crash, standby -->
  <coredump> true </coredump>

  <!-- attach following vifs and vdis to guest on startup -->
  <vifs> uuid-1234-5678-1234-5678, uuid-2345-6789-2345-6789 </vifs>
  <vdis> uuid-1234-5678-1234-5678, uuid-2345-6789-2345-6789 </vdis>

  <tpm>
    <instance>0</instance>
    <backend>0</backend>
  </tpm>

  <bios>
    <cdrom>/dev/cdrom</cdrom>
    <boot>cdrom</boot>
  </bios>

  <platform>
    <stdvga>false</stdvga>
    <sdl>false</sdl>
    <vnc>true</vnc>
    <serial>/dev/pty1</serial>
    <localtime>true</localtime>
    <enable_audio>true</enable_audio>
  </platform>

  <builder> /domain/builder/part/of/virtual/bios </builder>

  <console_port>1234</console_port>

  <boot_method>kernel_internal</boot_method>
  <kernel>
    <kernel>/boot/vmlinuz</kernel>
    <initrd>/boot/initrd.img</initrd>
    <args>root=/dev/nfs ... </args>
  </kernel>

  <grub>
    <cmdline></cmdline> <!-- not used by this boot_method -->
  </grub>
  <pci_bus></pci_bus> <!-- no devices to pass-through -->
  <!-- toolsVersion is a runtime property -->
</vm>
```